

IBM Spectrum LSF 10.1

Administering



Tables of Contents

Cluster overview	1
Terms and concepts	1
Cluster characteristics	4
File systems, directories, and files	5
Example directory structures	5
UNIX and Linux	6
Microsoft Windows	6
Important directories and configuration files	7
Work with LSF	10
Start, stop, and reconfigure LSF	10
Setting up the LSF environment	11
Starting your cluster	11
Stopping your cluster	12
Reconfiguring your cluster	12
Check LSF status	12
Check cluster configuration	13
Check cluster status	14
Check LSF batch system configuration	15
Find batch system status	15
Run jobs	16
Submit batch jobs	17
Display job status	18
Control job execution	18
Run interactive tasks	19
Integrate your applications with LSF	19
Manage users, hosts, and queues	20
Making your cluster available to users	20
Adding a host to your cluster	21
Removing a host from your cluster	23
Adding a queue	24
Removing a queue	25
Configure LSF startup	26
Allowing LSF administrators to start LSF daemons	26
Setting up automatic LSF startup	27
Manage software licenses and other shared resources	27
Troubleshooting LSF problems	30
Solving common LSF problems	30
LSF error messages	36
Administer LSF	42
Cluster management essentials	42
Work with your cluster	43
Viewing cluster information	43
Control LSF daemons	47
Controlling mbatchd	50
LSF daemon startup control	51
Overview	52
Configuration to enable	54
LSF daemon startup control behavior	55
Configuration to modify	56
Commands	56
Commands to reconfigure your cluster	57
Reconfiguring with the lsadmin and badmin commands	58
Reconfiguring by restarting the mbatchd daemon	58

Viewing configuration errors	59
Live reconfiguration	59
bconf command authentication	60
Enabling live reconfiguration	60
Adding a user share to a fair share queue	60
View bconf records	61
Merge configuration files	62
Adding cluster administrators	63
Working with hosts	64
Host status	64
View host information	65
Customize host information output	69
Customize host load information output	71
Controlling hosts	73
Connect to an execution host or container	76
Host names	76
Hosts with multiple addresses	78
Using IPv6 addresses	80
Specifying host names with condensed notation	81
Job directories and data	82
Directory for job output	83
Specifying a directory for job output	83
Temporary job directories	83
About flexible job CWD	84
About flexible job output directory	84
Job notification	85
Disabling job email	86
Size of job email	87
Monitoring cluster operations and health	87
Monitor cluster performance	88
Monitor performance metrics in real time	88
Enabling daemon log files for diagnostics	91
Diagnose scheduler buckets	92
Monitor scheduler efficiency and overhead	92
Monitor job information	93
Viewing host-level and queue-level suspending conditions	93
Viewing job-level suspending conditions	93
Viewing resume thresholds	93
View job priority information	94
Viewing job dependencies	94
View information about backfill jobs	95
Viewing information about job start time	95
Viewing the run limits for interruptible backfill jobs (bjobs and bhst)	96
Displaying available slots for backfill jobs	96
Viewing job array information	98
View information about reserved job slots	100
Viewing configured job slot share	100
Viewing slot allocation of running jobs	100
Monitor applications by using external scripts	101
Create external scripts	101
Configure the application profiles	102
Use the application profiles	103
View resource information	103
Viewing job-level resource requirements	104
Viewing queue-level resource requirements	104

Viewing shared resources for hosts	105
Viewing load on a host	105
Viewing job resource usage	105
View cluster resources (lsinfo)	106
Viewing host resources (lshosts)	106
Viewing host load by resource (lshosts -s)	106
Customize host resource information output	107
View resource reservation information	108
Viewing host-level resource information (bhosts)	108
Viewing queue-level resource information (bqueues)	109
Viewing reserved memory for pending jobs (bjobs)	109
Viewing per-resource reservation (bresources)	110
View information about resource allocation limits	110
View application profile information	111
Viewing available application profiles	112
View fair share information	114
View queue-level fair share information	114
Viewing cross-queue fair share information	114
Viewing hierarchical share information for a group	115
Viewing hierarchical share information for a host partition	115
Viewing host partition information	116
Viewing information about SLAs and service classes	116
Monitoring an SLA	119
Viewing configured guaranteed resource pools	122
Viewing guarantee policy information	123
View user and user group information	125
Viewing user information	126
Viewing user pending job threshold information	126
Customize user information output	126
Viewing user group information	127
Viewing user share information	127
Viewing user group admin information	127
View queue information	128
Queue states	128
Viewing available queues and queue status	129
Viewing detailed queue information	129
Customize queue information output	130
Viewing the state change history of a queue	132
Viewing queue administrators	132
Viewing exception status for queues (bqueues)	132
Managing job execution	133
Managing job execution	133
About job states	134
View job information	137
Viewing all jobs for all users	137
View job IDs	138
Viewing jobs for specific users	138
Viewing running jobs	138
Viewing done jobs	138
Viewing pending job information	138
Viewing job suspend reasons	139
Viewing post-execution states	139
Viewing exception status for jobs (bjobs)	139
Viewing unfinished job summary information	140
View the job submission environment	140

Customize job information output	141
Force job execution	150
Forcing a pending job to run	150
Suspend and resume jobs	150
Suspending a job	151
Resuming a job	151
Kill jobs	151
Killing a job	152
Killing multiple jobs	152
Killing jobs by status	153
Killing and recording jobs as DONE status	153
Forcefully removing a job from LSF	154
Removing hung jobs from LSF	154
Orphan job termination	155
Send a signal to a job	158
Signals on different platforms	158
Sending a signal to a job	158
Data provenance	159
Prerequisites	159
Using data provenance tools	159
Job file spooling	160
File spooling for job input, output, and command files	161
Specifying a job input file	161
Changing the job input file	162
Job spooling directory (JOB_SPOOL_DIR)	162
Specifying a job command file (bsub -Zs)	162
Remote file access with non-shared file space	163
Copying files from the submission host to execution host	163
Specifying an input file	164
Copying output files back to the submission host	164
Job submission option files	164
Specifying a JSON file	165
Specifying a YAML file	165
JSDL files with job submission options	166
Job data management	166
Copy a file to a remote host (bsub -f)	167
Use LSF Data Manager for data staging	167
Use direct data staging (bsub -stage)	168
Submitting and running direct data staging jobs	168
Configuring direct data staging	170
Job scheduling and dispatch	170
Use exclusive scheduling	171
Configuring an exclusive queue	172
Configuring a host to run one job at a time	172
Submitting an exclusive job	172
Configuring a compute unit exclusive queue	172
Submitting a compute unit exclusive job	172
Job dependency and job priority	172
Job dependency scheduling	173
Job dependency terminology	174
Dependency conditions	174
Job priorities	176
User-assigned job priority	176
Configuring job priority	177
Specifying job priority	177

Automatic job priority escalation	177
Configuring job priority escalation	178
Absolute priority scheduling	178
Enabling absolute priority scheduling	181
Modifying the system APS value (bmod)	182
Configuring APS across multiple queues	183
Job priority behavior	185
Job re-queue and job rerun	187
About job re-queuing	187
Automatic job re-queuing	188
Configuring automatic job re-queuing	188
Configuring job-level automatic re-queuing	189
Configuring reverse re-queuing	190
Exclusive job re-queuing	190
Configuring exclusive job re-queuing	191
Re-queuing a job	191
Automatic job reruns	192
Configuring queue-level job reruns	192
Submitting a re-runnable job	193
Disabling a job from re-running	193
Disabling post-execution for re-runnable jobs	193
Job start time prediction	193
Job affinity scheduling	
Job affinity scheduling with host attributes	195
Configuring host attributes for job affinity	195
Managing host attributes for job affinity	196
Submitting jobs with host attributes for job affinity	196
Control job execution	197
Pre-execution and post-execution processing	198
About pre- and post-execution processing	198
Configuration to enable pre- and post-execution processing	200
Pre- and post-execution processing behavior	202
Checking job history for a pre-execution script failure	205
Configuration to modify pre- and post-execution processing	205
Set host exclusion based on job-based pre-execution scripts	210
Pre- and post-execution processing commands	211
Job starters	213
About job starters	213
Command-level job starters	214
Queue-level job starters	215
Configuring a queue-level job starter	215
JOB_STARTER parameter (lsb.queues)	216
Control the execution environment with job starters	216
Job control actions	217
Submit jobs as other users	223
External job submission and execution controls	223
Job submission and execution controls	224
Configuration to enable job submission and execution controls	229
Job submission and execution controls behavior	236
Configuration to modify job submission and execution controls	238
Job submission and execution controls commands	239
Command arguments for job submission and execution controls	240
Interactive jobs and remote tasks	241
Interactive jobs with bsub	241

About interactive jobs	242
Submit interactive jobs	242
Submitting an interactive job	243
Submitting an interactive job by using a pseudo-terminal	243
Submitting an interactive job and redirect streams to files	244
Submitting an interactive job, redirect streams to files, and display streams	245
Performance tuning for interactive batch jobs	245
Interactive batch job messaging	247
Configuring interactive batch job messaging	247
Example messages	248
Run X applications with bsub	249
Configuring SSH X11 forwarding for jobs	249
Write job scripts	249
Register utmp file entries for interactive batch jobs	251
Interactive and remote tasks	252
Run remote tasks	252
Running a task on the best available host	252
Running a task on a host with specific resources	253
Resource usage	253
Running a task on a specific host	253
Running a task by using a pseudo-terminal	254
Running the same task on many hosts in sequence	254
Running parallel tasks	254
Running tasks on hosts specified by a file	254
Interactive tasks	254
Redirecting streams to files	256
Load sharing interactive sessions	256
Logging on to the least loaded host	256
Logging on to a host with specific resources	257
Configuring and sharing job resources	257
About LSF resources	257
Resource categories	258
How LSF uses resources	259
Representing job resources in LSF	259
Batch built-in resources	260
Static resources	261
How LIM detects cores, threads, and processors	263
Defining ncpus: processors, cores, or threads	264
Defining computation of ncpus on dynamic hosts	265
Defining computation of ncpus on static hosts	266
Load indices	266
About configured resources	269
Adding new resources to your cluster	270
Configuring the lsf.shared resource section	270
Configuring lsf.cluster.cluster_name Host section	271
Configuring lsf.cluster.cluster_name ResourceMap section	272
Reserving a static shared resource	273
External load indices	273
About external load indices	274
Configuration to enable external load indices	275
Define a dynamic external resource	276
Map an external resource	276
Create an elim executable file	277
Overriding built-in load indices	278
Setting up an ELIM to support JSDL	278

Example of an elim executable file	279
External load indices behavior	279
Configuration to modify external load indices	281
External load indices commands	281
External static load indices	282
Configuration to enable external static load indices	283
Create eslim executable files	283
Example of an eslim executable file	284
Modify a built-in load index	286
Configure host resources	286
Adding a host to your cluster	286
Dynamically adding hosts	288
Configuring and running batch jobs on dynamic hosts	289
Changing a dynamic host to a static host	290
Adding a dynamic host in a shared file system environment	290
Adding a dynamic host in a non-shared file system environment	291
Adding a host to the cluster using bconf	293
Removing a host from your cluster	294
Removing a host from management candidate list	294
Removing dynamic hosts	295
Share resources in queues	295
Controlling queues	296
Closing a queue	296
Opening a queue	297
Deactivating a queue	297
Activating a queue	297
Logging a comment on a queue control command	297
Configuring dispatch windows	298
Configuring run windows	299
Adding a queue	24
Removing a queue	25
Restricting which hosts can use queues	301
Restricting job size requested by parallel jobs in a queue	302
Adding queue administrators	302
Change job order within queues	303
Switch jobs from one queue to another	304
Switching a single job to a different queue	304
Switching all jobs to a different queue	304
Use external job switch controls	305
Configuration to enable job switch controls	305
Configuration to modify job switch controls	306
Command arguments for job switch controls	306
Application profiles	306
Manage application profiles	307
Add an application profile	307
Submitting jobs to application profiles	308
How application profiles interact with queue and job parameters	309
Application profile settings that override queue settings	310
Application profile limits and queue limits	311
Define application-specific environment variables	311
Task limits	312
Absolute run limits	312
Pre-execution	312
Post-execution	313

Re-runnable jobs	313
Resource requirements	313
Estimated job run time and runtime limits	314
Plan-based scheduling and reservations	317
Enabling plan-based scheduling	317
Plan-based allocations	318
Plan-based scheduling run time	320
Plan-based scheduling limits and prioritization	321
Configuring extendable run limits	321
Reserving resources for an allocation plan	322
Canceling planned allocations	323
Delaying planning for jobs	323
Limiting the number of planned jobs	323
Adjusting the plan window	323
Distributing job resources to users in LSF	324
Configure resource consumers	324
User groups	325
User groups in LSF	325
How to define user groups	325
Where to configure user groups	325
Configuring user groups	326
Configuring user group administrators	326
Configuring user group administrator rights	327
Import external user groups (egroup)	328
Existing user groups as LSF user groups	328
External host and user groups	329
About external host and user groups	329
Configuration to enable external host and user groups	331
External host and user groups behavior	332
Between-host user Account mapping	332
About between-host user account mapping	333
Configuration to enable between-host user account mapping	334
Between-host user account mapping behavior	335
Between-host user account mapping commands	336
Cross-cluster user account mapping	337
About cross-cluster user account mapping	337
Configuration to enable cross-cluster user account mapping	338
Cross-cluster user account mapping behavior	338
Cross-cluster user account mapping commands	339
UNIX and Windows user account mapping	340
About UNIX and Windows user account mapping	340
Configuration to enable UNIX and Windows user account mapping	342
UNIX and Windows user account mapping behavior	343
Configuration to modify UNIX and Windows user account mapping behavior	343
UNIX and Windows user account mapping commands	344
Creating a user group using bconf	345
Job groups	346
Job group limits	348
Creating a job group	349
Submitting jobs under a job group	350
Viewing information about job groups (bjgroup)	350
Viewing jobs for a specific job group (bjobs)	351
Job groups and time-based SLAs	352
Viewing job groups attached to a time-based SLA (bjgroup)	352

Control jobs in job groups	352
Suspending jobs (bstop)	352
Resuming suspended jobs (bresume)	353
Moving jobs to a different job group (bmod)	353
Terminating jobs (bkill)	354
Deleting a job group manually (bgdel)	354
Modifying a job group limit (bgmod)	355
Automatic job group cleanup	355
Host groups	356
Configuring host groups	356
Wildcard and special characters to define host names	357
Define condensed host groups	358
Specifying resource requirements	359
About resource requirements	359
Queue-level resource requirements	361
Job-level resource requirements	362
Resource requirement strings	363
Selection string	369
Order string	375
Usage string	378
Span string	385
Same string	389
Compute unit string	390
Affinity string	393
Specify GPU resource requirements	395
Reserving resources	395
About resource reservation	396
Use resource reservation	397
Configuring resource reservation at the queue level	397
Specifying job-level resource reservation	398
Configuring per-resource reservation	398
Memory reservation for pending jobs	398
Reserving host memory for pending jobs	399
Enabling memory reservation for sequential jobs	399
Configuring lsb Queues	399
Using memory reservation for pending jobs	399
How memory reservation for pending jobs works	400
Time-based slot reservation	401
Configuring time-based slot reservation	403
Assumptions and limitations	405
Reservation scenarios	407
Examples	407
Limiting job resource allocations	408
How resource allocation limits work	409
How job limits work	412
Configuring resource allocation limits	415
Enabling resource allocation limits	416
Configuring cluster-wide limits	416
Limit conflicts	416
How resource allocation limits map to pre-version 7 job slot limits	417
Creating a limit using bconf	418
Updating a limit using bconf	419
Make sure resources are distributed fairly	419
Runtime resource usage limits	420

About resource usage limits	420
Changing the units for resource usage limits	422
Specifying resource usage limits	423
Default run limits for backfill scheduling	424
Specifying job-level resource usage limits	424
Resource usage limits syntax	425
CPU time limit	425
Normalized CPU time	426
Data segment size limit	426
File size limit	426
Memory limit	427
Memory limit enforcement	427
Smart memory limit enforcement	427
OS memory limit enforcement	428
Process limit	428
Runtime limit	428
Normalized run time	429
LSF multicluster capability runtime limit	429
Thread limit	429
Stack limit	430
Swap limit	430
Linmit examples	431
CPU time and run time normalization	431
Memory and swap limit enforcement based on Linux cgroups	432
PAM resource limits	434
Configuring a PAM file	434
Load thresholds	435
Automatic job suspension	435
Suspending conditions	436
Configuring suspending conditions at queue level	437
About resuming suspended jobs	438
Specifying resume condition	438
Time configuration	438
Time windows	439
Time expressions	440
Automatic time-based configuration	440
Dispatch and run windows	442
Run windows	442
Configuring run windows	442
Viewing information about run windows	443
Dispatch windows	443
Configuring host dispatch windows	443
Configuring queue dispatch windows	443
Displaying host dispatch windows	444
Displaying queue dispatch windows	444
Deadline constraint scheduling	444
Disabling deadline constraint scheduling	444
Preemptive scheduling	445
Resource preemption	445
About resource preemption	445
Requirements for resource preemption	446
Custom job controls for resource preemption	447
Preempting resources	447
Configuring resource preemption	448

Memory preemption	449
About preemptive scheduling	450
Configuration to enable preemptive scheduling	452
Preemptive scheduling behavior	452
Configuration to modify preemptive scheduling behavior	455
Preemptive scheduling commands	458
Goal-oriented SLA-driven scheduling	459
Using goal-oriented SLA scheduling	460
Service classes for SLA scheduling	461
Configure service classes using bconf	464
Time-based service classes	464
Configuring time-based service classes	466
Time-based SLA examples	467
Configuring the SLA CONTROL_ACTION parameter (lsb.serviceclasses)	470
Submitting jobs to a service class	470
Modifying SLA jobs (bmod)	471
Global resources	471
Global resource collection	472
Configuring	472
Using	474
GPU resources	475
Enabling GPU features	475
Automatic GPU configuration	475
Enabling jobs to use GPU resources	476
Optimizing GPU resource metric collection	476
Nvidia Data Center GPU Manager (DCGM) features	477
GPU access enforcement	477
Decreasing GPU power consumption when a GPU is not in use	478
Nvidia Multi-Instance GPU (MIG) features	478
Monitoring GPU resources	479
Monitor GPU resources with lsload command	479
Monitor GPU resources with lshosts command	479
Submitting and monitoring GPU jobs	480
Configuring GPU resource requirements	480
Submitting jobs that require GPU resources	484
Monitoring GPU jobs	488
Example GPU job submissions	489
GPU features using ELIM	490
Manually configure and use GPU resources (legacy ELIM procedure)	491
Controlling GPU auto-boost	495
Configuring containers	495
LSF with Docker	496
Preparing LSF to run Docker jobs	497
Configuring LSF to run Docker jobs	498
Configuring LSF to run NVIDIA Docker jobs	499
Submitting Docker jobs	500
Submitting NVIDIA Docker jobs	501
LSF with Shifter	503
Configuring LSF to run Shifter jobs	503
Submitting Shifter jobs	505
LSF with Singularity	505
Configuring LSF to run Singularity jobs	506
Submitting Singularity jobs to LSF	507
LSF with Podman	508
Preparing LSF to run Podman jobs	508

Configuring LSF to run jobs in Podman containers	511
Submitting Podman jobs	512
LSF with Enroot	513
Configuring LSF to run jobs in Enroot containers	513
Submitting Enroot jobs	514
High throughput workload administration	515
Job packs	515
Job arrays	517
Creating a job array	518
Input and output files	519
Preparing input files	519
Passing arguments on the command line	520
Setting a whole array dependency	520
Controlling job arrays	521
Re-queuing jobs in DONE state	522
Job array job slot limit	523
Setting a job array slot limit at submission	523
Fair share scheduling	524
Fair share scheduling	525
Ways to configure fair share	526
Chargeback fair share	526
Configuring chargeback fair share	526
Equal share	527
Configuring equal share	527
Priority user and static priority fair share	527
Configuring priority user fair share	528
Configuring static priority fair share	528
Host partition fair share	529
Configuring host partition fair share	529
GPU runtime fair share	529
Configuring GPU run time	529
User-based fair share	530
Configure hierarchical fair share	531
Configuring a share tree	531
User share assignments	532
Dynamic user priority	533
Use time decay and committed run time	536
Historical run time decay	536
Configuring historical run time	536
How mbatchd reconfiguration and restart affects historical run time	537
Run time decay	537
Configuring run time decay	538
Committed run time weighting factor	538
Configuring committed run time	538
How fair share affects job dispatch order	540
Host partition user-based fair share	541
Configuring host partition fair share scheduling	541
Queue-level user-based fair share	541
Configuring queue-level fair share	542
Cross-queue user-based fair share	542
Configuring cross-queue fair share	543
Control job dispatch order in cross-queue fair share	544
Queue-based fair share	544
Slot allocation per queue	546
Configuring slot allocation per queue	546

Typical slot allocation scenarios	548
Users affected by multiple fair share policies	553
Submitting a job and specify a user group	553
Re-sizable jobs and fair share	553
Guaranteed resource pools	554
About guaranteed resources	554
Configuration overview of guaranteed resource pools	555
Submitting jobs to use guarantees	559
Package guarantees	560
Adding consumers to a guaranteed resource pool	561
Reserving memory and license resources	562
Memory reservation for pending jobs	398
Reserving host memory for pending jobs	399
Enabling memory reservation for sequential jobs	399
Configuring lsb.queues	399
Using memory reservation for pending jobs	399
How memory reservation for pending jobs works	400
Reserving license resources	564
Parallel workload administration	565
Running parallel jobs	565
How LSF runs parallel jobs	566
Preparing your environment to submit parallel jobs to LSF	567
Using a job starter	567
Submitting a parallel job	567
Starting parallel tasks with LSF utilities	568
Job slot limits for parallel jobs	569
Specify a minimum and maximum number of tasks	569
Restricting job size requested by parallel jobs	570
About specifying a first execution host	571
Specifying a first execution host	572
Rules	572
Compute units	573
Control job locality using compute units	574
Configuring compute units	582
Wildcard and special characters to define names in compute units	583
Define condensed compute units	583
Import external host groups (egroup)	584
Use compute units with advance reservation	584
Control processor allocation across hosts	584
Run parallel processes on homogeneous hosts	587
Limit the number of processors allocated	588
Limit the number of allocated hosts	590
Reserve processors	591
Configuring processor reservation	592
Reserve memory for pending parallel jobs	592
Configuring memory reservation for pending parallel jobs	592
Enabling per-task memory reservation	593
Backfill scheduling	593
Configuring a backfill queue	595
Enforce run limits	595
Use backfill on memory	596
Use interruptible backfill	597
Configuring an interruptible backfill queue	599
Submitting backfill jobs according to available slots	599
How deadline constraint scheduling works for parallel jobs	599

Optimized preemption of parallel jobs	600
Configuring optimized preemption	600
Controlling CPU and memory affinity	600
Submit affinity jobs	602
Submit affinity jobs for IBM POWER8 systems	609
Managing jobs with affinity resource requirements	612
Affinity preemption	616
Affinity binding based on Linux cgroup cpuset subsystem	617
Portable hardware locality	618
Processor binding for LSF job processes	619
Enabling processor binding for LSF job processes	623
Processor binding for parallel jobs	624
Running parallel jobs with blaunch	624
blaunch distributed application framework	625
SGI vendor MPI support	631
Running jobs with task geometry	631
Enforcing resource usage limits for parallel tasks	633
Running MPI workload through IBM Parallel Environment Runtime Edition	634
Enabling IBM PE Runtime Edition for LSF	634
Network-aware scheduling	635
Submitting IBM Parallel Environment jobs through LSF	637
Managing IBM Parallel Environment jobs through LSF	637
Advance reservation	639
Types of advance reservations	639
Enable advance reservation	640
Allow users to create advance reservations	641
Use advance reservation	642
Adding reservations	643
Changing reservations	650
Removing reservations	655
Viewing reservations	655
Submitting and modifying jobs that use advance reservations	659
Viewing jobs that are associated with an advance reservation	660
Advance reservation behavior and operations	661
Fair share scheduling	524
Fair share scheduling	525
Parallel fair share	665
Configuring parallel fair share	665
User share assignments	532
Dynamic user priority	533
Use time decay and committed run time	536
Historical run time decay	536
Configuring historical run time	536
How mbatchd reconfiguration and restart affects historical run time	537
Run time decay	537
Configuring run time decay	538
Committed run time weighting factor	538
Configuring committed run time	538
How fair share affects job dispatch order	540
Host partition user-based fair share	541
Configuring host partition fair share scheduling	541
Queue-level user-based fair share	541
Configuring queue-level fair share	542
Cross-queue user-based fair share	542

Configuring cross-queue fair share	543
Control job dispatch order in cross-queue fair share	544
User-based fair share	530
Configure hierarchical fair share	531
Configuring a share tree	531
Queue-based fair share	544
Slot allocation per queue	546
Configuring slot allocation per queue	546
Typical slot allocation scenarios	548
Users affected by multiple fair share policies	553
Submitting a job and specify a user group	553
Ways to configure fair share	526
Host partition fair share	529
Configuring host partition fair share	529
Chargeback fair share	526
Configuring chargeback fair share	526
Equal share	527
Configuring equal share	527
Priority user and static priority fair share	527
Configuring priority user fair share	528
Configuring static priority fair share	528
GPU runtime fair share	529
Configuring GPU run time	529
Re-sizable jobs and fair share	553
Job count based fair share	694
Job checkpoint and restart	694
About job checkpoint and restart	694
Configuration to enable job checkpoint and restart	696
Job checkpoint and restart behavior	699
Configuration to modify job checkpoint and restart	700
Job checkpoint and restart commands	702
Job migration for checkpoint-able and re-runnable jobs	703
Job migration behavior	705
Configuration to enable job migration	705
Configuration to modify job migration	707
Job migration commands	708
Re-sizable jobs	709
Re-sizable job behavior	709
Configuration to enable re-sizable jobs	710
Re-sizable job commands	711
Re-sizable job management	713
Submitting a re-sizable job	713
Checking pending resize requests	714
Canceling an active pending request	714
Specifying a resize notification command manually	714
Script for resizing	715
How re-sizable jobs work with other LSF features	715
Security in LSF	717
Security considerations	717
Communications between daemons and commands	718
Transmission of IBM Spectrum LSF commands for remote execution	718
Access to jobs belonging to other users	718
Accessing remote hosts	720
False requests	720
Authentication	720

Secure your LSF cluster	721
Secure communications between daemons and commands	721
Encrypt transmission of LSF commands for remote execution and login	722
Restrict user access to remote hosts	723
Secure your cluster against false requests	723
Customize external authentication	724
Enable external authentication of LSF daemons	725
Secure the cluster from root access for batch interactive jobs in pseudoterminals	725
Restrict user access to administration commands and log files	726
Job information access control	726
Setting job information access control	727
Secure the lsf.conf file and prevent users from changing the job user	728
Temporarily enable root privileges	729
View the cluster security settings	729
Advanced configuration	729
Error and event logging	730
System directories and log files	730
About LSF log files	731
Log levels and descriptions	732
Manage error logs	732
Set the log files owner	733
View the number of file descriptors remaining	733
Locate error logs	734
System event log	734
Duplicate logging of event logs	734
Configure duplicate logging	735
Set daemon message log to debug level	736
Set daemon timing levels	738
LSF job termination reason logging	739
View logged job exit information (bacct -l)	739
View recent job exit information (bjobs -l)	740
Termination reasons	740
LSF job exit codes	741
Event generation	742
Event generation	743
Enable event generation for custom programs	743
Events list	743
Arguments passed to the LSF event program	744
Customize batch command messages	744
How LIM determines host models and types	745
Automatically detect operating system types and versions	746
Add a custom host type or model	747
Automatic detection of hardware reconfiguration	748
Set the external static LIM	749
Shared file access	749
Shared files on Windows	750
Use LSF with non-shared file systems	750
Shared configuration file content	750
Authentication and authorization	753
Change authentication method	754
Authentication options	754
Operating system authorization	756
LSF authorization	757
Authorization failure	758
External authentication	759

External authentication with LSF (eauth)	759
Configuration to enable external authentication	762
External authentication behavior	762
Configuration to modify external authentication	763
External authentication commands	764
Kerberos authentication	765
Kerberos authentication with LSF	765
Configuration to enable Kerberos authentication	766
Configuration to modify Kerberos authentication	769
Kerberos authentication commands	770
Handle job exceptions	770
Email job exception details	771
Default eadmin actions	772
Handle job initialization failures	772
Handle host-level job exceptions	774
Handle job exceptions in queues	775
Understand successful application exit values	776
Specify successful application exit values	778
Tune CPU factors	778
View normalized ratings	779
Tune CPU factors	779
Set clean period for DONE jobs	779
Enable host-based resources	780
Portable hardware locality	618
Define GPU resources	782
Define Intel Xeon Phi resources	788
Global fair share scheduling	791
Global fair share background	792
Remote fair share load	793
Sync mode of global fair share policy	794
Global fair share setup and configuration	796
Global policy daemon	797
Global fair share policy	798
Global fair share dynamic user priority	798
Share load synchronization rules	799
Configure queue level user-based global fair share	801
Configure cross-queue user-based global fair share	805
Global fair share scheduling constraints	805
Manage LSF on EGO	806
About LSF on EGO	806
LSF and EGO directory structure	808
Configure LSF and EGO	810
LSF and EGO corresponding parameters	811
Parameters that have changed in LSF 10	812
Special resource groups for LSF management hosts	812
Manage LSF daemons through EGO	813
Bypass EGO login at startup (lsf.sudoers)	814
Set the command-line environment	814
LSF features with EGO-enabled SLA scheduling (Obsolete)	815
Supported LSF features with EGO-enabled SLA scheduling (Obsolete)	815
LSF features that require modification to work with EGO-enabled SLA scheduling (Obsolete)	816
Unsupported LSF features with EGO-enabled SLA scheduling (Obsolete)	817
Logging and troubleshooting	817
EGO log files	818
Troubleshooting using multiple EGO log files	820

Frequently asked questions	822
Load sharing X applications	823
Start an xterm	823
xterm on a PC	823
Set up Exceed to log on the least loaded host	824
Start an xterm in Exceed	824
Examples	824
Using LSF with the Etnus TotalView Debugger	825
How IBM Spectrum LSF Works with TotalView	825
Running jobs for TotalView debugging	827
Controlling and monitoring jobs being debugged in TotalView	828
Register LSF host names and IP addresses to LSF servers	828
Performance tuning	829
Tune your cluster	829
Tune LIM	830
Load thresholds	830
Compare LIM load thresholds	831
LIM reports a host as busy	832
Interactive jobs	832
Multiprocessor systems	832
How LSF works with LSF_MASTER_LIST	832
Using a DNS host cache to improve cluster startup performance	833
Improve mbatchd response time after mbatchd restart	834
Improve mbatchd query performance	834
Configuring mbatchd to use multithreading	834
Multithread batch queries	836
Setting a dedicated query port for mbatchd	836
Specify an expiry time for child mbatchd	836
Configure mbatchd to push new job information to child mbatchd	837
Specify hard CPU affinity	837
Offloading the mbatchd daemon using the LSF rate limiter (lsfproxycd daemon)	838
Enabling and configuring the LSF rate limiter	839
Diagnostics for the LSF rate limiter and lsfproxycd daemon	841
Logging mbatchd performance metrics	841
Logging mbatchd and mbschd profiling information	843
Improve performance of mbatchd for job array switching events	843
Increase queue responsiveness	844
Automatically bind LSF daemons to specific CPU cores	844
Use LSF Explorer to improve the performance of the bacct and bhist commands, or to retrieve additional data	845
Improve slot utilization by preventing bwait from running in jobs	846
Achieve performance and scalability	846
Optimize performance in large sites	846
Tune UNIX for large clusters	846
Increase the file descriptor limit	847
Tune LSF for large clusters	847
Manage scheduling performance	848
Enable fast job dispatch	849
Enable continuous scheduling	849
Use scheduler threads to evaluate resource requirement matching	849
Limit job dependency evaluation	850
Limit the number of batch queries	850
Improve the speed of host status updates	851
Limit your user's ability to move jobs in a queue	851
Manage the number of pending reasons	852
Achieve efficient event switching	852

Automatic load updates	853
Manage I/O performance of the info directory	853
Job ID limit	854
Energy aware scheduling	854
Managing host power states	855
Configuring host power state management	855
Power parameters in lsb.params	856
PowerPolicy section in lsb.resources	857
Controlling and monitoring host power state management	858
Valid host statuses for power saved mode	861
Disabling the power operation feature	862
Changing lsf.shared / lsf.cluster	862
Integration with Advance Reservation	862
Integration with provisioning systems	862
CPU frequency management	863
Configuring CPU frequency management	864
Specifying CPU frequency management for jobs	864
Job energy usage reporting	865
Resource usage in job summary email	865
Automatic CPU frequency selection	865
Prerequisites	866
Configure MySQL database	866
Configuring automatic CPU frequency selection	867
Installing and configuring benchmarking programs	867
Checking compute node performance	869
Calculating coefficient data	869
Creating an energy policy tag	871
Energy policy tag format	872
Generate an energy policy tag	872
Enable automatic CPU frequency selection	873
LSF multicluster capability	873
Overview of the LSF multicluster capability	874
Benefits of the LSF multicluster capability	874
Two multicluster models	874
Set up LSF multicluster capability	875
Setup overview	875
Multicluster system requirements	876
Installation and configuration procedures for LSF multicluster	877
Install LSF multicluster	878
Set common ports	878
Non-uniform name spaces	879
User-level account mapping	880
Restricted awareness of remote clusters	881
Add or modify RemoteClusters list	882
Security of daemon communication	883
Authentication between clusters	883
Resource usage updates for MultiCluster jobs	885
Global limits for job resource allocations	885
Configure global limits	885
View the global limits	886
Global job IDs for forwarding and forwarded clusters using LSF multicluster capability	886
Multicluster information cache	887
Job forwarding model	888
Job forwarding model overview	888
Job scheduling under the job forwarding model	891

Queue scheduling parameters under job forwarding model	892
Advance reservations across clusters	893
Special considerations under job forwarding model	894
Job migration	896
Checkpoint a multicluster job	897
Absolute priority scheduling	898
Strict resource requirement select string syntax	899
Compute unit requirement strings	899
Multicluster queues	899
Enable multicluster queues	899
Remote-only queues	901
Configure a remote-only queue	901
Request a specific cluster	905
Remote cluster equivalency	906
Remote Resources	906
Remote queue workload job-forwarding scheduler	907
Enable queue preference	913
Configure queue preference	913
Enable job slot limit	914
Configure pending job slot limit	914
Pre-exec retry threshold	915
Retry threshold and suspend notification	915
Pending MultiCluster job limit	915
Update pending reason for MultiCluster jobs	916
Configure the pending reason updating interval	916
Configure the pending reason update package size	917
Remote timeout limit	917
Enable job priority in MultiCluster job forward mode	917
Specify a job priority (bsub -sp)	918
Configure maximum job priority	919
Enhance fair share calculation to include the job forwarding mode	919
Resource leasing model	920
Lease model overview	920
Using the lease model	921
Special considerations under resource leasing model	923
Resource export	923
Create an export policy	924
Export workstations	925
Distribution policy for automatically selected hosts	926
Export special hosts	926
Distribution policy for named hosts	927
Export other resources	928
Export shared resources	929
Shared lease	929
Enable shared leasing	930
Borrow resources	931
Parallel jobs and the lease model	932
Submitting jobs using JSDL	932
Using JSDL files with LSF	932
Submit a job using a JSDL file	941
Collect resource values using elim.jsdl	942
Enabling JSDL resource collection	942
Submitting jobs using JSDL	932
Using JSDL files with LSF	932

Submit a job using a JSDL file	941
Collect resource values using elim.jsdl	942
Enabling JSDL resource collection	942
LSF Session Scheduler	952
About LSF Session Scheduler	953
Installing LSF Session Scheduler	955
How LSF Session Scheduler runs tasks	955
Running and monitoring LSF Session Scheduler jobs	958
Troubleshooting	962
LSF on Cray	966
Downloading and installing	967
Configuring the integration	968
Integration directory and file structure	971
Submitting and running jobs	973
Assumptions and limitations	974

LSF cluster overview

Get an overview of your cluster and the location of important LSF directories and configuration files.

- [LSF terms and concepts](#)
Learn LSF basic terms and concepts that will help when using LSF.
- [Cluster characteristics](#)
Find the name of your cluster after installation, cluster administrators, and where hosts are defined.
- [File systems, directories, and files](#)
LSF is designed for networks where all hosts have shared file systems, and files have the same names on all hosts.
- [Important directories and configuration files](#)
LSF configuration is administered through several configuration files, which you use to modify the behavior of your cluster.

LSF terms and concepts

Learn LSF basic terms and concepts that will help when using LSF.

Job states

IBM® Spectrum LSF jobs have several states.

PEND

Waiting in a queue for scheduling and dispatch.

RUN

Dispatched to a host and running.

DONE

Finished normally with zero exit value.

EXIT

Finished with nonzero exit value.

PSUSP

Suspended while the job is pending.

USUSP

Suspended by user.

SSUSP

Suspended by the LSF system.

POST_DONE

Post-processing completed without errors.

POST_ERR

Post-processing completed with errors.

UNKWN

The **mbatchd** daemon lost contact with the **sbatchd** daemon on the host where the job runs.

WAIT

For jobs submitted to a chunk job queue, members of a chunk job that are waiting to run.

ZOMBI

A job is in **ZOMBI** state if the job is killed when its state is **UNKWN** since the execution host is unreachable, or if a re-runnable job is requeued since the execution host is unavailable.

A job can be in **ZOMBI** state, if:

- A job in **UNKWN** status gets a kill signal, LSF will change the job to **ZOMBI** state. For instance, running the **bkill** command to kill a job in **UNKWN** state, a re-runnable job is requeued when the execution host is unavailable, or the **mbatchd** daemon killing the **UNKWN** job when the [REMOVE_HUNG_JOBS_FOR](#) parameter is set in the `lsb.params` file. (See [Removing hung jobs from LSF](#) for details about `REMOVE_HUNG_JOBS_FOR`.)
- You force remove a job from LSF (that is, run the [bkill -r command](#) on a running job), the **mbatchd** daemon will first change the job status to **ZOMBI** state and also send a signal to the **sbatchd** daemon for the job's execution

host. After **sbatchd** successfully kills the job processes, it sends a reply back to **mbatchd**; **mbatchd** can then change the job's status from **ZOMBI** to **EXIT**. In typical cases, this procedure would not take long.

Host

An LSF host is an individual computer in the cluster.

Each host might have more than one processor. Multiprocessor hosts are used to run parallel jobs. A multiprocessor host with a single process queue is considered a single machine. A box full of processors that each have their own process queue is treated as a group of separate machines.

Tip:

The names of your hosts should be unique. They cannot be the same as the cluster name or any queue that is defined for the cluster.

Job

An LSF job is a unit of work that runs in the LSF system.

A job is a command that is submitted to LSF for execution, by using the **bsub** command. LSF schedules, controls, and tracks the job according to configured policies.

Jobs can be complex problems, simulation scenarios, extensive calculations, anything that needs compute power.

Job files

When a job is submitted to a queue, LSF holds it in a job file until conditions are right for it run. Then, the job file is used to run the job.

On UNIX, the job file is a Bourne shell script that is run at execution time.

On Windows, the job file is a batch file that is processed at execution time.

Interactive batch job

An interactive batch job is a batch job that allows you to interact with the application and still take advantage of LSF scheduling policies and fault tolerance.

All input and output are through the terminal that you used to type the job submission command.

When you submit an interactive job, a message is displayed while the job is awaiting scheduling. A new job cannot be submitted until the interactive job is completed or terminated.

Interactive task

An interactive task is a command that is not submitted to a batch queue and scheduled by LSF, but is dispatched immediately.

LSF locates the resources that are needed by the task and chooses the best host among the candidate hosts that has the required resources and is lightly loaded. Each command can be a single process, or it can be a group of cooperating processes.

Tasks are run without using the batch processing features of LSF but still with the advantage of resource requirements and selection of the best host to run the task based on load.

Local task

A local task is an application or command that does not make sense to run remotely.

For example, the **ls** command on UNIX.

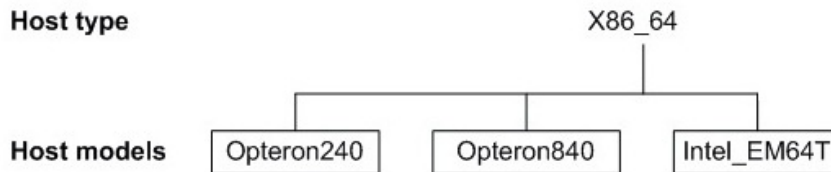
Remote task

A remote task is an application or command that can be run on another machine in the cluster.

Host types and host models

Hosts in LSF are characterized by host type and host model.

The following example is a host with type X86_64, with host models Opteron240, Opteron840, Intel_EM64T, and so on.



Host type

An LSF host type is the combination of operating system and host CPU architecture.

All computers that run the same operating system on the same computer architecture are of the same type. These hosts are binary-compatible with each other.

Each host type usually requires a different set of LSF binary files.

Host model

An LSF host model is the host type of the computer, which determines the CPU speed scaling factor that is applied in load and placement calculations.

The CPU factor is considered when jobs are being dispatched.

Resources

LSF resources are objects in the LSF system resources that LSF uses track job requirements and schedule jobs according to their availability on individual hosts.

Resource usage

The LSF system uses built-in and configured resources to track resource availability and usage. Jobs are scheduled according to the resources available on individual hosts.

Jobs that are submitted through the LSF system will have the resources that they use monitored while they are running. This information is used to enforce resource limits and load thresholds as well as fair share scheduling.

LSF collects the following kinds of information:

- Total CPU time that is consumed by all processes in the job
- Total resident memory usage in KB of all currently running processes in a job
- Total virtual memory usage in KB of all currently running processes in a job
- Currently active process group ID in a job
- Currently active processes in a job

On UNIX and Linux, job-level resource usage is collected through PIM.

Load indices

Load indices measure the availability of dynamic, non-shared resources on hosts in the cluster. Load indices that are built into the LIM are updated at fixed time intervals.

External load indices

Defined and configured by the LSF administrator and collected by an External Load Information Manager (**ELIM**) program. The **ELIM** also updates LIM when new values are received.

Static resources

Built-in resources that represent host information that does not change over time, such as the maximum RAM available to user processes or the number of processors in a machine. Most static resources are determined by the **LIM** at startup time.

Static resources can be used to select appropriate hosts for particular jobs that are based on binary architecture, relative CPU speed, and system configuration.

Load thresholds

Two types of load thresholds can be configured by your LSF administrator to schedule jobs in queues. Each load threshold specifies a load index value:

- The **loadSched** load threshold determines the load condition for dispatching pending jobs. If a host's load is beyond any defined **loadSched**, a job cannot be started on the host. This threshold is also used as the condition for resuming suspended jobs.
- The **loadStop** load threshold determines when running jobs can be suspended.

To schedule a job on a host, the load levels on that host must satisfy both the thresholds that are configured for that host and the thresholds for the queue from which the job is being dispatched.

The value of a load index might either increase or decrease with load, depending on the meaning of the specific load index. Therefore, when you compare the host load conditions with the threshold values, you need to use either greater than (>) or less than (<), depending on the load index.

Runtime resource usage limits

Limit the use of resources while a job is running. Jobs that consume more than the specified amount of a resource are signaled.

Hard and soft limits

Resource limits that are specified at the queue level are hard limits while limits that are specified with job submission are soft limits.

Resource allocation limits

Restrict the amount of a resource that must be available during job scheduling for different classes of jobs to start, and which resource consumers the limits apply to. If all of the resource is consumed, no more jobs can be started until some of the resource is released.

Resource requirements (bsub -R command option)

The **bsub -R** option specifies resources requirements for the job. Resource requirements restrict which hosts the job can run on. Hosts that match the resource requirements are the candidate hosts. When LSF schedules a job, it collects the load index values of all the candidate hosts and compares them to the scheduling conditions. Jobs are only dispatched to a host if all load values are within the scheduling thresholds.

Cluster characteristics

Find the name of your cluster after installation, cluster administrators, and where hosts are defined.

Cluster name and administrators

Your cluster is installed according to the installation options specified by the **lsfinstall -f install.config** command and the options you chose in the `install.config` file. The cluster name that you specified at installation is part of the name of the `LSF_CONFDIR/lsf.cluster.cluster_name` file.

`/usr/share/lsf/lsf_10/conf/lsf.cluster.lsf_10`

Cluster administrators are listed in the `ClusterAdmins` section of the `LSF_CONFDIR/lsf.cluster.cluster_name` file.

LSF hosts

- Host types that are installed in your cluster are listed in the `Hosts` section of the `LSF_CONFDIR/lsf.cluster.cluster_name` file.
- The LSF management host is the first host that is configured in the `Hosts` section of `LSF_CONFDIR/lsf.cluster.cluster_name` file.
- LSF *server hosts* that are defined in your cluster are indicated by `1` in the server column of the `Hosts` section in the `LSF_CONFDIR/lsf.cluster.cluster_name` file.
- LSF *client-only hosts* that are defined in your cluster are indicated by `0` in the server column of the `Hosts` section in the `LSF_CONFDIR/lsf.cluster.cluster_name` file.

File systems, directories, and files

LSF is designed for networks where all hosts have shared file systems, and files have the same names on all hosts.

LSF includes support for copying user data to the execution host before a batch job runs, and for copying results back after the job runs.

In networks where the file systems are not shared, this support can be used to give remote jobs access to local data.

Supported file systems

UNIX

On UNIX systems, LSF supports the following shared file systems:

Network File System (NFS)

NFS file systems can be mounted permanently or on demand by using the **automount** command.

Andrew File System (AFS)

Supported on an on-demand basis under the parameters of the 9.1.2 integration with some published configuration parameters. Supports sequential and parallel user jobs that access AFS, `JOB_SPOOL_DIR` on AFS, and job output and error files on AFS.

Distributed File System (DCE/DFS)

Supported on an on-demand basis.

Windows

On Windows, directories that contain LSF files can be shared among hosts from a Windows server machine.

Non-shared directories and files

LSF is used in networks with shared file space. When shared file space is not available, LSF can copy needed files to the execution host before the job runs, and copy result files back to the submission host after the job completes.

Some networks do not share files between hosts. LSF can still be used on these networks, with reduced fault tolerance.

- [Example directory structures](#)

The following figures show typical directory structures for a new installation on UNIX and Linux or on Microsoft Windows. Depending on which products you installed and platforms you selected, your directory structure might be different.

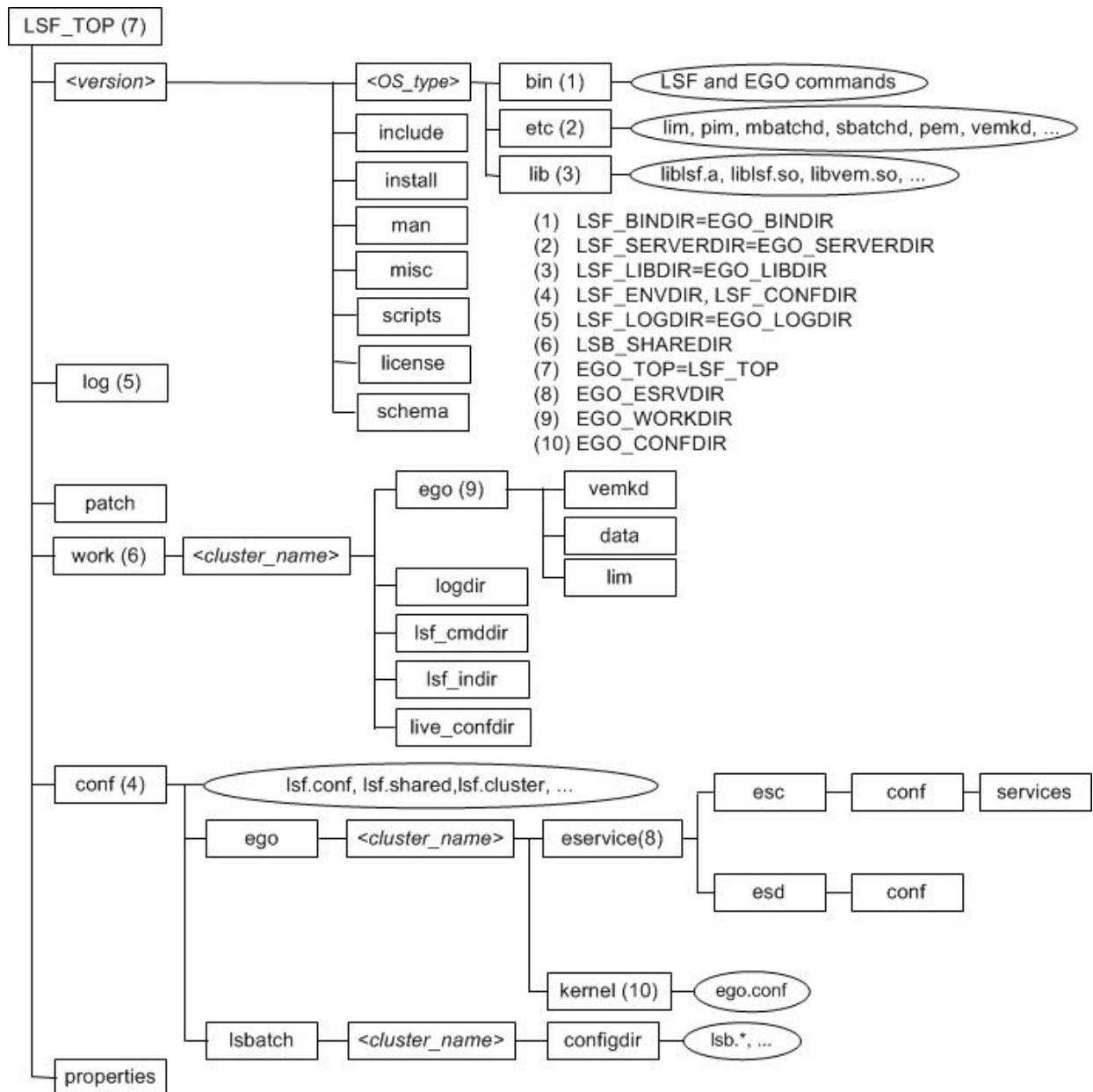
Example directory structures

The following figures show typical directory structures for a new installation on UNIX and Linux or on Microsoft Windows. Depending on which products you installed and platforms you selected, your directory structure might be different.

- [UNIX and Linux](#)
The following figure shows a typical directory structure for a new UNIX or Linux installation with the **lsfinstall** command.
- [Microsoft Windows directory structure](#)
The following figure shows a typical directory structure for a new Windows installation.

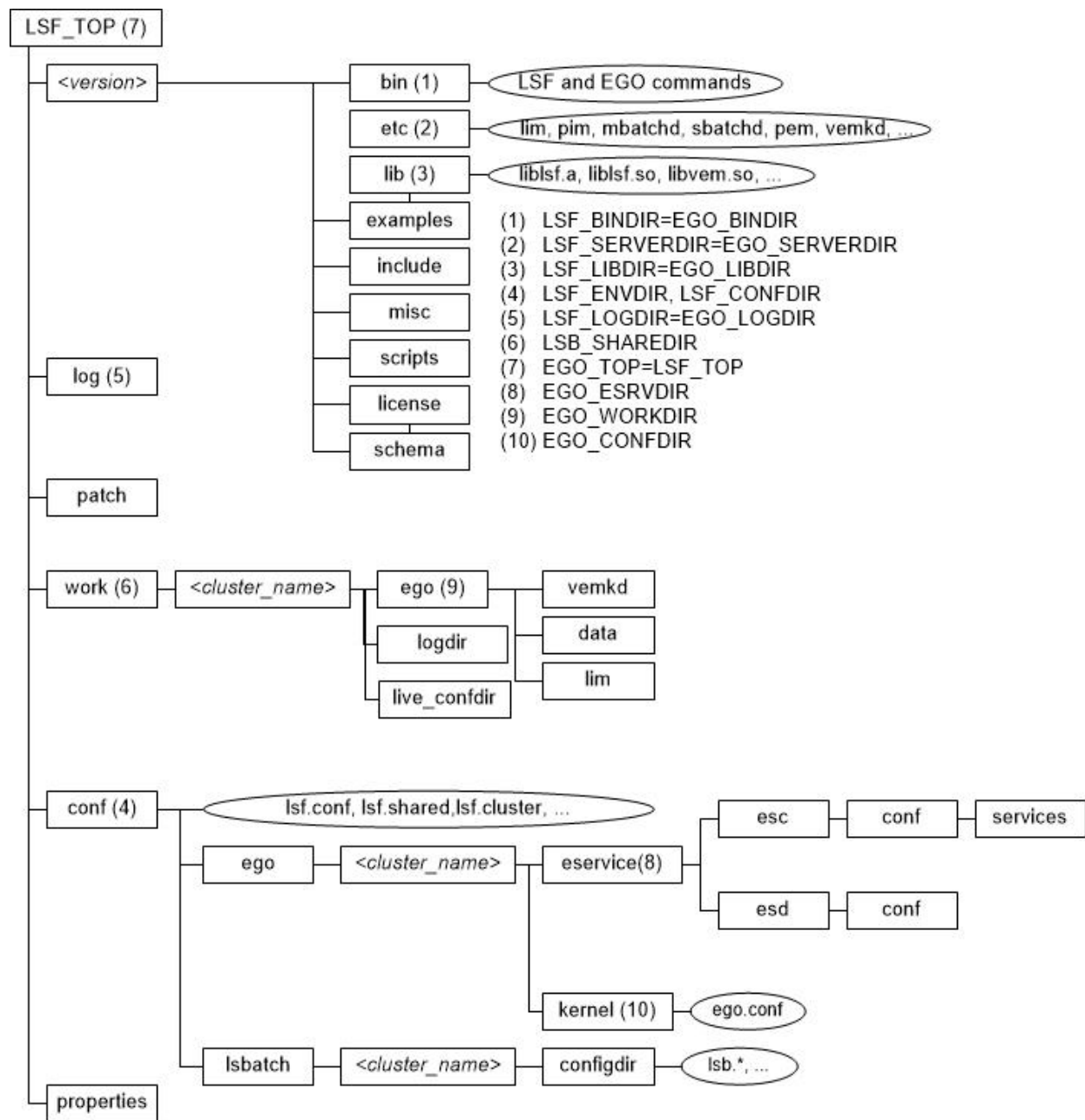
UNIX and Linux

The following figure shows a typical directory structure for a new UNIX or Linux installation with the **lsfinstall** command.



Microsoft Windows directory structure

The following figure shows a typical directory structure for a new Windows installation.



Important directories and configuration files

LSF configuration is administered through several configuration files, which you use to modify the behavior of your cluster.

Four important LSF configuration files

The following are the four most important files you work with most often:

- LSF_CONFDIR/lsf.conf
- LSF_CONFDIR/lsf.cluster.cluster_name
- LSF_CONFDIR/lsf.shared
- LSB_CONFDIR/cluster_name/configdir/lsb.queues

These files are created during product installation according to the options you specified in the install.config file. After installation, you can change the configuration parameters in these files to suit the needs of your site.

Who owns these files

Except for LSF_CONFDIR/lsf.conf, which is owned by root, all of these files are owned by the primary LSF administrator, and readable by all cluster users.

lsf.conf

The most important file in LSF. It contains the paths to the configuration directories, log directories, libraries, and other global configuration information. The location of the lsf.conf file is defined by the LSF_ENVDIR variable. If LSF cannot find this file, it cannot start properly.

By default, LSF checks the directory that is defined by the LSF_ENVDIR parameter for the location of the lsf.conf file. If the lsf.conf file is not in LSF_ENVDIR, LSF looks for it in the /etc directory.

lsf.cluster.cluster_name

Defines the host names, models, and types of all of the hosts in the cluster. It also defines the user names of the LSF administrators, and the locations of different shared resources for one cluster.

lsf.shared

This file is like a dictionary that defines all the keywords that are used by the cluster. You can add your own keywords to specify the names of resources or host types.

lsb.queues

Defines the workload queues and their parameters for one cluster.

LSF directories

The following directories are owned by the primary LSF administrator and are readable by all cluster users:

Directory	Description	Example
LSF_CONFDIR	LSF configuration directory	/usr/share/lsf/cluster1/conf/
LSB_CONFDIR	Batch system configuration directory	/usr/share/lsf/cluster1/conf/lsbatch/
LSB_SHAREDIR	Job history directory	/usr/share/lsf/cluster1/work/
LSF_LOGDIR	Server daemon error logs, one for each daemon	/usr/share/lsf/cluster1/log/

The following directories are owned by root and are readable by all cluster users:

Directory	Description	Example
LSF_BINDIR	LSF user commands, which are shared by all hosts of the same type	/usr/share/lsf/cluster1/10.1.0/sparc-sol10/bin/
LSF_INCLUDEDIR	Header files lsf/lsf.h and lsf/lsbatch.h	/usr/share/lsf/cluster1/10.1.0/include/
LSF_LIBDIR	LSF libraries, which are shared by all hosts of the same type	/usr/share/lsf/cluster1/10.1.0/sparc-sol10/lib/
LSF_MANDIR	LSF man pages	/usr/share/lsf/cluster1/10.1.0/man/
LSF_MISC	Examples and other miscellaneous files	/usr/share/lsf/cluster1/10.1.0/misc/
LSF_SERVERDIR	Server daemon binary files, scripts, and other utilities, which are shared by all hosts of the same type	/usr/share/lsf/cluster1/10.1.0/sparc-sol10/etc/
LSF_TOP	Top-level installation directory	/usr/share/lsf/cluster1/

Other configuration directories can be specified in the LSF_CONFDIR/lsf.conf file.

LSF cluster configuration files

The following files are owned by the primary LSF administrator and are readable by all cluster users:

File	Example
------	---------

File	Example
Global configuration files, which describe the configuration and operation of the cluster	<ul style="list-style-type: none"> • /usr/share/lsf/cluster1/conf/ego/cluster1/kernel/ego.conf • /usr/share/lsf/cluster1/conf/lsf.conf
Keyword definition file that is shared by all clusters. Defines cluster name, host types, host models, and site-specific resources	/usr/share/lsf/cluster1/conf/lsf.shared
Cluster configuration file that defines hosts, administrators, and location of site-defined shared resources	/usr/share/lsf/cluster1/conf/lsf.cluster.cluster1

LSF batch workload system configuration files

The following files are owned by the primary LSF administrator and are readable by all cluster users:

File	Example
Server hosts and their attributes, such as scheduling load thresholds, dispatch windows, and job slot limits. If no hosts are defined in this file, then all LSF server hosts listed in LSF_CONFDIR/lsf.cluster. <i>cluster_name</i> are assumed to be LSF batch server hosts.	/usr/share/lsf/cluster1/conf/lsbatch/cluster1/configdir/lsb.hosts
LSF scheduler and resource broker plug-in modules. If no scheduler or resource broker modules are configured, LSF uses the default scheduler plug-in module named schmod_default.	/usr/share/lsf/cluster1/conf/lsbatch/cluster1/configdir/lsb.modules
LSF batch system parameter file	/usr/share/lsf/cluster1/conf/lsbatch/cluster1/configdir/lsb.params
Job queue definitions	/usr/share/lsf/cluster1/conf/lsbatch/cluster1/configdir/lsb.queues
Resource allocation limits, exports, and resource usage limits.	/usr/share/lsf/cluster1/conf/lsbatch/cluster1/configdir/lsb.resources
LSF user groups, hierarchical fair share for users and user groups, and job slot limits for users and user groups. Also used to configure account mappings for the LSF multicluster capability.	/usr/share/lsf/cluster1/conf/lsbatch/cluster1/configdir/lsb.users
Application profiles, which contain common parameters for the same type of jobs, including the execution requirements of the applications, the resources they require, and how they are run and managed. This file is optional. Use the DEFAULT_APPLICATION parameter in the lsb.params file to specify a default application profile for all jobs. LSF does not automatically assign a default application profile.	/usr/share/lsf/cluster1/conf/lsbatch/cluster1/configdir/lsb.applications

LSF batch log files

File	Example
Batch events log	/usr/share/lsf/cluster1/work/cluster1/logdir/lsb.events
Batch accounting log	/usr/share/lsf/cluster1/work/cluster1/logdir/lsb.acct

Daemon log files

LSF server daemon log files are stored in the directory that is specified by LSF_LOGDIR in LSF_CONFDIR/lsf.conf.

File	Example
Load information manager (lim)	/usr/share/lsf/cluster1/log/lim.log.hosta
Remote execution server (res)	/usr/share/lsf/cluster1/log/res.log.hosta
Management batch daemon (mbatchd)	/usr/share/lsf/cluster1/log/mbatchd.log.hosta
Management scheduler daemon (mbschd)	/usr/share/lsf/cluster1/log/mbschd.log.hosta
Server batch daemon (sbatchd)	/usr/share/lsf/cluster1/log/sbatchd.log.hosta
process information manager (pim)	/usr/share/lsf/cluster1/log/pim.log.hosta

Who owns and who should write to LSF_LOGDIR

Note: Make sure that the primary LSF administrator owns the LSF log directory (LSF_LOGDIR parameter), and that root can write to this directory. If an LSF server cannot write to LSF_LOGDIR parameter, the error logs are created in /tmp.

Where to go next

Use your new IBM® Spectrum LSF cluster, described in [Work with LSF](#).

Work with LSF

Start and stop LSF daemons, and reconfigure cluster properties. Check LSF status and submit LSF jobs.

- [Start, stop, and reconfigure LSF](#)
Use LSF administration commands **lsadmin** and **badmin** to start and stop LSF daemons, and reconfigure cluster properties.
- [Check LSF status](#)
Use LSF administration commands to check cluster configuration, see cluster status, and LSF batch workload system configuration and status.
- [Run LSF jobs](#)
Use the **bsub** and **lrun** commands to run jobs through LSF. Use the **bjobs** command to see the status of your jobs. Control job execution with the **bstop**, **bresume**, and **bkill** commands.
- [Manage users, hosts, and queues](#)
Make your cluster available to users with `cschrc.lsf` and `profile.lsf`. Add or remove hosts and queues from your cluster.
- [Configure LSF startup](#)
Use the `lsf.sudoers` file so that LSF administrators can start and stop LSF daemons. Set up LSF to start automatically.
- [Manage software licenses and other shared resources](#)
Set up an LSF external LIM (**ELIM**) to monitor software licenses as dynamic shared resources.

Start, stop, and reconfigure LSF

Use LSF administration commands **lsadmin** and **badmin** to start and stop LSF daemons, and reconfigure cluster properties.

Two LSF administration commands (lsadmin and badmin)

Important: Only LSF administrators or root can run these commands.

To start and stop LSF, and to reconfigure LSF after you change any configuration file, use the following commands:

- The **lsadmin** command controls the operation of the **lim** and **res** daemons.
- The **badmin** command controls the operation of the **mbatchd** and **sbatchd** daemons.

If you installed LSF as a non-root user

By default, only root can start LSF daemons. If the **lsfinstall** command detected that you installed as non-root user, you chose to configure either a multi-user cluster or a single-user cluster:

Multi-user configuration

Only root can start LSF daemons. Any user can submit jobs to your cluster.

For information about changing ownership and permissions for the **lsadmin** and **badadmin** commands, see [Troubleshooting LSF problems](#).

To permit LSF administrators to start and stop LSF daemons, set up the `/etc/lsf.sudoers` file, as described in [Configure LSF Startup](#).

Single-user

Your user account must be primary LSF administrator. You are able to start LSF daemons, but only your user account can submit jobs to the cluster. Your user account must be able to read the system kernel information, such as `/dev/kmem`.

- [Setting up the LSF environment with `cschrc.lsf` and `profile.lsf`](#)
Before you use LSF, you must set up the LSF execution environment with the `cschrc.lsf` or `profile.lsf` file.
- [Starting your cluster](#)
Use the **lsadmin** and **badadmin** commands to start the LSF daemons.
- [Stopping your cluster](#)
Use the **lsadmin** and **badadmin** commands to stop the LSF daemons.
- [Reconfiguring your cluster with `lsadmin` and `badadmin`](#)
Use the **lsadmin** and **badadmin** commands to reconfigure LSF after you change any configuration file.
- [LSF Cluster Management and Operations](#)

Setting up the LSF environment with `cschrc.lsf` and `profile.lsf`

Before you use LSF, you must set up the LSF execution environment with the `cschrc.lsf` or `profile.lsf` file.

Procedure

After you log in to an LSF host, use one of the following shell environment files to set your LSF environment.

- In the **csch** or **tcsh** shell, run the **source** command:

```
% source <LSF_TOP>/conf/cschrc.lsf
```
- In the **sh**, **ksh**, or **bash** shell run the following command:

```
$ . <LSF_TOP>/conf/profile.lsf
```

The files `cschrc.lsf` and `profile.lsf` are created during installation by the **lsfinstall** command to set up the LSF operating environment.

Starting your cluster

Use the **lsadmin** and **badadmin** commands to start the LSF daemons.

Procedure

1. Log in as root to each LSF server host.
If you installed a single-user cluster as a non-root user, log in as primary LSF administrator.

Start with the LSF management host, and repeat these steps on all LSF hosts.
2. Use the following commands to start the LSF cluster:

```
# bctrld start lim
# bctrld start res
# bctrld start sbd
```

Before you use any LSF commands, wait a few minutes for the **lim** daemon all hosts to do the following operations:

- Contact each other
- Select the management host
- Exchange initialization information

Stopping your cluster

Use the **lsadmin** and **badmin** commands to stop the LSF daemons.

Procedure

1. Log in as root to each LSF server host.
If you installed a single-user cluster as a non-root user, log in as primary LSF administrator.
2. Use the following commands to stop the LSF cluster:

```
# bctrld stop sbd all
# bctrld stop res all
# bctrld stop lim all
```

Reconfiguring your cluster with lsadmin and badmin

Use the **lsadmin** and **badmin** commands to reconfigure LSF after you change any configuration file.

Procedure

1. Log in as root to each LSF server host.
If you installed a single-user cluster as a non-root user, log in as primary LSF administrator.
2. Use the following commands to reconfigure the LSF cluster:

- Reload modified LSF configuration files and restart **lim**:

```
# lsadmin reconfig
```

- Reload modified LSF batch configuration files:

```
# badmin reconfig
```

- Reload modified LSF batch configuration files and restart **mbatchd**:

```
# badmin mbdrestart
```

This command also reads the LSF_LOGDIR/lsb.events file, so it can take some time to complete if a lot of jobs are running.

Check LSF status

Use LSF administration commands to check cluster configuration, see cluster status, and LSF batch workload system configuration and status.

Example command output

The LSF commands that are shown in this section show examples of typical output. The output that you see might differ according to your configuration.

The commands are described briefly so that you can easily use them to verify your LSF installation. See the [LSF Command Reference](#) or the LSF man pages for complete usage and command options. You can use these commands on any LSF host.

If you get proper output from these commands, your cluster is ready to use. If your output has errors, see [Troubleshooting LSF problems](#) for help.

- [Check cluster configuration with the lsadmin command](#)
The **lsadmin** command controls the operation of an LSF cluster and administers the LSF daemons **lim** and **res**.
- [Check cluster status with the lsid and lsload commands](#)
The **lsid** command tells you if your LSF environment is set up properly. The **lsload** command displays the current load levels of the cluster.
- [Check LSF batch system configuration with badmin](#)
The **badmin** command controls and monitors the operation of the LSF batch workload system.
- [Find batch system status using the bhosts and bqueues commands](#)
Use the **bhosts** command to see whether the LSF batch workload system is running properly. The **bqueues** command displays the status of available queues and their configuration parameters.

Related concepts

- [LSF Cluster Management and Operations](#)

Check cluster configuration with the lsadmin command

The **lsadmin** command controls the operation of an LSF cluster and administers the LSF daemons **lim** and **res**.

Use the **lsadmin ckconfig** command to check the LSF configuration files. The **-v** option displays detailed information about the LSF configuration:

The messages that are shown in the following output are typical of **lsadmin ckconfig -v**. Other messages might indicate problems with your LSF configuration.

```
% lsadmin ckconfig -v
Checking configuration files ...

EGO 3.6.0 build 800000, Jul 25 2017
Copyright International Business Machines Corp. 1992, 2016.
US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP
Schedule Contract with IBM Corp.

binary type: linux2.6-glibc2.3-x86_64
Reading configuration from /opt/lsf/conf/lsf.conf
Aug 3 13:45:27 2017 20884 6 3.6.0 Lim starting...
Aug 3 13:45:27 2017 20884 6 3.6.0 LIM is running in advanced workload execution mode.
Aug 3 13:45:27 2017 20884 6 3.6.0 Master LIM is not running in EGO_DISABLE_UNRESOLVABLE_HOST
mode.
Aug 3 13:45:27 2017 20884 5 3.6.0 /opt/lsf/10.1/linux2.6-glibc2.3-x86_64/etc/lim -C
Aug 3 13:45:27 2017 20884 7 3.6.0 Could not construct product entitlement version array
Aug 3 13:45:27 2017 20884 Last message repeated 1 time(s).
Aug 3 13:45:27 2017 20884 6 3.6.0 initEntitlement: EGO_AUDIT_MAX_SIZE was not set. Default
value <100> will be used.
Aug 3 13:45:27 2017 20884 6 3.6.0 initEntitlement: EGO_AUDIT_MAX_ROTATE was not set. Default
value <20> will be used.
Aug 3 13:45:27 2017 20884 6 3.6.0 LIM is running as IBM Spectrum LSF Standard Edition.
Aug 3 13:45:27 2017 20884 6 3.6.0 reCheckClass: numhosts 1 so reset exchIntvl to 15.00
Aug 3 13:45:27 2017 20884 6 3.6.0 Checking Done.
-----
No errors found.
```

Check cluster status with the `lsid` and `lsload` commands

The **lsid** command tells you if your LSF environment is set up properly. The **lsload** command displays the current load levels of the cluster.

lsid command

The **lsid** command displays the current LSF version number, cluster name, and host name of the current LSF management host for your cluster.

The LSF management host name that is displayed by the **lsid** command can vary, but it is usually the first host that is configured in the **Hosts** section of the `LSF_CONFDIR/lsf.cluster.cluster_name` file.

```
% lsid
IBM Spectrum
LSF Standard 10.1.0.0, Apr 04 2016
Copyright International Business Machines Corp, 1992-2016.
US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP
Schedule Contract with IBM Corp.
```

```
My cluster name is cluster1
My master name is hosta
```

If you see the message

```
Cannot open lsf.conf file
```

The `LSF_ENVDIR` environment variable is probably not set correctly. Use the `cschrc.lsf` or `profile.lsf` file to set up your environment. See [Troubleshooting LSF problems](#) for more help

lsload command

The output of the **lsload** command contains one line for each host in the cluster. Normal status is **ok** for all hosts in your cluster.

```
% lsload
HOST_NAME  status  r15s  r1m  r15m  ut   pg   ls   it  tmp  swp  mem
hosta      ok      0.0   0.0  0.1   1%   0.0   1   224 43G  67G  3G
hostc      -ok     0.0   0.0  0.0   3%   0.0   3   0   38G 40G  7G
hostf      busy    *6.2   6.9  9.5   85%  1.1  30   0   5G  400G 385G
hosth      busy    0.1   0.1  0.3   7%   *17   6   0   9G  23G  28G
hostv      unavail
```

A **busy** status is shown for hosts with any load index beyond their configured thresholds. An asterisk (*) marks load indexes that are beyond their thresholds, causing the host status to be **busy**. A minus sign (-) in front of the value **ok** means that **res** is not running on that host.

If you see one of the following messages after you start or reconfigure LSF, wait a few seconds and try the **lsload** command again to give the **lim** daemon on all hosts time to initialize.

```
lsid: getentitlementinfo() failed: LIM is down; try later
```

or

```
LSF daemon (LIM) not responding ... still trying
```

If the problem persists, see [Troubleshooting LSF problems](#) for help.

Other useful commands

- The **bparams** command displays information about the LSF batch system configuration parameters.
- The **bhist** command displays historical information about jobs.

Check LSF batch system configuration with badmin

The **badmin** command controls and monitors the operation of the LSF batch workload system.

Use the **badmin ckconfig** command to check the LSF batch system configuration files. The **-v** option displays detailed information about the configuration:

The messages in the following output are typical of **badmin ckconfig -v**. Other messages might indicate problems with your LSF batch workload system configuration.

```
% badmin ckconfig -v
Checking configuration files ...
Dec 20 12:22:55 2015 20246 9 9.1.3 minit: Trying to call LIM to get cluster name
...
Dec 20 12:22:55 2015 20246 9 9.1.3 Batch is enabled
Dec 20 12:22:55 2015 4433 9 9.1.3 Checking Done
-----
No errors found.
```

See [Troubleshooting LSF problems](#) or the [LSF Command Reference](#) for help with some common configuration errors.

Find batch system status using the bhosts and bqueues commands

Use the **bhosts** command to see whether the LSF batch workload system is running properly. The **bqueues** command displays the status of available queues and their configuration parameters.

To use LSF batch commands, the cluster must be up and running. See [Starting your cluster](#) for information about starting LSF daemons.

bhosts command

The **bhosts** command displays the status of LSF batch server hosts in the cluster, and other details about the batch hosts:

- Maximum number of job slots that are allowed by a single user
- Total number of jobs in the system, running jobs, jobs that are suspended by users, and jobs that are suspended by the system
- Total number of reserved job slots

Normal status **ok** for all hosts in your cluster.

```
% bhosts
HOST_NAME      STATUS      JL/U      MAX      NJOBS      RUN      SSUSP      USUSP      RSV
hosta          ok          -         -         0           0         0         0         0
hostb          ok          -         -         0           0         0         0         0
hostc          ok          -         -         0           0         0         0         0
hostd          ok          -         -         0           0         0         0         0
```

If you see the following message when you start or reconfigure LSF, wait a few seconds and try the **bhosts** command again to give the **mbatchd** daemon time to initialize.

```
batch system daemon not responding ... still trying
```

If the problem persists, see [Solving common LSF problems](#) for help.

bqueues command

LSF queues organize jobs with different priorities and different scheduling policies.

The **bqueues** command displays the status of available queues and their configuration parameters. For a queue to accept and dispatch jobs, the status must be **Open:Active**.

```
% bqueues
QUEUE_NAME      PRIO STATUS      MAX JL/U JL/P JL/H NJOBS PEND  RUN  SUSP
owners          43  Open:Active  -   -   -   -   0    0    0    0
priority        43  Open:Active  -   -   -   -   0    0    0    0
night           40  Open:Inact   -   -   -   -   0    0    0    0
chkpnt_rerun_qu 40  Open:Active  -   -   -   -   0    0    0    0
short           35  Open:Active  -   -   -   -   0    0    0    0
license         33  Open:Active  -   -   -   -   0    0    0    0
normal          30  Open:Active  -   -   -   -   0    0    0    0
idle            20  Open:Active  -   -   -   -   0    0    0    0
```

To see more detailed queue information, use the **bqueues -l** command:

```
% bqueues -l normal
```

```
QUEUE: normal
-- For normal low priority jobs, running only if hosts are lightly loaded. This is the
default queue.
```

PARAMETERS/STATISTICS

```
PRIO NICE STATUS      MAX JL/U JL/P JL/H NJOBS PEND  RUN  SSUSP USUSP RSV
30   20  Open:Active  -   -   -   -   0    0    0    0    0    0
Interval for a host to accept two jobs is 0 seconds
```

SCHEDULING PARAMETERS

```
          r15s  r1m  r15m  ut      pg    io    ls    it    tmp    swp    mem
loadSched -    -    -    -      -    -    -    -    -    -    -
loadStop  -    -    -    -      -    -    -    -    -    -    -
```

SCHEDULING POLICIES: FAIRSHARE NO_INTERACTIVE

USER_SHARES: [default, 1]

USERS: all

HOSTS: all

The **bqueues -l** command shows the following kinds of information about the queue:

- What kinds of jobs are meant to run on the queue
- Resource usage limits
- Hosts and users able to use the queue
- Scheduling threshold values:
 - **loadSched** is the threshold for LSF to stop dispatching jobs automatically
 - **loadStop** is the threshold for LSF to suspend a job automatically

Other useful commands

- The **bparams** command displays information about the LSF batch system configuration parameters.
- The **bhist** command displays historical information about jobs.

Run LSF jobs

Use the **bsub** and **lrun** commands to run jobs through LSF. Use the **bjobs** command to see the status of your jobs. Control job execution with the **bstop**, **bresume**, and **kill** commands.

Run LSF jobs with bsub and lrun

Use two basic commands to run jobs through LSF:

- **bsub** submits jobs to the LSF batch scheduler. LSF schedules and dispatches jobs to the best available host based on the scheduling policies you configure in your LSF queues.
- The **lrun** command runs an interactive task on the best available host, based on current system load information gathered by the **lim** daemon.

For most jobs, all you need to do is add either the **lrun** or **bsub** command in front of the job commands you normally use. You usually don't need to modify your executable applications or execution scripts.

- [Submit batch jobs with bsub](#)
The **bsub** command submits jobs to LSF batch scheduling queues.
- [Display job status with bjobs](#)
Use the **bjobs** command to see the job ID and other information about your jobs.
- [Control job execution with bstop, bresume, and bkill](#)
Use LSF commands to suspend (**bstop**), resume (**bresume**), and kill (**bkill**) jobs.
- [Run interactive tasks with lrun and lsgrun](#)
The **lrun** command runs a task on either the current local host or remotely on the best available host, provided it can find the necessary resources and the appropriate host type. The **lsgrun** command is similar to **lrun**, but it runs a task on a group of hosts.
- [Integrate your applications with LSF](#)
By integrating your applications with LSF, you can make sure that your users can submit and run their jobs with correct and complete job submission options without making them learn LSF commands.
- [Managing LSF Job Execution](#)
- [LSF Command Reference](#)

Submit batch jobs with bsub

The **bsub** command submits jobs to LSF batch scheduling queues.

The following command submits a **sleep** job to the default queue (**normal**):

```
% bsub sleep 60
Job <3616> is submitted to default queue <normal>.
```

When a job is submitted to LSF, it is assigned a unique job ID, in this case 3616.

You can specify a wide range of job options on the **bsub** command. For example, you can specify a queue, and the job command **sleep 60** is the last option:

```
% bsub -q short sleep 60
Job <3628> is submitted to queue <short>.
```

What LSF does with job output

By default, when the job is finished, LSF sends email with a job report and any output and error messages to the user account from which the job was submitted. You can optionally save standard output and standard error to files with the **-o** and **-e** options.

The following command appends the standard output and standard error of the job to the files **output.3640** and **errors.3640** in the **jobs** subdirectory of the home directory of **user1**.

```
% bsub -q short -o /home/user1/job/output.%J -e /home/user1/job/errors.%J ls -l
Job <3640> is submitted to queue <short>.
```

The **%J** variable is replaced by the job ID when the files are created. Using **%J** helps you find job output when you run a lot of jobs.

Interactive batch jobs with bsub -I

To submit an interactive job through LSF, use the `-I` option:

The following command submits a batch interactive job that displays the output of the `ls` command:

```
% bsub -I ls
```

To submit a batch interactive job by using a pseudo-terminal, use the `bsub -Ip` option.

To submit a batch interactive job and create a pseudo-terminal with shell mode support, use the `bsub -Is` option.

Display job status with `bjobs`

Use the `bjobs` command to see the job ID and other information about your jobs.

The status of each LSF job is updated periodically.

```
% bjobs
JOBID USER      STAT  QUEUE   FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
1266  user1        RUN   normal  hosta      hostb      sleep 60   Jun 5 17:39:58
```

The job that is named `sleep 60` runs for 60 seconds. When the job completes, LSF sends email to report the job completion.

You can use the job ID to monitor the status of a specific job.

If all hosts are busy, the job is not started immediately and the `STAT` column says `PEND`.

Control job execution with `bstop`, `bresume`, and `bkill`

Use LSF commands to suspend (`bstop`), resume (`bresume`), and kill (`bkill`) jobs.

`bstop` command

To suspend a running job, use the `bstop` command and specify the job ID:

```
% bstop 1266
Job <1266> is being stopped
```

If the job was running when it was stopped, the `bjobs` command shows `USUSP` status for job 1266:

```
% bjobs
JOBID USER      STAT  QUEUE   FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
1266  user1      USUSP  normal  hosta      hostb      sleep 60   Jun 5 17:39:58
```

Job owners can suspend only their own jobs. LSF administrators can suspend any job.

`bresume` command

To resume a suspended job, use the `bresume` command.

```
% bresume 1266
Job <1266> is being resumed
```

If the job resumes immediately, the `bjobs` command shows `RUN` status for job 1266:

```
% bjobs
JOBID USER      STAT  QUEUE   FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
1266  user1        RUN   normal  hosta      hostb      sleep 60   Jun 5 17:39:58
```

Job owners can resume only their own jobs. LSF administrators can resume any job.

bkill command

To kill a job, use the **bkill** command, which sends a signal to the specified jobs. For example, if the job owner or the LSF administrator runs the following command, job 1266 is killed:

```
% bkill 1266
Job <1266> is being terminated
```

Run interactive tasks with lsrun and lsgrun

The **lsrun** command runs a task on either the current local host or remotely on the best available host, provided it can find the necessary resources and the appropriate host type. The **lsgrun** command is similar to **lsrun**, but it runs a task on a group of hosts.

The following command runs the **ls** command. In this case, the command ran through LSF on the local host:

```
% lsrun ls -l /usr/share/lsf/cluster1/conf/
total 742
-rw-r--r-- 1 root lsf 11372 Jul 16 16:23 cshrc.lsf
-rw-r--r-- 1 root lsf 365 Oct 25 10:55 hosts
drwxr-xr-x 3 lsfadmin lsf 512 Jul 16 15:53 lsbatch
-rw-r--r-- 1 lsfadmin lsf 1776 Nov 23 15:13 lsf.conf
-rw-r--r-- 1 lsfadmin lsf 8453 Nov 16 17:46 lsf.shared
-rw-r--r-- 1 root lsf 10485 Jul 16 17:08 profile.lsf
```

You can also specify a host where you want to run a command. For example, the following command runs the **hostname** command on the remote host **hosta**:

```
% lsrun -v -m hosta hostname
<<Execute hostname on remote host hosta>>
hosta
```

The following command runs the **hostname** command on three remote hosts:

```
% lsgrun -v -m "hosta hostb hostc" hostname
<<Executing hostname on hosta>>
hosta
<<Executing hostname on hostb>>
hostb
<<Executing hostname on hostc>>
hostc
```

Integrate your applications with LSF

By integrating your applications with LSF, you can make sure that your users can submit and run their jobs with correct and complete job submission options without making them learn LSF commands.

Integrate applications with LSF three ways:

- Wrapper shell scripts
- Wrapper binary executables
- Modifying existing application source code and interfaces

Wrapper shell scripts

The easiest integration method is to put the **bsub** command into an executable file like a shell script. A wrapper script is an executable file for launching your application through LSF. It gives users a simple interface to run their jobs that is easy to deploy and maintain.

For example, if your application is called **abc**, rename **abc** to **abc_real** and create a wrapper script that is called **abc**:

```
#!/bin/sh
bsub -R "rusage[abc_license=1:duration=1]" abc_real
```

When users run `abc`, they are actually running a script to submit a job `abc_real` to LSF that uses 1 shared resource named `abc_license`.

For more information about specifying shared resources by using the resource requirement (`rusage`) string on the `-R` option of the `bsub` command, see [Manage software licenses and other shared resources](#).

By adding appropriate options to the script, you can enhance your integration:

- Requeue jobs based on license availability
- Copy input and output files to and from the local directory on the execution host
- Calculate and estimate resource requirements

Wrapper binary programs

A wrapper binary is similar to a wrapper shell script in the form of a compiled binary executable. Compiled wrapper files usually run faster and more efficiently than shell scripts, and they also have access to the LSF API (LSLIB and LSBLIB). Binary code is also more secure because users cannot modify it without the source code and appropriate libraries, but it is more time consuming to develop wrapper binary programs than wrapper shell scripts.

Modifying existing application source code and interfaces

LSF is already integrated closely with many commonly used software products. IBM and other software application vendors provide facilities and services for closer integration of LSF and other applications. By modifying existing application user interfaces, you can enable easy job submission, license maximization, parallel execution, and other advanced LSF features. In some cases, you are able to run an LSF job directly from the application user interface.

Where to go next

Learn more about administering your cluster, described in [Manage users, hosts, and queues](#).

Manage users, hosts, and queues

Make your cluster available to users with `cschrc.lsf` and `profile.lsf`. Add or remove hosts and queues from your cluster.

- [Making your cluster available to users with `cschrc.lsf` and `profile.lsf`](#)
Make sure that all LSF users include either the `cschrc.lsf` or `profile.lsf` file at the end of their own `.cschrc` or `.profile` file, or run one of these two files before you use LSF.
- [Adding a host to your cluster](#)
Use the LSF installation script `lsfinstall` to add new hosts and host types to your cluster.
- [Removing a host from your cluster](#)
Removing a host from LSF involves closing a host to prevent any additional jobs from running on the host and removing references to the host from the `lsf.cluster.cluster_name` file and other configuration files.
- [Adding a queue](#)
Edit the `lsb.queues` file to add the new queue definition. Adding a queue does not affect pending or running jobs.
- [Removing a queue](#)
Edit `lsb.queues` to remove a queue definition.

Making your cluster available to users with `cschrc.lsf` and `profile.lsf`

Make sure that all LSF users include either the `cschrc.lsf` or `profile.lsf` file at the end of their own `.cschrc` or `.profile` file, or run one of these two files before you use LSF.

About this task

To set up the LSF environment for your users, use the following two shell files:

LSF_CONFDIR/`cschrc.lsf`

Use this file for **csch** or **tcsh** shell.

LSF_CONFDIR/`profile.lsf`

Use this file for **sh**, **ksh**, or **bash** shell.

Procedure

For **csch** or **tcsh** shell:

- Add the `cschrc.lsf` file to the end of the `.cschrc` file for all users:
 - Copy the contents of the `cschrc.lsf` file into the `.cschrc` file.
 - Add a line with the **source** command to the end of the `.cschrc` file:
For example, if your the *LSF_TOP* directory for your cluster is `/usr/share/lsf/conf`, add the following line to the `.cschrc` file:

```
source /usr/share/lsf/conf/cschrc.lsf
```

For **sh**, **ksh**, or **bash** shell:

- Add the `profile.lsf` file to the end of the `.profile` file for all users:
 - Copy the contents of the `profile.lsf` file into the `.profile` file.
 - For example, if your the *LSF_TOP* directory for your cluster is `/usr/share/lsf/conf`, add a line similar to the following to the end of the `.profile` file:

```
. /usr/share/lsf/conf/profile.lsf
```

Adding a host to your cluster

Use the LSF installation script **lsfinstall** to add new hosts and host types to your cluster.

Before you begin

Make sure that you have the LSF distribution files for the host types you want to add. For example, to add a Linux system that runs x86-64 Kernel 2.6 and 3.x to your cluster, get the file `lsf10.1.0_linux2.6-glibc2.3-x86_64.tar.Z`.

Distribution packages for all supported LSF releases are available for download through [IBM Passport Advantage](#).

See [LSF System Requirements](#) on IBM developerWorks for a complete list of supported operating systems.

The following videos provide more help about downloading LSF through IBM Passport Advantage:

- [YouTube](#)
- [IBM Education Assistant](#)

About this task

Adding a host to your cluster has the following major steps:

1. Install LSF binary files for the host type.
2. Add host information to the `lsf.cluster.cluster_name` file.
3. Set up the new host.

Procedure

1. Install the binary files for a new host type.

Use the **lsfinstall** command to add new host types to your cluster. If you already have the distribution files for the host types you want to add, you can skip these steps.

- a. Log on as root to any host that can access the LSF installation script directory.
- b. Change to the installation script directory.

```
# cd /usr/share/lsf/cluster1/10.1.0/install
```

- c. Edit the install.config file to specify the options you want for new host types.
For more information about the install.config file, see the [IBM® Spectrum LSF Configuration Reference](#). For information about the **lsfinstall** command, see [Installing IBM Spectrum LSF on UNIX and Linux](#) and the [IBM Spectrum LSF Command Reference](#).
- d. Run the `./lsfinstall -f install.config` command.
- e. Follow the steps for host setup in [After Installing LSF](#) in [Installing IBM Spectrum LSF on UNIX and Linux](#) (or in the `lsf_getting_started.html` file that is generated by the **lsfinstall** script) to set up the new hosts.

2. Add host information to the `lsf.cluster.cluster_name` file.

- a. Log on to the LSF management host as the primary LSF administrator.
- b. Edit the `LSF_CONFDIR/lsf.cluster.cluster_name` file, and add host information for the new host to the Host section.

- Add the name of the host.
- Add model or type.

If you enter the `!` keyword in the `model` and `type` columns, the host model is automatically detected by **lim** running on the host.

You might want to use the default values for that host type now, and change them later on when you have more experience or more information.

- Specify LSF server or client in the `server` column:
 - 1 (one) indicates an LSF server host.
 - 0 (zero) indicates an LSF client-only host.

By default, all hosts are considered LSF server hosts.

HOSTNAME	model	type	server	rlm	mem	RESOURCES	REXPRI
hosta	!	SUNSOL	1	1.0	4	()	0
hostb	!	LINUX	0	1.0	4	()	0
hostc	!	HPPA	1	1.0	4	()	0
End Host							

- c. Save the changes to `LSF_CONFDIR/lsf.cluster.cluster_name`.
- d. Reconfigure **lim** to enable the new host in the cluster.

```
% lsadmin reconfig
Checking configuration files ...
No errors found.
Do you really want to restart LIMs on all hosts? [y/n] y
Restart LIM on <hosta> ..... done
Restart LIM on <hostc> ..... done
Restart LIM on <hostd> ..... done
```

The **lsadmin reconfig** command checks for configuration errors. If no unrecoverable errors are found, you are asked to confirm that you want to restart **lim** on all hosts and **lim** is reconfigured. If unrecoverable errors are found, reconfiguration exits.

- e. Reconfigure **mbatchd**.

```
% badmin reconfig
Checking configuration files ...
No errors found.
Do you want to reconfigure? [y/n] y
Reconfiguration initiated
```

The **badmin reconfig** command checks for configuration errors. If no unrecoverable errors are found, you are asked to confirm reconfiguration. If unrecoverable errors are found, reconfiguration exits.

3. (Optional) Use the **hostsetup** command to set up the new host.
 - a. Log on as root to any host that can access the LSF installation script directory.
 - b. Change to the installation script directory.

```
# cd /usr/share/lsf/cluster1/10.1.0/install
```

- c. Run the **hostsetup** command to set up the new host.

```
# ./hostsetup --top="/usr/share/lsf/lsf_62" --boot="y"
```

For information about the **hostsetup** command, see [Installing IBM Spectrum LSF on UNIX and Linux](#) and the [IBM Spectrum LSF Command Reference](#).

- d. Start LSF on the new host.
Run the following commands:

```
# bctrld start lim
# bctrld start res
# bctrld start sbd
```

- e. Run the **bhosts** and **lshosts** commands to verify your changes.
If any host type or host model is UNKNOWN or DEFAULT, see [Working with hosts](#) in [IBM Spectrum LSF Cluster Management and Operations](#) to fix the problem.

Results

- Use dynamic host configuration to add hosts to the cluster without manually changing the LSF configuration. For more information about adding hosts dynamically, see [IBM Spectrum LSF Cluster Management and Operations](#).
- If you get errors, see [Troubleshooting LSF problems](#) for help with some common configuration errors.

Removing a host from your cluster

Removing a host from LSF involves closing a host to prevent any additional jobs from running on the host and removing references to the host from the `lsf.cluster.cluster_name` file and other configuration files.

About this task

CAUTION:

Never remove the management host from LSF. If you want to change your current default management host, change the `lsf.cluster.cluster_name` file to assign a different default management host. Then remove the host that was formerly the management host.

Procedure

1. Log on to the LSF host as root.
2. Run **badmin hclose** to close the host.
Closing the host prevents jobs from being dispatched to the host and allows running jobs to finish.
3. Stop all running daemons manually.
4. Remove any references to the host in the `Host` section of the `LSF_CONFDIR/lsf.cluster.cluster_name` file.
5. Remove any other references to the host, if applicable, from the following configuration files:
 - `LSF_CONFDIR/lsf.shared`
 - `LSB_CONFDIR/cluster_name/configdir/lsb.hosts`
 - `LSB_CONFDIR/cluster_name/configdir/lsb.queues`
 - `LSB_CONFDIR/cluster_name/configdir/lsb.resources`
6. Log off the host to be removed, and log on as `root` or the primary LSF administrator to any other host in the cluster.
7. Run the **lsadmin reconfig** command to reconfigure LIM.

```
% lsadmin reconfig
Checking configuration files ...
No errors found.
```

```
Do you really want to restart LIMs on all hosts? [y/n] y
Restart LIM on <hosta> ..... done
Restart LIM on <hostc> ..... done
```

The **lsadmin reconfig** command checks for configuration errors.

If no errors are found, you are asked to confirm that you want to restart **lim** on all hosts and **lim** is reconfigured. If unrecoverable errors are found, reconfiguration exits.

8. Run the **badmin mbdrestart** command to restart **mbatchd**.

```
% badmin reconfig
Checking configuration files ...
No errors found.
Do you want to reconfigure? [y/n] y
Reconfiguration initiated
```

The **badmin mbdrestart** command checks for configuration errors.

If no unrecoverable errors are found, you are asked to confirm reconfiguration. If unrecoverable errors are found, reconfiguration exits.

9. If you configured LSF daemons to start automatically at system startup, remove the **LSF** section from the host's system startup files.

For more information about automatic LSF daemon startup, see [Setting up automatic LSF startup](#)

Results

- Use dynamic host configuration to remove hosts to the cluster without manually changing the LSF configuration. For more information about removing hosts dynamically, see [IBM Platform LSF Cluster Management and Operations](#).
- If you get errors, see [./lsf_admin/chap_troubleshooting_lsf.html#v3523448](#) for help with some common configuration errors.

Adding a queue

Edit the **lsb.queues** file to add the new queue definition. Adding a queue does not affect pending or running jobs.

Procedure

1. Log in as the administrator on any host in the cluster.
2. Edit the **LSB_CONFDIR/cluster_name/configdir/lsb.queues** file to add the new queue definition.
You can copy another queue definition from this file as a starting point. Remember to change the **QUEUE_NAME** parameter of the copied queue.
3. Save the changes to the **lsb.queues** file.
4. When the configuration files are ready, run the **badmin ckconfig** command to check the new queue definition.
If any errors are reported, fix the problem and check the configuration again.
5. Run the **badmin reconfig** command to reconfigure the cluster.

```
% badmin reconfig
Checking configuration files ...
No errors found.
Do you want to reconfigure? [y/n] y
Reconfiguration initiated
```

The **badmin reconfig** command also checks for configuration errors. If no unrecoverable errors are found, you are asked to confirm reconfiguration. If unrecoverable errors are found, reconfiguration exits.

Results

If you get errors, see [Troubleshooting LSF problems](#) for help with some common configuration errors.

- For more information about the [lsb.queues](#) file, see the [Configuration Reference](#).
- For more information about the [badmin reconfig](#) command, see the [Command Reference](#).

Example

```
Begin Queue
QUEUE_NAME = normal
PRIORITY = 30
STACKLIMIT= 2048
DESCRIPTION = For normal low priority jobs, running only if hosts are lightly loaded.
QJOB_LIMIT = 60      # job limit of the queue
PJOB_LIMIT = 2       # job limit per processor
ut = 0.2
io = 50/240
USERS = all
HOSTS = all
NICE = 20
End Queue
```

Removing a queue

Edit `lsb.queues` to remove a queue definition.

Before you begin

Important: Before you remove a queue, make sure that no jobs are running in the queue.

Use the **bqueues** command to view a list of existing queues and the jobs that are running in those queues. If jobs are in the queue that you want to remove, you must switch pending and running jobs to another queue, then remove the queue. If you remove a queue that has pending jobs in it, the jobs are temporarily moved to a `lost_and_found` queue. The job state does not change. Running jobs continue, and jobs that are pending in the original queue are pending in the `lost_and_found` queue. Jobs remain pending until the user or the queue administrator uses the **bswitch** command to switch the jobs into a regular queue. Jobs in other queues are not affected.

Procedure

1. Log in as the primary administrator on any host in the cluster.
2. Close the queue to prevent any new jobs from being submitted.

```
badmin qclose night
Queue night is closed
```

3. Switch all pending and running jobs into another queue.

For example, the **bswitch -q night idle 0** command chooses jobs from the `night` queue to the `idle` queue. The job ID number 0 switches all jobs.

```
bjobs -u all -q night
JOBID USER  STAT  QUEUE FROM_HOST EXEC_HOST  JOB_NAME  SUBMIT_TIME
5308 user5  RUN   night hostA   hostD     job5      Nov 21 18:16
5310 user5  PEND  night hostA   hostC     job10     Nov 21 18:17

bswitch -q night idle 0
Job <5308> is switched to queue <idle>
Job <5310> is switched to queue <idle>
```

4. Edit the `LSB_CONFDIR/cluster_name/configdir/lsb.queues` file and remove or comment out the definition for the queue that you want to remove.
5. Save the changes to the `lsb.queues` file.
6. Run the **badmin reconfig** command to reconfigure the cluster.

```
% badmin reconfig
Checking configuration files ...
```

```
No errors found.  
Do you want to reconfigure? [y/n] y  
Reconfiguration initiated
```

The **badmin reconfig** command checks for configuration errors. If no unrecoverable errors are found, you are asked to confirm reconfiguration. If unrecoverable errors are found, reconfiguration exits.

Results

If you get errors, see [Troubleshooting LSF problems](#) for help with some common configuration errors.

- For more information about the [lsb.queues](#) file, see the [Configuration Reference](#).
- For more information about the **badmin reconfig** command, see the [Command Reference](#).

Configure LSF startup

Use the `lsf.sudoers` file so that LSF administrators can start and stop LSF daemons. Set up LSF to start automatically.

- [Allowing LSF administrators to start LSF daemons with `lsf.sudoers`](#)
To allow LSF administrators to start and stop LSF daemons, configure the `/etc/lsf.sudoers` file. If the `lsf.sudoers` file does not exist, only root can start and stop LSF daemons.
- [Setting up automatic LSF startup](#)
Configure LSF daemons to start automatically on every LSF server host in the cluster.

Related concepts

- [Installing LSF on UNIX and Linux](#)

Related reference

- [lsf.sudoers](#)
- [LSF daemon startup control](#)

Allowing LSF administrators to start LSF daemons with `lsf.sudoers`

To allow LSF administrators to start and stop LSF daemons, configure the `/etc/lsf.sudoers` file. If the `lsf.sudoers` file does not exist, only root can start and stop LSF daemons.

About this task

Using the `lsf.sudoers` file requires you to enable the `setuid` bit. Since this allows LSF administration commands to run with root privileges, do not proceed if you do not want these commands to run with root privileges.

Procedure

1. Log on as root to each LSF server host.
Start with the LSF management host, and repeat these steps on all LSF hosts.
2. Create an `/etc/lsf.sudoers` file on each LSF host and specify the `LSF_STARTUP_USERS` and `LSF_STARTUP_PATH` parameters.

```
LSF_STARTUP_USERS="lsfadmin user1"
LSF_STARTUP_PATH=/usr/share/lsf/cluster1/10.1.0/sparc-sol2/etc
```

LSF_STARTUP_PATH is normally the path to the LSF_SERVERDIR directory, where the LSF server binary files (**lim**, **res**, **sbatchd**, **mbatchd**, **mbschd**, and so on) are installed, as defined in your LSF_CONFDIR/lsf.conf file.

The lsf.sudoers file must have file permission mode `-rw-----` (600) and be readable and writable only by root:

```
# ls -la /etc/lsf.sudoers
-rw----- 1 root lsf 95 Nov 22 13:57 lsf.sudoers
```

3. Run **hostsetup --top="/path" --setuid** to enable the set-user-ID mode bit on the LSF administration commands, where *path* is the LSF top-level installation directory that contains the cluster to which the local host belongs. The path must be accessible to the local host where **hostsetup** is running. Additionally, by default, the **hostsetup** command is not added to the system PATH, so specify a full or relative path to run it.
For example, if your **hostsetup** command is in the */LSF_TOP/10.1/install* directory, then specify the full path to the command and run:

```
# /LSF_TOP/10.1/install/hostsetup --top="/LSF_TOP/10.1/install" --setuid
```

Alternatively, you can switch to the directory where you have the **hostsetup** command to run the command; in this case, preface the command with `./`. For example, switch to the */LSF_TOP/10.1/install* directory, and then run:

```
# ./hostsetup --top="/LSF_TOP/10.1/install" --setuid
```

The `--setuid` setting enables the set-user-ID mode bit for the following LSF executable files: **badmin**, **lsadmin**, **egosh**, **utmpreg**, **swtbl_api**, **ntbl_api**, **lstbl_nid**, and **swtbl_poe**.

4. Run the **lsfrestart** command to restart the cluster:

```
# lsfrestart
```

Setting up automatic LSF startup

Configure LSF daemons to start automatically on every LSF server host in the cluster.

Procedure

Use the `boot=y` option of the **hostsetup** command.

Manage software licenses and other shared resources

Set up an LSF *external LIM* (**ELIM**) to monitor software licenses as dynamic shared resources.

How LSF uses dynamic shared resources

LSF recognizes two main types of resources:

- Host-based resources are available on all hosts in the cluster, for example, host type and model, or nodelocked software licenses.
- Shared resources are managed as dynamic load indexes available for a group of hosts in the cluster, for example, networked floating software licenses, shared file systems.

Shared resources are shared by a group of LSF hosts. LSF manages shared resources for host selection and batch or interactive job execution. These resources are *dynamic resources* because the load on the system changes with the availability of the resources.

Software licenses as shared resources

The most common application of shared resources is to manage software application licenses. You submit jobs that require those licenses and LSF runs the jobs according to their priorities when licenses are available. When licenses are not available, LSF queues the jobs then dispatches them when licenses are free. Configuring application licenses as shared resources ensures optimal use of costly and critical resources.

Define dynamic shared resources in an ELIM

For LSF to use a shared resource like a software license, you must define the resource in the `Resource` section of the `lsf.shared` file. You define the type of resource and how often you want LSF to refresh the value of the resource.

For LSF to track the resources correctly over time, you must define them as external load indexes. LSF updates load indexes periodically with a program called an External Load Information Manager (ELIM).

An ELIM can be a shell script or a compiled binary program, which returns the values of the shared resources you define. The ELIM must be named `elim` and located in the `LSF_SERVERDIR` directory:

```
/usr/share/lsf/lsf/cluser1/10.1.0/sparc-sol2/etc/elim
```

You can find examples of **ELIM** in the `misc/examples` directory.

Example of shared licenses

In the `lsf.shared` file, define two dynamic shared resources for software licenses, named `license1` and `license2`:

```
Begin Resource
RESOURCENAME  TYPE    INTERVAL  INCREASING  RELEASE  DESCRIPTION  # Keywords
license1      Numeric 30        N           Y        (license1 resource)
license2      Numeric 30        N           Y        (license2 resource)
End Resource
```

- The **TYPE** parameter for a shared resource can be one of the following types:
 - **Numeric**
 - **Boolean**
 - **String**

In this case, the resource is **Numeric**.

- The **INTERVAL** parameter specifies how often you want the value to be refreshed. In this example, the ELIM updates the value of the shared resources `license1` and `license2` every 30 seconds.
- The **N** in the **INCREASING** column means that the license resources are decreasing; that is, as more licenses become available, the load becomes lower.
- The **Y** in the **RELEASE** column means that the license resources are released when a job that uses the license is suspended.

Map dynamic shared resources to hosts

To make LSF aware of where the defined dynamic shared resources `license1` and `license2` you defined, map them to the hosts where they are located.

In the `LSF_CONFDIR/lsf.cluster.cluster_name` file, configure a `ResourceMap` section to specify the mapping between shared resources `license1` and `license2` you defined in the `LSF_CONFDIR/lsf.shared` file, and the hosts you want to map them to:

```
Begin ResourceMap
RESOURCENAME  LOCATION
license1      [all]
license1      [all]
End ResourceMap
```

In this resource map, the `[all]` attribute under the **LOCATION** parameter means that resources `license1` and `license2` under the **RESOURCENAME** parameter are available on all hosts in the cluster. Only one ELIM needs to run on the management

host because the two resources are the same for all hosts. If the location of the resources is different on different hosts, a different ELIM must run on every host.

Monitor dynamic shared resources

For LSF to receive external load indexes correctly, the ELIM must send a count of the available resources to standard output in the following format:

```
number_indexes [index_name index_value] ...
```

The fields in this example contain `2 license1 3 license2 2`, which represents:

- The total number of external load indexes (2)
- The name of the first external load index (`license1`)
- The value of the first load index (3)
- The name of the second external load index (`license2`)
- The value of the second load index (2)

Write the ELIM program

The ELIM must be an executable program, named `elim`, located in the `LSF_SERVERDIR` directory.

When the `lim` daemon is started or restarted, it runs the `elim` program on the same host and takes the standard output of the external load indexes that are sent by the `elim` program. In general, you can define any quantifiable resource as an external load index, write an ELIM to report its value, and use it as an LSF resource.

The following example `ELIM` program uses `license1` and `license2` and assumes that the **FLEXlm** license server controls them:

```
#!/bin/sh
NUMLIC=2          # number of dynamic shared resources
while true
do
  TMPLICS='/usr/share/lsf/cluster1/10.1.0/sparc-sol2/etc/lic -c
/usr/share/lsf/cluster1/conf/license.dat -f license1, license2'

  LICS='echo $TMPLICS | sed -e s/-/_/g'
  echo $NUMLIC $LICS # $NUMLIC is number of dynamic shared
  resource
  sleep 30          # Resource
done
```

In the script, the `sed` command changes the minus sign (`-`) to underscore (`_`) in the license feature names because LSF uses the minus sign for calculation, and it is not allowed in resource names.

The `lic` utility is available from IBM Support. You can also use the **FLEXlm** command `lmstat` to make your own ELIM.

Use the dynamic shared resources

To enable the new shared resources in your cluster, restart LSF with the following commands:

- `lsadmin reconfig`
- `badmin reconfig`

If no errors are found, use the `lsload -l` command to verify the value of your dynamic shared resources:

HOST_NAME	status	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem	license1	license2
hosta	ok	0.1	0.3	0.4	8%	0.2	50	73	0	62M	700M	425M	3	0
hostb	ok	0.1	0.1	0.4	4%	5.7	3	3	0	79M	204M	64M	3	0

Submit jobs that use shared resources

To submit a batch job that uses one `license1` resource, use the command following command:


```
% bsub -R 'rusage[license1=1:duration=1]' myjob
```

In the resource requirement (`rusage`) string, `duration=1` means that `license1` is reserved for 1 minute to give LSF time to check it out from **FLEXlm**.

You can also specify the resource requirement string at queue level, in the `RES_REQ` parameter for the queue. In the `LSB_CONFDIR/cluster_name/configdir/lsb.queues` file, specify the following resource requirement string:

```
Begin Queue
QUEUE_NAME = license1
RES_REQ=rusage[license1=1:duration=1]
...
End Queue
```

Then, submit a batch job that uses one `license1` resource by using the following command:

```
% bsub -q license1 myjob
```

When licenses are available, LSF runs your jobs right away; when all licenses are in use, LSF puts your job in a queue and dispatches them as licenses become available. This way, all of your licenses are used to the best advantage.

For more information

- For more information about the [lsf.shared](#) and [lsf.cluster.cluster_name](#) files and the parameters for configuring shared resources, see the [Configuration Reference](#).
- For more information about adding external resources to your cluster and configuring an ELIM to customize resources, see [External load indices](#) in [Administering IBM® Spectrum LSF](#).

Troubleshooting LSF problems

Troubleshoot common LSF problems and understand LSF error messages. If you cannot find a solution to your problem here, contact IBM Support.

- [Solving common LSF problems](#)
Most problems are due to incorrect installation or configuration. Before you start to troubleshoot LSF problems, always check the error log files first. Log messages often point directly to the problem.
- [LSF error messages](#)
The following error messages are logged by the LSF daemons, or displayed by the **lsadmin ckconfig** and **badmin ckconfig** commands.

Solving common LSF problems

Most problems are due to incorrect installation or configuration. Before you start to troubleshoot LSF problems, always check the error log files first. Log messages often point directly to the problem.

Finding LSF error logs

When something goes wrong, LSF server daemons log error messages in the LSF log directory (specified by the `LSF_LOGDIR` parameter in the `lsf.conf` file).

Procedure

Make sure that the primary LSF administrator owns `LSF_LOGDIR`, and that root can write to this directory.

If an LSF server is unable to write to `LSF_LOGDIR`, then the error logs are created in `/tmp`. LSF logs errors to the following files:

- `lim.log.host_name`
- `res.log.host_name`

- `pim.log.host_name`
- `mbatchd.log.management_host`
- `mbschd.log.management_host`
- `sbatchd.log.management_host`
- `vemkd.log.management_host`

If these log files contain any error messages that you do not understand, contact IBM Support.

Diagnosing and fixing most LSF problems

General troubleshooting steps for most LSF problems.

Procedure

1. Run the **lsadmin ckconfig -v** command and note any errors that are shown in the command output. Look for the error in one of the problems described in this section. If none of these troubleshooting steps applies to your situation, contact IBM Support.
2. Use the following commands to restart the LSF cluster:

```
# bctrld restart lim all
# bctrld restart res all
# bctrld restart sbd all
```

3. Run the **ps -ef** command to see whether the LSF daemons are running. Look for the processes similar to the following command output:

```
root 17426      1  0   13:30:40 ?        0:00 /opt/lsf/cluster1/10.1.0/sparc-sol10/etc/lim
root 17436      1  0   13:31:11 ?        0:00 /opt/lsf/cluster1/10.1.0/sparc-sol10/etc/sbatchd
root 17429      1  0   13:30:56 ?        0:00 /opt/lsf/cluster1/10.1.0/sparc-sol10/etc/res
```

4. Check the LSF error logs on the first few hosts that are listed in the **Host** section of the `LSF_CONFDIR/lsf.cluster.cluster_name` file. If the `LSF_MASTER_LIST` parameter is defined in the `LSF_CONFDIR/lsf.conf` file, check the error logs on the hosts that are listed in this parameter instead.

Cannot open the lsf.conf file

You might see this message when you run the **lsid** file. The message usually means that the `LSF_CONFDIR/lsf.conf` file is not accessible to LSF.

About this task

By default, LSF checks the directory that is defined by the `LSF_ENVDIR` parameter for the `lsf.conf` file. If the `lsf.conf` file is not in `LSF_ENVDIR`, LSF looks for it in the `/etc` directory.

For more information, see [Setting up the LSF environment with `cschrc.lsf` and `profile.lsf`](#).

Procedure

- Make sure that a symbolic link exists from `/etc/lsf.conf` to `lsf.conf`
- Use the `cschrc.lsf` or `profile.lsf` script to set up your LSF environment.
- Ensure that the `cschrc.lsf` or `profile.lsf` script is available for users to set the LSF environment variables.

LIM dies quietly

When the LSF **LIM** daemon exits unexpectedly, check for errors in the **LIM** configuration files.

Procedure

Run the **lsadmin ckconfig -v** commands.

This command displays most configuration errors. If the command does not report any errors, check in the **LIM** error log.

LIM communication times out

Sometimes the **LIM** is up, but running the **lsload** command prints the `Communication time out` error message.

About this task

If the **LIM** just started, **LIM** needs time to get initialized by reading configuration files and contacting other instances of **LIM**. If the **LIM** does not become available within one or two minutes, check the **LIM** error log for the host you are working on.

To prevent communication timeouts when the local **LIM** is starting or restarting, define the parameter `LSF_SERVER_HOSTS` in the `lsf.conf` file. The client contacts the **LIM** on one of the `LSF_SERVER_HOSTS` and runs the command. At least one of the hosts that are defined in the list must have a **LIM** that is up and running.

When the local **LIM** is running but the cluster has no management host, LSF applications display the `Cannot locate master LIM now, try later. message`.

Procedure

Check the **LIM** error logs on the first few hosts that are listed in the **Host** section of the `lsf.cluster.cluster_name` file. If the `LSF_MASTER_LIST` parameter is defined in the `lsf.conf` file, check the **LIM** error logs on the hosts that are listed in this parameter instead.

Management host LIM is down

Sometimes the management host **LIM** is up, but running the **lsload** or **lshosts** command displays the `Master LIM is down; try later. message`.

About this task

If the `/etc/hosts` file on the host where the management host **LIM** is running is configured with the host name that is assigned to the loopback IP address (127.0.0.1), LSF client **LIM** cannot contact the management host **LIM**. When the management host **LIM** starts up, it sets its official host name and IP address to the loopback address. Any client requests get the management host **LIM** address as 127.0.0.1, and try to connect to it, and in fact tries to access itself.

Procedure

Check the IP configuration of your management host **LIM** in `/etc/hosts`.

The following example incorrectly sets the management host **LIM** IP address to the loopback address:

```
127.0.0.1    localhost    myhostname
```

The following example correctly sets the management host LIM IP address:

```
127.0.0.1    localhost
192.168.123.123 myhostname
```

For a management host **LIM** running on a host that uses an IPv6 address, the loopback address is `::1`.

The following example correctly sets the management host **LIM** IP address by using an IPv6 address:

```
::1    localhost    ipv6-localhost    ipv6-loopback
fe00::0    ipv6-localnet
ff00::0    ipv6-mcastprefix
ff02::1    ipv6-allnodes
ff02::2    ipv6-allrouters
ff02::3    ipv6-allhosts
```

User permission denied

If the remote host cannot securely determine the user ID of the user that is requesting remote execution, remote execution fails with an `User permission denied` error message.

Procedure

1. Check the **RES** error log on the remote host for more detailed error message.
2. If you do not want to configure an identification daemon (LSF_AUTH in lsf.conf), all applications that do remote executions must be owned by root with the **setuid** bit set. Run the **chmod 4755 filename** command.
3. If the application binary files are on an NFS-mounted file system, make sure that the file system is not mounted with the **nosuid** flag.
4. If you are using an identification daemon (the LSF_AUTH parameter in the lsf.conf file), the **inetd** daemon must be configured. The identification daemon must not be run directly.
5. Inconsistent host names in a name server with /etc/hosts and /etc/hosts.equiv can also cause this problem. If the LSF_USE_HOSTEQUIV parameter is defined in the lsf.conf file, check that the /etc/hosts.equiv file or the HOME/.rhosts file on the destination host has the client host name in it.
6. For Windows hosts, users must register and update their Windows passwords by using the **lspasswd** command. Passwords must be 3 characters or longer, and 31 characters or less.
For Windows password authentication in a non-shared file system environment, you must define the parameter LSF_MASTER_LIST in the lsf.conf file so that jobs run with correct permissions. If you do not define this parameter, LSF assumes that the cluster uses a shared file system environment.

Remote execution fails because of non-uniform file name space

A non-uniform file name space might cause a command to fail with the `chdir(...) failed: no such file or directory` message.

About this task

You are trying to run a command remotely, but either your current working directory does not exist on the remote host, or your current working directory is mapped to a different name on the remote host.

If your current working directory does not exist on a remote host, do not run commands remotely on that host.

Procedure

- If the directory exists, but is mapped to a different name on the remote host, you must create symbolic links to make them consistent.
- LSF can resolve most, but not all, problems by using **automount**. The automount maps must be managed through NIS. Contact IBM Support if you are running automount and LSF is not able to locate directories on remote hosts.

Batch daemons die quietly

When the LSF batch daemons **sbatchd** and **mbatchd** exit unexpectedly, check for errors in the configuration files.

About this task

If the **mbatchd** daemon is running but the **sbatchd** daemon dies on some hosts, it might be because **mbatchd** is not configured to use those hosts.

Procedure

- Check the **sbatchd** and **mbatchd** daemon error logs.
- Run the **admin ckconfig** command to check the configuration.
- Check for email in the LSF administrator mailbox.

sbatchd starts but mbatchd does not

When the **sbatchd** daemon starts but the **mbatchd** daemon is not running, it is possible that **mbatchd** is temporarily unavailable because the management host LIM is temporarily unknown. The **sbatchd: unknown service** error message displays.

Procedure

1. Run the **lsid** command to check whether **LIM** is running.

If LIM is not running properly, follow the steps in the following topics to fix **LIM** problems:

- [LIM dies quietly](#)
- [LIM communication times out](#)
- [Management host LIM is down](#)

2. Check whether services are registered properly.

Avoiding orphaned job processes

LSF uses process groups to track all the processes of a job. However, if the application forks a child, the child becomes a new process group. The parent dies immediately, and the child process group is orphaned from the parent process, and cannot be tracked.

About this task

For more information about process tracking with Linux cgroups, see [Memory and swap limit enforcement based on Linux cgroup memory subsystem](#).

Procedure

1. When a job is started, the application runs under the job RES or root process group.
2. If an application creates a new process group, and its parent process ID (PPID) still belongs to the job, PIM can track this new process group as part of the job.
The only reliable way to not lose track of a process is to prevent it from using a new process group. Any process that daemonizes itself is lost when child processes are orphaned from the parent process group because it changes its process group right after it is detached.

Host not used by LSF

The **mbatchd** daemon allows the **sbatchd** daemon to run only on the hosts that are listed in the **Host** section of the `lsb.hosts` file. If you configure an unknown host in the following configurations, **mbatchd** logs an error message: **HostGroup** or **HostPartition** sections of the `lsb.hosts` file, or as a **HOSTS** definition for a queue in the `lsb.queue` file.

About this task

If you try to configure a host that is not listed in the **Host** section of the `lsb.hosts` file, the **mbatchd** daemon logs the following message.

```
mbatchd on host: LSB_CONFDIR/cluster1/configdir/file(line #): Host hostname is not used by lsbatch; ignored
```

If you start the **mbatchd** daemon on a host that is not known by the **mbatchd** daemon, **mbatchd** rejects the **sbatchd**. The **sbatchd** daemon logs the `This host is not used by lsbatch system.` message and exits.

Procedure

- Add the unknown host to the list of hosts in the **Host** section of the `lsb.hosts` file.
- Start the LSF daemons on the new host.
- Run the following commands to reconfigure the cluster:

```
lsadmin reconfig  
badmin reconfig
```

Unknown host type or model

A model or type **UNKNOWN** indicates that the host is down or the **LIM** on the host is down. You need to take immediate action to restart **LIM** on the **UNKNOWN** host.

Procedure

1. Start the host.
2. Run the **lshosts** command to see which host has the **UNKNOWN** host type or model.

```
lshosts
HOST_NAME  type      model    cpuf    ncpus   maxmem   maxswp   server  RESOURCES
hostA      UNKNOWN   Ultra2   20.2    2       256M     710M     Yes     ()
```

- Run the **bctrld start lim** command to start **LIM** on the host:

```
bctrld start lim hostA
Starting up LIM on <hostA> .... done
```

If EGO is enabled in the LSF cluster, you can run the following command instead:

```
egosh ego start lim hostA
Starting up LIM on <hostA> .... done
```

You can specify more than one host name to start **LIM** on multiple hosts. If you do not specify a host name, **LIM** is started on the host from which the command is submitted.

To start **LIM** remotely on UNIX or Linux, you must be root or listed in the `lsf.sudoers` file (or the `ego.sudoers` file if EGO is enabled in the LSF cluster). You must be able to run the **rsh** command across all hosts without entering a password.

- Wait a few seconds, then run the **lshosts** command again.

The **lshosts** command displays a specific model or type for the host or **DEFAULT**. If you see **DEFAULT**, it means that automatic detection of host type or model failed, and the host type that is configured in the `lsf.shared` file cannot be found. LSF works on the host, but a **DEFAULT** model might be inefficient because of incorrect CPU factors. A **DEFAULT** type might also cause binary incompatibility because a job from a **DEFAULT** host type can be migrated to another **DEFAULT** host type.

Default host type or model

If you see **DEFAULT** in **lim -t**, it means that automatic detection of host type or model failed, and the host type that is configured in the `lsf.shared` file cannot be found. LSF works on the host, but a **DEFAULT** model might be inefficient because of incorrect CPU factors. A **DEFAULT** type might also cause binary incompatibility because a job from a **DEFAULT** host type can be migrated to another **DEFAULT** host type.

Procedure

- Run the **lshosts** command to see which host has the **DEFAULT** host model or type.

```
lshosts
HOST_NAME  type      model    cpuf    ncpus   maxmem   maxswp   server  RESOURCES
hostA      DEFAULT   DEFAULT   1       2       256M     710M     Yes     ()
```

If **Model** or **Type** are displayed as **DEFAULT** when you use the **lshosts** command and automatic host model and type detection is enabled, you can leave it as is or change it.

If the host model is **DEFAULT**, LSF works correctly but the host has a CPU factor of 1, which might not make efficient use of the host model.

If the host type is **DEFAULT**, there might be binary incompatibility. For example, if one host is Linux and another is AIX, but both hosts are set to type **DEFAULT**, jobs that are running on the Linux host can be migrated to the AIX host, or jobs running the AIX host can be migrated to the Linux hosts, which might cause the job to fail.

- Run **lim -t** on the host whose type is **DEFAULT**:

```
lim -t
Host Type           : NTX64
Host Architecture   : EM64T_1596
Total NUMA Nodes    : 1
Total Processors    : 2
Total Cores         : 4
Total Threads       : 2
Matched Type        : NTX64
Matched Architecture : EM64T_3000
Matched Model       : Intel_EM64T
CPU Factor          : 60.0
```

Note: The value of **HostType** and **Host Architecture**.

3. Edit the `lsf.shared` file to configure the host type and host model for the host.
 - a. In the **HostType** section, enter a new host type. Use the host type name that is detected with the **lim -t** command.

```
Begin HostType
TYPENAME
DEFAULT
CRAYJ
NTX64
...
End HostType
```

- b. In the **HostModel** section, enter the new host model with architecture and CPU factor. Use the architecture that is detected with the **lim -t** command. Add the host model to the end of the host model list. The limit for host model entries is 127. Lines commented out with **#** are not counted in the 127 line limit.

```
Begin HostModel
MODELNAME    CPUFACTOR    ARCHITECTURE # keyword
Intel_EM64T    20                EM64T_1596
End HostModel
```

4. Save changes to the `lsf.shared` file.
5. Run the **lsadmin reconfig** command to reconfigure **LIM**.
6. Wait a few seconds, and run the **lim -t** command again to check the type and model of the host.

LSF error messages

The following error messages are logged by the LSF daemons, or displayed by the **lsadmin ckconfig** and **badmin ckconfig** commands.

General errors

The following messages can be generated by any LSF daemon:

- **can't open file: error**
The daemon might not open the named file for the reason that is given by *error*. This error is usually caused by incorrect file permissions or missing files. All directories in the path to the configuration files must have execute (**x**) permission for the LSF administrator, and the actual files must have read (**r**) permission.

Missing files might be caused by the following errors:
 - Incorrect path names in the `lsf.conf` file
 - Running LSF daemons on a host where the configuration files are not installed
 - Having a symbolic link that points to a file or directory that does not exist
- **file(line): malloc failed**
Memory allocation failed. Either the host does not have enough available memory or swap space, or there is an internal error in the daemon. Check the program load and available swap space on the host. If the swap space is full, you must add more swap space or run fewer (or smaller) programs on that host.
- **auth_user: getservbyname(ident/tcp) failed: error; ident must be registered in services**
The `LSF_AUTH=ident` parameter is defined in the `lsf.conf` file, but the `ident/tcp` service is not defined in the services database. Add `ident/tcp` to the services database, or remove the `LSF_AUTH=ident` parameter from the `lsf.conf` file and use the **setuid root** command on the LSF files that require authentication.
- **auth_user: operation(<host>/<port>) failed: error**
The `LSF_AUTH=ident` parameter is defined in the `lsf.conf` file, but the LSF daemon failed to contact the **identd** daemon on the host. Check that **identd** is defined in `inetd.conf` and the **identd** daemon is running on host.
- **auth_user: Authentication data format error (rbuf=<data>) from <host>/<port>**
auth_user: Authentication port mismatch (...) from <host>/<port>

The LSF_AUTH=ident parameter is defined in the lsf.conf file, but there is a protocol error between LSF and the **ident** daemon on *host*. Make sure that the **identd** daemon on the host is configured correctly.

- **userok: Request from bad port**
(<port_number>), denied

The LSF_AUTH=ident parameter is not defined, and the LSF daemon received a request that originates from a non-privileged port. The request is not serviced.

Set the LSF binary files to be owned by root with the **setuid** bit set, or define the LSF_AUTH=ident parameter and set up an **ident** server on all hosts in the cluster. If the files are on an NFS-mounted file system, make sure that the file system is not mounted with the **nosuid** flag.

- **userok: Forged username suspected from**
<host>/<port>:
<claimed_user>/<actual_user>

The service request claimed to come from user *claimed_user* but **ident** authentication returned that the user was *actual_user*. The request was not serviced.

- **userok: ruserok(<host>,<uid>) failed**

The LSF_USE_HOSTEQUIV parameter is defined in the lsf.conf file, but *host* is not set up as an equivalent host in /etc/host.equiv, and user *uid* is not set up in a .rhosts file.

- **init_AcceptSock: RES service(res) not registered,**
exiting
init_AcceptSock: res/tcp: unknown service,
exiting

initSock: LIM service not
registered.

initSock: Service lim/udp is unknown. Read LSF Guide
for help

get_ports: <serv> service not registered

The LSF services are not registered.

- **init_AcceptSock: Can't bind daemon socket to port <port>: error,**
exiting
init_ServSock: Could not bind socket to port <port>:
error

These error messages can occur if you try to start a second LSF daemon (for example, **RES** is already running, and you run **RES** again). If so, and you want to start the new daemon, kill the running daemon or use the **lsadmin** or **badmin** commands to shut down or restart the daemon.

Configuration errors

The following messages are caused by problems in the LSF configuration files. General errors are listed first, and then errors from specific files.

- **file(line): Section name expected after Begin; ignoring**
section
file(line): Invalid section name name; ignoring
section

The keyword **Begin** at the specified line is not followed by a section name, or is followed by an unrecognized section name.

- **file(line): section section: Premature EOF**

The end of file was reached before reading the End section line for the named section.

- **file(line): keyword line format error for section section; Ignore this**
section

The first line of the section must contain a list of keywords. This error is logged when the keyword line is incorrect or contains an unrecognized keyword.

- **file(line): values do not match keys for section section; Ignoring line**
The number of fields on a line in a configuration section does not match the number of keywords. This error can be caused by not putting () in a column to represent the default value.
- **file: HostModel section missing or invalid**
file: Resource
section missing or invalid

file: HostType section missing or invalid

The **HostModel**, **Resource**, or **HostType** section in the `lsf.shared` file is either missing or contains an unrecoverable error.
- **file(line): Name name reserved or previously defined. Ignoring index**
The name that is assigned to an external load index must not be the same as any built-in or previously defined resource or load index.
- **file(line): Duplicate cluster name name in section cluster. Ignoring current line**
A cluster name is defined twice in the same `lsf.shared` file. The second definition is ignored.
- **file(line): Bad cpuFactor for host model model. Ignoring line**
The CPU factor declared for the named host model in the `lsf.shared` file is not a valid number.
- **file(line): Too many host models, ignoring model name**
You can declare a maximum of 127 host models in the `lsf.shared` file.
- **file(line): Resource name name too long in section resource. Should be less than 40 characters. Ignoring line**
The maximum length of a resource name is 39 characters. Choose a shorter name for the resource.
- **file(line): Resource name name reserved or previously defined. Ignoring line.**
You attempted to define a resource name that is reserved by LSF or already defined in the `lsf.shared` file. Choose another name for the resource.
- **file(line): illegal character in resource name: name, section resource. Line ignored.**
Resource names must begin with a letter in the set `[a-zA-Z]`, followed by letters, digits, or underscores `[a-zA-Z0-9_]`.

LIM messages

The following messages are logged by the **LIM**:

- **findHostbyAddr/<proc>: Host <host>/<port> is unknown by <myhostname>**
function: Gethostbyaddr_(<host>/<port>) failed:
error

main: Request from unknown host <host>/<port>:
error

function: Received request from non-LSF host
<host>/<port>

The daemon does not recognize *host*. The request is not serviced. These messages can occur if *host* was added to the configuration files, but not all the daemons were reconfigured to read the new information. If the problem still occurs after reconfiguring all the daemons, check whether the host is a multi-addressed host.
- **rcvLoadVector: Sender (<host>/<port>) may have different config?**

MasterRegister: Sender (host) may have different config?

LIM detected inconsistent configuration information with the sending **LIM**. Run the **lsadmin reconfig** command so that all instance of **LIM** have the same configuration information.

Note any hosts that failed to be contacted.

- **rcvLoadVector:** Got load from client-only host <host>/<port>. Kill **LIM** on <host>/<port>
A **LIM** is running on a client host. Run the following command, or go to the client host and kill the **LIM** daemon.
bctrld stop lim host
- **saveIndx:** Unknown index name <name> from **ELIM**
LIM received an external load index name that is not defined in the `lsf.shared` file. If name is defined in `lsf.shared`, reconfigure the **LIM**. Otherwise, add name to the `lsf.shared` file and reconfigure all instances of **LIM**.
- **saveIndx:** **ELIM** over-riding value of index <name>
This warning message is logged when the **ELIM** sent a value for one of the built-in index names. **LIM** uses the value from **ELIM** in place of the value that is obtained from the kernel.
- **getusr:** Protocol error numIndx not read (cc=num): error
getusr:
Protocol error on index number (cc=num): error

Protocol error between **ELIM** and **LIM**.

RES messages

The following messages are logged by the **RES**:

- **doacceptconn:** getpwnam(<username>@<host>/<port>) failed:
error
doacceptconn: User <username> has uid <uid1> on client host <host>/<port>, uid <uid2> on RES host; assume bad user

authRequest:
username/uid <userName>/<uid>@<host>/<port> does not exist

authRequest: Submitter's name <cname>@<clhost> is different from name <lname> on this host

RES assumes that a user has the same user ID and user name on all the LSF hosts. These messages occur if this assumption is violated. If the user is allowed to use LSF for interactive remote execution, make sure the user's account has the same user ID and user name on all LSF hosts.

- **doacceptconn:** root remote execution permission denied
authRequest:
root job submission rejected

Root tried to run or submit a job but `LSF_ROOT_REX` is not defined in the `lsf.conf` file.

- **resControl:** operation permission denied, uid = <uid>
The user with user ID `uid` is not allowed to make **RES** control requests. Only the LSF administrator can make **RES** control requests. If the `LSF_ROOT_REX` parameter is defined in the `lsf.conf` file, can also make **RES** control requests.
- **resControl:** access(respath, X_OK): error
The **RES** received a restart request, but failed to find the file `respath` to re-execute itself. Ensure that `respath` contains the **RES** binary, and it has execution permission.

mbatchd and sbatchd messages

The following messages are logged by the **mbatchd** and **sbatchd** daemons:

- **renewJob:** Job <jobId>: rename(<from>,<to>) failed:
error
mbatchd failed in trying to resubmit a re-runnable job. Check that the file *from* exists and that the LSF administrator can rename the file. If *from* is in an AFS directory, check that the LSF administrator's token processing is properly setup.
- **logJobInfo_:** fopen(<logdir/info/jobfile>) failed:
error
logJobInfo_: write <logdir/info/jobfile> <data> failed:
error

logJobInfo_: seek <logdir/info/jobfile> failed:
error

logJobInfo_: write <logdir/info/jobfile> xdrpos <pos> failed:
error

logJobInfo_: write <logdir/info/jobfile> xdr buf len <len>
failed: error

logJobInfo_: close(<logdir/info/jobfile>) failed:
error

rmLogJobInfo: Job <jobId>: can't unlink(<logdir/info/jobfile>):
error

rmLogJobInfo_: Job <jobId>: can't stat(<logdir/info/jobfile>):
error

readLogJobInfo: Job <jobId> can't open(<logdir/info/jobfile>):
error

start_job: Job <jobId>: readLogJobInfo failed:
error

readLogJobInfo: Job <jobId>: can't read(<logdir/info/jobfile>)
size size: error

initLog: mkdir(<logdir/info>) failed:
error

<fname>: fopen(<logdir/file>) failed:
error

getElogLock: Can't open existing lock file <logdir/file>:
error

getElogLock: Error in opening lock file <logdir/file>:
error

releaseElogLock: unlink(<logdir/lockfile>) failed:
error

touchElogLock: Failed to open lock file <logdir/file>:
error

touchElogLock: close <logdir/file> failed:
error

mbatchd failed to create, remove, read, or write the log directory or a file in the log directory, for the reason that is given in *error*. Check that the LSF administrator has read, write, and execute permissions on the logdir directory.
- **replay_newjob:** File <logfile> at line <line>: Queue <queue> not found, saving
to queue <lost_and_found>
replay_switchjob: File <logfile> at line
<line>: Destination queue <queue> not found, switching to queue
<lost_and_found>

When the **mbatchd** daemon was reconfigured, jobs were found in *queue* but that queue is no longer in the configuration.
- **replay_startjob:** JobId <jobId>: exec host <host> not found, saving to host
<lost_and_found>

When the **mbatchd** daemon was reconfigured, the event log contained jobs that are dispatched to host, but that host is no longer configured to be used by LSF.

- **do_restartReq: Failed to get hData of host <host_name>/<host_addr>**
The **mbatchd** received a request from **sbatchd** on host *host_name*, but that host is not known to **mbatchd**. Either the configuration file has changed but **mbatchd** was not reconfigured to pick up the new configuration, or *host_name* is a client host but the **sbatchd** daemon is running on that host. Run the following **badmadmin reconfig** command to reconfigure the **mbatchd** daemon or kill the **sbatchd** daemon on *host_name*.

LSF command messages

LSF daemon (LIM) not responding ... still trying

During **LIM** restart, LSF commands might fail and display this error message. User programs that are linked to the **LIM** API also fail for the same reason. This message is displayed when **LIM** running on the management host list or server host list is restarted after configuration changes, such as adding new resources, or binary upgrade.

Use the **LSF_LIM_API_NTRIES** parameter in the **lsf.conf** file or as an environment variable to define how many times LSF commands retry to communicate with the **LIM** API while **LIM** is not available. The **LSF_LIM_API_NTRIES** parameter is ignored by LSF and EGO daemons and all EGO commands.

When the **LSB_API_VERBOSE=Y** parameter is set in the **lsf.conf** file, LSF batch commands display the not responding retry error message to stderr when **LIM** is not available.

When the **LSB_API_VERBOSE=N** parameter is set in the **lsf.conf** file, LSF batch commands do not display the retry error message when **LIM** is not available.

Batch command client messages

LSF displays error messages when a batch command cannot communicate with the **mbatchd** daemon. The following table provides a list of possible error reasons and the associated error message output.

Point of failure	Possible reason	Error message output
Establishing a connection with the mbatchd daemon	The mbatchd daemon is too busy to accept new connections. The connect() system call times out.	LSF is processing your request. Please wait...
	The mbatchd daemon is down or no process is listening at either the LSB_MBD_PORT or the LSB_QUERY_PORT	LSF is down. Please wait...
	The mbatchd daemon is down and the LSB_QUERY_PORT is busy	bhosts displays LSF is down. Please wait. ... bjobs displays Cannot connect to LSF. Please wait...
	Socket error on the client side	Cannot connect to LSF. Please wait...
	connect() system call fails	Cannot connect to LSF. Please wait...
	Internal library error	Cannot connect to LSF. Please wait...
Send/receive handshake message to/from the mbatchd daemon	The mbatchd daemon is busy. Client times out when LSF is waiting to receive a message from mbatchd .	LSF is processing your request. Please wait...
	Socket read()/write() fails	Cannot connect to LSF. Please wait...
	Internal library error	Cannot connect to LSF. Please wait...

EGO command messages

You cannot run the `egosh` command because the administrator has chosen not to enable EGO in `lsf.conf`: `LSF_ENABLE_EGO=N`.

If EGO is not enabled, the **egosh** command cannot find the `ego.conf` file or cannot contact the **vemkd** daemon (likely because it is not started).

Administer IBM Spectrum LSF

Learn how to manage your IBM® Spectrum LSF cluster, control daemons, change cluster configurations, and work with hosts and queues. Manage your LSF jobs and job scheduling policies. View job information and control jobs. Learn how to configure and allocate resources to your LSF jobs. Learn how to submit, monitor, and control high throughput and parallel workload in your LSF cluster. Learn about LSF error and event logging and how LSF handles job exceptions. Tune the performance and scalability of your LSF cluster.

- [IBM Spectrum LSF cluster management essentials](#)
Learn how to manage your LSF cluster, control daemons, change cluster configurations, and work with hosts, queues, and users.
- [Monitoring IBM Spectrum LSF cluster operations and health](#)
Learn how to monitor cluster performance, job resource usage, and other information about queues, jobs, and users.
- [Managing IBM Spectrum LSF job execution](#)
Learn how to manage your LSF jobs and job scheduling policies. View job information, control jobs, and manage job dependencies, job priorities, job arrays, interactive jobs, job pre-execution and post-execution, and job starters.
- [Configuring and sharing IBM Spectrum LSF job resources](#)
Learn how to configure and allocate resources to your LSF jobs. Share compute resources fairly among users and projects. Apply resource allocation limits to your jobs, manage host and user groups, reserve resources, and specify resource requirements for jobs.
- [GPU resources](#)
Learn how to configure and use GPU resources for your LSF jobs.
- [Configuring containers with LSF](#)
Configure and use LSF integrations for containers.
- [Administering IBM Spectrum LSF high throughput workload](#)
Learn how to submit, monitor, and control high throughput workload in your LSF cluster. Configure scheduling policies that enable efficient queuing, dispatch, and execution of short-running jobs.
- [Administering IBM Spectrum LSF parallel workload](#)
Learn how to submit, monitor, and control parallel workload in your LSF cluster. Configure scheduling policies that reserve resources to enable efficient execution of large parallel jobs.
- [IBM Spectrum LSF Security](#)
Learn how to optimize the security of your LSF cluster.
- [IBM Spectrum LSF advanced configuration](#)
Learn about LSF error and event logging and how LSF handles job exceptions. Configure advanced LSF features.
- [IBM Spectrum LSF performance tuning](#)
Tune the performance and scalability of your LSF cluster.
- [IBM Spectrum LSF energy aware scheduling](#)
Configure, manage, and use IBM Spectrum LSF energy-aware scheduling features for large-scale LSF installations, where the energy requirements for operating large systems are becoming a significant factor in the overall cost of these systems.
- [IBM Spectrum LSF multicluster capability](#)
Learn how to use and manage the IBM Spectrum LSF multicluster capability to share resources across your LSF clusters.
- [Submitting jobs using JSDL](#)

IBM Spectrum LSF cluster management essentials

Learn how to manage your LSF cluster, control daemons, change cluster configurations, and work with hosts, queues, and users.

- [Work with your cluster](#)
Learn about LSF directories and files, commands to see cluster information, control workload daemons, and how to configure your cluster.
- [Working with hosts](#)
Check the status of hosts in your cluster, view information about your hosts, control hosts. Add and remove hosts in your cluster.
- [Job directories and data](#)
Jobs use temporary directories for working files and temporary output. By default, IBM Spectrum LSF uses the default operating system temporary directory. Use the LSF current working directory (CWD) feature to create and manage the job CWD dynamically based on configuration parameters, and any dynamic patterns included in the path. Use the flexible job output directory to create and manage the job output directory dynamically based on configuration parameters.
- [Job notification](#)
By default, when a batch job completes or exits, LSF sends a job report by email to the submitting user account.

Work with your cluster

Learn about LSF directories and files, commands to see cluster information, control workload daemons, and how to configure your cluster.

- [Viewing LSF cluster information](#)
Use the **lsid**, **badmin**, **bparams**, and **lsclusters** commands to find information about the LSF cluster.
- [Control LSF system daemons](#)
Commands for starting, shutting down, restarting, and reconfiguring LSF system daemons.
- [Commands to reconfigure your cluster](#)
After you change parameters in LSF configuration files, you must run commands for LSF to reread the files to update the configuration.
- [Live reconfiguration](#)
Use live reconfiguration to make configuration changes in LSF active memory that takes effect immediately. Live reconfiguration requests use the **bconf** command, and generate updated configuration files in the directory set by the LSF_LIVE_CONFDIR parameter in the `lsf.conf` file.
- [Adding cluster administrators](#)
Add or change the list of administrators for your cluster.

Viewing LSF cluster information

Use the **lsid**, **badmin**, **bparams**, and **lsclusters** commands to find information about the LSF cluster.

Procedure

Cluster information includes the cluster management host, cluster name, cluster resource definitions, cluster administrator, and other details.

Table 1. LSF commands to view cluster information

View	Command
Version of LSF	lsid
Cluster name	lsid
Current management host	lsid
Cluster administrators	lsclusters

View	Command
Configuration parameters	bparams
LSF system runtime information	badmin showstatus

Viewing LSF version, cluster name, and current management host

Use the **lsid** command to display the version of LSF, the name of your cluster, and the current management host.

Procedure

The **lsid** command displays cluster version management host information.

```
lsid
LSF 10.1.0, Jan 5 2016
© Copyright IBM Corp. 1992, 2022.
US Government Users Restricted Rights - Use, duplication or disclosure restricted
  by GSA ADP Schedule Contract with IBM Corp.
My cluster name is lsf10
My master name is hosta.company.com
```

Viewing cluster administrators

Use the **lsclusters** command to find out who your cluster administrator is and see a summary of your cluster.

Procedure

The **lsclusters** command summarizes current cluster status:

```
lsclusters
CLUSTER_NAME  STATUS  MASTER_HOST  ADMIN  HOSTS  SERVERS
cluster1      ok      hostA        lsfadmin  6      6
```

If you are using the IBM® Spectrum LSF multicluster capability, you can see one line for each of the clusters that your local cluster is connected to in the output of the **lsclusters** command.

Viewing configuration parameters

Use the **bparams** command to display the generic configuration parameters of LSF.

Procedure

1. The **bparams** command shows default queues, job dispatch interval, job checking interval, and job acceptance interval.

```
bparams
Default Queues:  normal idle
MBD_SLEEP_TIME used for calculations: 20 seconds
Job Checking Interval: 15 seconds
Job Accepting Interval: 20 seconds
```

2. Use the **bparams -l** command to display the information in long format, which gives a brief description of each parameter and the name of the parameter as it appears in the lsb.params file.

```
bparams -l
System default queues for automatic queue selection:
  DEFAULT_QUEUE = normal idle
Amount of time in seconds used for calculating parameter values:
  MBD_SLEEP_TIME = 20 (seconds)
The interval for checking jobs by server batch daemon:
  SBD_SLEEP_TIME = 15 (seconds)
The interval for a host to accept two batch jobs subsequently:
  JOB_ACCEPT_INTERVAL = 1 (* MBD_SLEEP_TIME)
The idle time of a host for resuming pg suspended jobs:
  PG_SUSP_IT = 180 (seconds)
The amount of time during which finished jobs are kept in core:
  CLEAN_PERIOD = 3600 (seconds)
```

```

The maximum number of finished jobs that are logged in current event file:
MAX_JOB_NUM = 2000
The maximum number of retries for reaching a server batch daemon:
MAX_SBD_FAIL = 3
The number of hours of resource consumption history:
HIST_HOURS = 5
The default project assigned to jobs.
DEFAULT_PROJECT = default
Sync up host status with management host LIM is enabled:
LSB_SYNC_HOST_STAT_LIM = Y
MBD child query processes will only run on the following CPUs:
MBD_QUERY_CPUS=1 2 3

```

3. Use the **bparams -a** command to display all configuration parameters and their values in the `lsb.params` file. For example,

```

bparams -a
    MBD_SLEEP_TIME = 20
    SBD_SLEEP_TIME = 15
    JOB_ACCEPT_INTERVAL = 1
    SUB_TRY_INTERVAL = 60
    LSB_SYNC_HOST_STAT_LIM = N
    MAX_JOBINFO_QUERY_PERIOD = 2147483647
    PEND_REASON_UPDATE_INTERVAL = 30
    ...

```

Viewing daemon parameter configuration

Use the **badmin showconf mbd** command and the **lsadmin showconf** command to show current cluster configuration settings.

Before you begin

Log on to a server host.

Procedure

1. Display all configuration settings for running LSF daemons.
 - Use the **lsadmin showconf** command to display all configured parameters and their values in the `lsf.conf` or `ego.conf` file for LIM.
 - Use the **badmin showconf mbd** command or the **badmin showconf sbd** command to display all configured parameters and their values in the `lsf.conf` or `ego.conf` file for the **mbatchd** and **sbatchd** daemons.

In IBM Spectrum LSF multicluster capability, the parameters apply to the local cluster only.
2. Display **mbatchd** and root **sbatchd** daemon configuration.
 - Use the **badmin showconf mbd** command to display the parameters that are configured in the `lsf.conf` or `ego.conf` file that apply to the **mbatchd** daemon.
 - Use the **badmin showconf sbd** command to display the parameters that are configured in the `lsf.conf` or `ego.conf` file that apply to the root **sbatchd** daemon.

Example

- Run the **badmin showconf mbd** command to show the **mbatchd** daemon configuration:

```

badmin showconf mbd
MBD configuration at Fri Jun 8 10:27:52 CST 2011
    LSB_SHAREDIR=/dev/lsf/user1/0604/work
    LSF_CONFDIR=/dev/lsf/user1/0604/conf
    LSF_LOG_MASK=LOG_WARNING
    LSF_ENVDIR=/dev/lsf/user1/0604/conf
    LSF_EGO_DAEMON_CONTROL=N
    ...

```

- Run the **badmin showconf sbd host_name** command to show the **sbatchd** daemon configuration on a specific host:

```

badmin showconf sbd hosta
SBD configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2011

```



```
LSB_SHAREDIR=/dev/lsf/user1/0604/work
LSF_CONFDIR=/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/dev/lsf/user1/0604/conf
LSF_DAEMON_CONTROL=N
...
```

- Run the **badadmin showconf sbd all** command to show the **sbatchd** daemon configuration for all hosts:

```
badadmin showconf sbd all
SBD configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/dev/lsf/user1/0604/work
LSF_CONFDIR=/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
...
SBD configuration for host <hostb> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/dev/lsf/user1/0604/work
LSF_CONFDIR=/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
...
```

- Run the **lsadmin showconf lim** command to show the **lim** daemon configuration:

```
lsadmin showconf lim
LIM configuration at Fri Jun 8 10:27:52 CST 2010
LSB_SHAREDIR=/dev/lsf/user1/0604/work
LSF_CONFDIR=/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
...
```

- Run the **lsadmin showconf lim host_name** command to show the **lim** daemon configuration for a specific host:

```
lsadmin showconf lim hosta
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/dev/lsf/user1/0604/work
LSF_CONFDIR=/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
...
```

- Run the **lsadmin showconf lim all** command to show the **lim** daemon configuration for all hosts:

```
lsadmin showconf lim all
LIM configuration for host <hosta> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/dev/lsf/user1/0604/work
LSF_CONFDIR=/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
...
LIM configuration for host <hostb> at Fri Jun 8 10:27:52 CST 2011
LSB_SHAREDIR=/dev/lsf/user1/0604/work
LSF_CONFDIR=/dev/lsf/user1/0604/conf
LSF_LOG_MASK=LOG_WARNING
LSF_ENVDIR=/dev/lsf/user1/0604/conf
LSF_EGO_DAEMON_CONTROL=N
...
```

Viewing runtime cluster summary information

Use the **badadmin showstatus** command to display a summary of the current LSF runtime information.

Procedure

The **badmin showstatus** command displays information about hosts, jobs, users, user groups, and the **mbatchd** daemon startup and reconfiguration:

```
% badmin showstatus
```

LSF runtime mbatchd information

Available local hosts (current/peak):

```
Clients:      0/0
Servers:      8/8
CPUs:         14/14
Cores:        50/50
Slots:        50/50
```

```
Number of servers: 8
Ok:                8
Closed:            0
Unreachable:       0
Unavailable:       0
```

```
Number of jobs:    7
Running:           0
Suspended:         0
Pending:           0
Finished:          7
```

```
Number of users:   3
Number of user groups: 1
Number of active users: 0
```

```
Latest mbatchd start: Thu Nov 22 21:17:01 2012
Active mbatchd PID:   26283
```

```
Latest mbatchd reconfig: Thu Nov 22 21:18:06 2012
```

mbatchd restart information

```
New mbatchd started: Thu Nov 22 21:18:21 2012
New mbatchd PID:     27474
```

Control LSF system daemons

Commands for starting, shutting down, restarting, and reconfiguring LSF system daemons.

Permissions required

To control all daemons in the cluster, the following permissions are required:

- You must be logged on as **root** or as a user listed in the `/etc/lsf.sudoers` file.
- You must be able to run the **rsh** or **ssh** commands across all LSF hosts without having to enter a password. See your operating system documentation for information about configuring these commands. The shell command that is specified by the `LSF_RSH` parameter in the `lsf.conf` file is used before attempting to use the **rsh** command.

LSF system daemon commands

The following table lists an overview of commands that you use to control LSF daemons.

Table 1. Commands to control LSF daemons

Daemon	Action	Command (Note that once you use systemctl commands, continue to use them instead of other control commands)	Permissions
--------	--------	--	-------------

Daemon	Action	Command (Note that once you use systemctl commands, continue to use them instead of other control commands)	Permissions
All daemons in the cluster	Start	<ul style="list-style-type: none"> lsfstartup (Available starting in Fix Pack 14) systemctl start lsfd 	Must be root or a user who is listed in the lsf.sudoers file for all these commands
	Shut down	<ul style="list-style-type: none"> lsfshutdown (Available starting in Fix Pack 14) systemctl stop lsfd 	
	Restart	(Available starting in Fix Pack 14) systemctl restart lsfd	
sbatchd	Start	<ul style="list-style-type: none"> bctrld start sbd [<i>host_name</i> ... all] (Available starting in Fix Pack 14) systemctl start lsfd-sbd 	Must be root or a user who is listed in the lsf.sudoers file for the startup command
	Shut down	<ul style="list-style-type: none"> bctrld stop sbd [<i>host_name</i> ... all] (Available starting in Fix Pack 14) systemctl stop lsfd-sbd 	Must be root or the LSF administrator for other commands
	Restart	<ul style="list-style-type: none"> bctrld restart sbd [<i>host_name</i> ... all] (Available starting in Fix Pack 14) systemctl restart lsfd-sbd 	
mbatchd	Shut down	<ul style="list-style-type: none"> bctrld stop sbd badmin mbdrestart 	Must be root or the LSF administrator for these commands
	Restart	badmin mbdrestart	
	Reconfigure	badmin reconfig	
RES	Start	<ul style="list-style-type: none"> bctrld start res [<i>host_name</i> ... all] (Available starting in Fix Pack 14) systemctl start lsfd-res 	Must be root or a user who is listed in the lsf.sudoers file for the startup command
	Shut down	<ul style="list-style-type: none"> bctrld stop res [<i>host_name</i> ... all] (Available starting in Fix Pack 14) systemctl stop lsfd-res 	Must be the LSF administrator for other commands
	Restart	<ul style="list-style-type: none"> bctrld restart res [<i>host_name</i> ... all] 	
LIM	Start	<ul style="list-style-type: none"> bctrld start lim [<i>host_name</i> ... all] (Available starting in Fix Pack 14) systemctl start lsfd-lim 	Must be root or a user who is listed in the lsf.sudoers file for the startup command
	Shut down	<ul style="list-style-type: none"> bctrld stop lim [<i>host_name</i> ... all] (Available starting in Fix Pack 14) systemctl stop lsfd-lim 	Must be the LSF administrator for other commands
	Restart	<ul style="list-style-type: none"> bctrld restart lim [<i>host_name</i> ... all] (Available starting in Fix Pack 14) systemctl restart lsfd-lim 	

Daemon	Action	Command (Note that once you use systemctl commands, continue to use them instead of other control commands)	Permissions
	Restart all hosts in the cluster	lsadmin reconfig	

sbatchd daemon

Restarting the **sbatchd** daemon on a host does not affect jobs that are running on that host.

If the **sbatchd** daemon is shut down, the host is not available to run new jobs. Any existing jobs that are running on that host continue, but the results are not sent to the user until the **sbatchd** daemon is restarted.

LIM and RES daemons

Jobs running on the host are not affected by restarting the daemons.

If a daemon is not responding to network connections, the **lsadmin** command displays an error message with the host name. In this case, you must stop and restart the daemon manually.

If the load information manager (LIM) and the other daemons on the current management host are shut down, another host automatically takes over as the management host.

If resource execution server (RES) is shut down while remote interactive tasks are running on the host, the running tasks continue but no new tasks are accepted.

LSF daemons or binary files protected from operating system out-of-memory (OS OOM) killer

The following LSF daemons are protected from being stopped on systems that support out-of-memory (OOM) killer:

- root RES
- root LIM
- root **sbatchd**
- **pim**
- **melim**
- **mbatchd**
- **rla**
- **mbschd**
- **krbrenwd**
- **elim**
- **lim -2** (root)
- **mbatchd -2** (root)

For the preceding daemons, the `oom_adj` parameter is automatically set to -17 or the `oom_score_adj` parameter is set to -1000 when the daemons are started or restarted. This feature ensures that LSF daemons survive the OOM killer but not user jobs.

When the `oom_adj` or `oom_score_adj` parameters are set, the log messages are set to DEBUG level: Set `oom_adj` to -17. and Set `oom_score_adj` to -1000.

The root RES, root LIM, root **sbatchd**, **pim**, **melim**, and **mbatchd** daemons protect themselves actively and log messages.

All logs must set the LSF_LOG_MASK as LOG_DEBUG parameters.

In addition, the following parameters must be set:

- RES must be configured as LSF_DEBUG_RES="LC_TRACE"
- LIM must be configured as LSF_DEBUG_LIM="LC_TRACE"

When the enterprise grid orchestrator (EGO) is enabled, the EGO_LOG_MASK=LOG_DEBUG parameter must be set in the ego.conf file

- The **sbatchd** daemon must be configured as LSB_DEBUG_SBD="LC_TRACE"
- The **pim** daemon must be configured as LSF_DEBUG_PIM="LC_TRACE"
- The **mbatchd** daemon must be configured as LSB_DEBUG_MBD="LC_TRACE"
- [Controlling mbatchd](#)
Use the **badmin reconfig**, **badmin mbdrestart**, **badmin mbdrestart -C**, and **bctrl stop sbd** commands to control the **mbatchd** daemon.
- [LSF daemon startup control](#)
Use the LSF daemon startup control feature to specify a list of user accounts other than **root** that can start LSF daemons on UNIX hosts.

Related reference

- [lsf.sudoers](#)

Controlling mbatchd

Use the **badmin reconfig**, **badmin mbdrestart**, **badmin mbdrestart -C**, and **bctrl stop sbd** commands to control the **mbatchd** daemon.

Procedure

You use the **badmin** command to control **mbatchd**.

Reconfiguring mbatchd

About this task

If you add a host to a host group, a host to a queue, or change resource configuration in the Hosts section of the `lsf.cluster.cluster_name` file, the change is not recognized by jobs that were submitted before you reconfigured.

If you want the new host to be recognized, you must restart the **mbatchd** daemon (or add the host that uses the **bconf** command if you are using live reconfiguration).

Procedure

Run the **badmin reconfig** command.

Results

When you reconfigure the cluster, **mbatchd** does not restart. Only configuration files are reloaded.

Restarting mbatchd

Procedure

Run the **badmin mbdrestart** command.

LSF checks configuration files for errors and prints the results to stderr. If no errors are found, LSF runs the following tasks:

- Reload configuration files
- Restart the **mbatchd** daemon
- Reread events in the `lsb.events` file and replay the events to recover the running state of the last instance of the **mbatchd** daemon.

Results

Tip: Whenever LSF restarts the **mbatchd** daemon, **mbatchd** is not available for service requests. In large clusters with many events in the `lsb.events` file, restarting the **mbatchd** daemon can take some time. To avoid replaying events in the `lsb.events` file, use the **badadmin reconfig** command.

Logging a comment when you restart mbatchd

Procedure

1. Use the **-C** option of the **badadmin mbdrestart** command to log an administrator comment in the `lsb.events` file. For example, to add "Configuration change" as a comment to the `lsb.events` file, run the following command:

```
badadmin mbdrestart -C "Configuration change"
```

The comment text `Configuration change` is recorded in the `lsb.events` file.

2. Run the **badadmin hist** or **badadmin mbdhist** commands to display administrator comments for the **mbatchd** daemon restart.

Shutting down mbatchd

Procedure

1. Run the **bctrld stop sbd** command to shut down the **sbatchd** daemon on the management host. For example, to shut down the **sbatchd** daemon on the `hostD` host, run the following command:

```
bctrld stop sbd hostD
```

2. Run the **badadmin mbdrestart** command:

```
badadmin mbdrestart
```

Running this command causes the **mbatchd** and **mbatchd** daemons to exit. The **mbatchd** daemon cannot be restarted because the **sbatchd** daemon is shut down. All LSF services are temporarily not available, but existing jobs are not affected. When the **sbatchd** daemon later starts up the **mbatchd** daemon, the previous status of the **mbatchd** daemon is restored from the event log file and job scheduling continues.

LSF daemon startup control

Use the LSF daemon startup control feature to specify a list of user accounts other than **root** that can start LSF daemons on UNIX hosts.

This feature also enables UNIX and Windows users to bypass the additional login that is required to start the **res** and **sbatchd** daemons when the enterprise grid orchestrator service controller (**egosc**) is configured to control LSF daemons. Bypassing the enterprise grid orchestrator (EGO) administrator login enables the use of scripts to automate system startup.

For more information about EGO, see [Manage LSF on EGO](#).

- [LSF daemon startup control overview](#)
The LSF daemon startup control feature specifies a list of user accounts that are allowed to start LSF daemons.
- [Configuration to enable LSF daemon startup control](#)
Edit the `lsf.sudoers` file to enable LSF daemon startup control.
- [LSF daemon startup control behavior](#)
- [Configuration to modify LSF daemon startup control](#)
Not applicable. This feature has no parameters to modify behavior.
- [LSF daemon startup control commands](#)
LSF daemon startup control commands include **bhosts**, **lsload**, **bctrld start sbd**, **bctrld start lim**, **bctrld start res**, and **badadmin showconf**

LSF daemon startup control overview

The LSF daemon startup control feature specifies a list of user accounts that are allowed to start LSF daemons.

Startup of LSF daemons by users other than root (UNIX only)

On UNIX hosts, by default only root can manually start LSF daemons. To manually start LSF daemons, a user runs the commands **lsadmin** and **badmin**. The LSF daemon startup control feature specifies a list of user accounts that are allowed to run the commands **lsadmin** and **badmin** to start LSF daemons. The list is defined in the `lsf.sudoers` file.

On Windows hosts, the services admin group identifies the user accounts that can start and shut down LSF daemons.

Figure 1. Default behavior (feature not enabled)

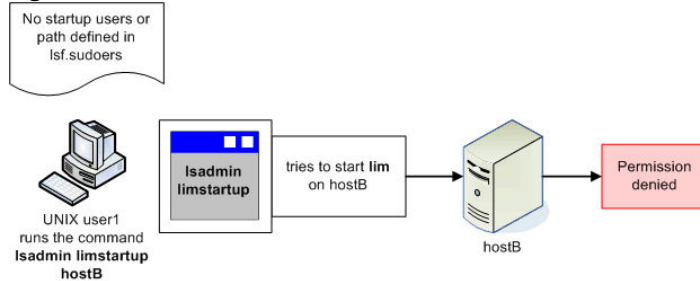
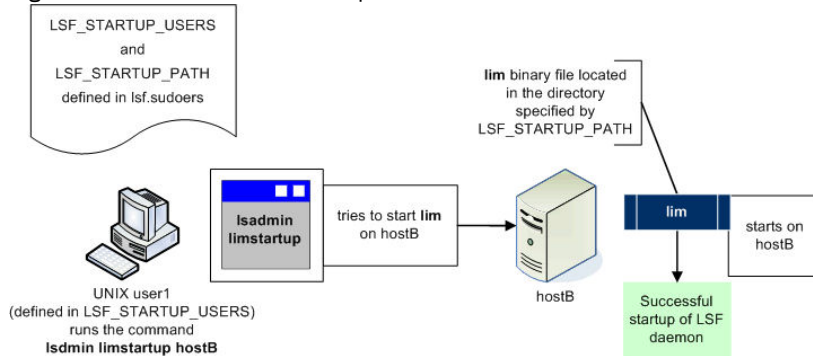


Figure 2. With LSF daemon startup control enabled



EGO administrator login bypass

If the EGO service controller (**egosc**) is configured to control LSF daemons, EGO automatically restarts the **res** and **sbatchd** daemons unless a user has manually shut them down. When manually starting a **res** or **sbatchd** daemon that EGO did not start, the user who starts **lsadmin** or **badmin** is prompted to enter EGO administrator credentials. You can configure LSF to bypass this step by specifying the EGO administrator credentials in the `lsf.sudoers` file.

In the following illustrations, an authorized user is either a UNIX user who is listed in the `LSF_STARTUP_USERS` parameter or a Windows user with membership in the services admin group.

Figure 3. EGO administrator login bypass not enabled

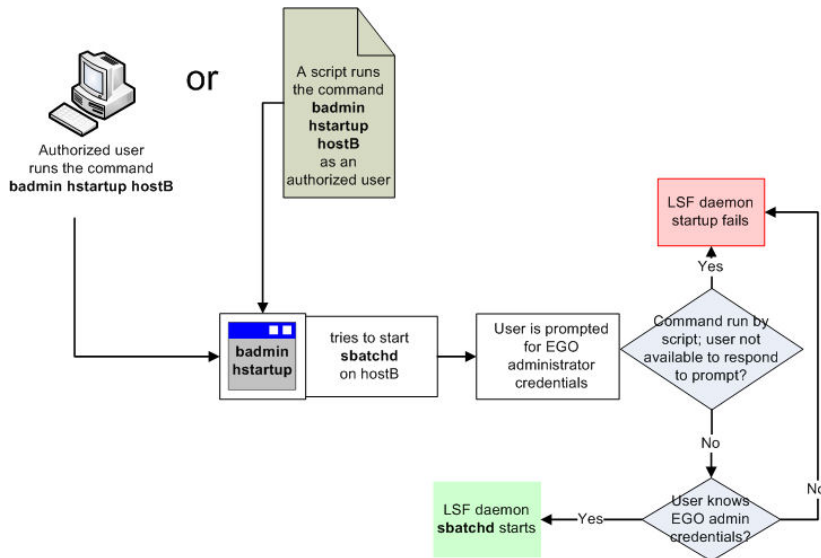
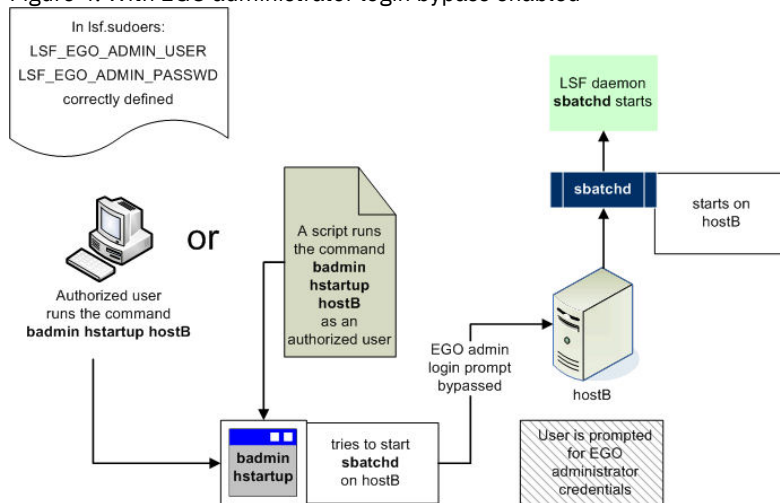


Figure 4. With EGO administrator login bypass enabled



Scope

Table 1. Scope of LSF daemon startup control

Applicability	Details
Operating system	<ul style="list-style-type: none"> For UNIX hosts only within a UNIX or mixed UNIX/Windows cluster, you can configure startup of LSF daemons by users other than root. For UNIX and Windows hosts, you can configure EGO administrator login bypass.
Dependencies	<ul style="list-style-type: none"> For startup of LSF daemons by users other than root: <ul style="list-style-type: none"> You must define both a list of users <i>and</i> the absolute path of the directory that contains the LSF daemon binary files. The non-root user must be a cluster administrator that is specified by the LSF_STARTUP_USERS parameter in the lsfsudoers file. The LSF_STARTUP_PATH parameter specifies the directory that contains the LSF daemon binary files. The commands lsadmin and badadmin must be installed as setuid root. Since this allows the LSF commands to run with root privileges, do not enable the setuid bit if you do not want these LSF commands to run with root privileges. For EGO administrator login bypass, the default Admin EGO cluster administrator account must be defined.

Applicability	Details
Limitations	<ul style="list-style-type: none"> Startup of LSF daemons by users other than root applies only to the following lsadmin and badmin subcommands: <ul style="list-style-type: none"> bctrld start sbd bctrld start lim bctrld start res

Configuration to enable LSF daemon startup control

Edit the `lsf.sudoers` file to enable LSF daemon startup control.

Startup of LSF daemons by users other than root (UNIX only)

The LSF daemon startup control feature is enabled for UNIX hosts by defining the `LSF_STARTUP_USERS` and `LSF_STARTUP_PATH` parameters in the `lsf.sudoers` file. Permissions for the `lsf.sudoers` file must be set to 600. For Windows hosts, this feature is already enabled at installation when the services admin group is defined.

Note: To use the `lsf.sudoers` file, you must enable the `setuid` bit for the LSF administration commands. Run the **hostsetup --setuid** command option on the LSF management and candidate hosts. Since this allows LSF administration commands to run with root privileges, do not enable the `setuid` bit if you do not want these LSF commands to run with root privileges. The **hostsetup --setuid** command enables the `setuid` bit for the following LSF executable files: **badmin**, **lsadmin**, **egosh**, **utmpreg**, **swtbl_api**, **ntbl_api**, **lstbl_nid**, and **swtbl_poe**.

Table 1. Configuration parameters

Configuration file	Parameter and syntax	Default behavior
lsf.sudoers	<code>LSF_STARTUP_USERS=all_admins</code>	<ul style="list-style-type: none"> Enables LSF daemon startup by users other than root when the <code>LSF_STARTUP_PATH</code> parameter is also defined. Allows all UNIX users who are defined as LSF administrators in the <code>lsf.cluster.cluster_name</code> file to start LSF daemons as root by running the lsadmin and badmin commands. CAUTION: This configuration introduces the security risk of a non-root LSF administrator who can add to the list of administrators in the <code>lsf.cluster.cluster_name</code> file. Not required for Windows hosts because all users with membership in the services admin group can start LSF daemons.
	<code>LSF_STARTUP_USERS="user_name1 user_name2 ..."</code> <code>LSF_STARTUP_USERS=user_name</code>	<ul style="list-style-type: none"> Enables LSF daemon startup by users other than root when the <code>LSF_STARTUP_PATH</code> parameter is also defined. Allows the specified user accounts to start LSF daemons as root by running the lsadmin and badmin commands. Specify only cluster administrator accounts; if you add a non-administrative user, the user can start, but not shut down, LSF daemons. Separate multiple user names with a space. For a single user, do not use quotation marks.

Configuration file	Parameter and syntax	Default behavior
	LSF_STARTUP_PATH=path	<ul style="list-style-type: none"> Enables LSF daemon startup by users other than root when the LSF_STARTUP_USERS parameter is also defined. Specifies the directory that contains the LSF daemon binary files. LSF daemons are installed in the path that is specified by the LSF_SERVERDIR parameter in the cshrc.lsf, profile.lsf, or lsf.conf files. <p>Important: For security reasons, move the LSF daemon binary files to a directory other than LSF_SERVERDIR or LSF_BINDIR. The user accounts specified by the LSF_STARTUP_USERS parameter can start any binary in the LSF_STARTUP_PATH directory.</p>

EGO administrator login bypass

For both UNIX and Windows hosts, you can bypass the EGO administrator login for the **res** and **sbatchd** daemons by defining the LSF_EGO_ADMIN_USER and LSF_EGO_ADMIN_PASSWORD parameters in the lsf.sudoers file.

Table 2. Configuration parameters

Configuration file	Parameter and syntax	Default behavior
lsf.sudoers	LSF_EGO_ADMIN_USER=Admin	<ul style="list-style-type: none"> Enables a user or script to bypass the EGO administrator login prompt when the LSF_EGO_ADMIN_PASSWORD parameter is also defined. Applies only to startup of res or sbatchd. Specify the Admin EGO cluster administrator account.
	LSF_EGO_ADMIN_PASSWORD=password	<ul style="list-style-type: none"> Enables a user or script to bypass the EGO administrator login prompt when the LSF_EGO_ADMIN_USER parameter is also defined. Applies only to startup of res or sbatchd. Specify the password for the Admin EGO cluster administrator account.

LSF daemon startup control behavior

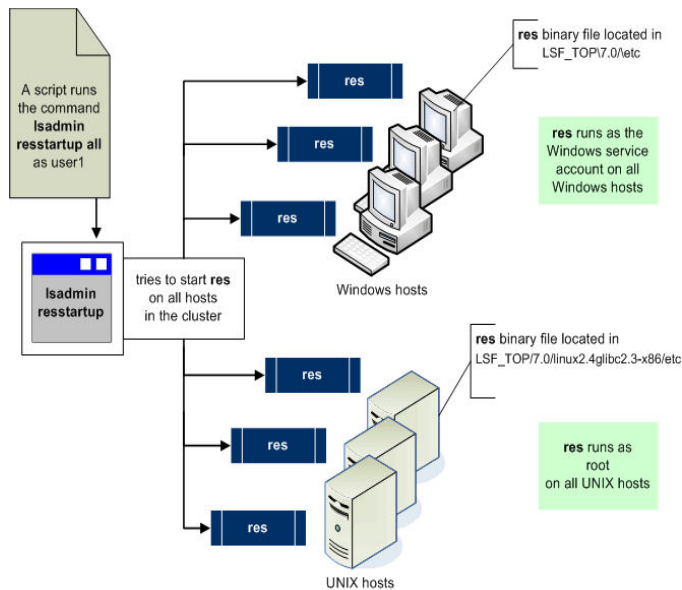
This example illustrates how LSF daemon startup control works when configured for UNIX hosts in a cluster with the following characteristics:

- The cluster contains both UNIX and Windows hosts
- The UNIX account user1 is mapped to the Windows account BUSINESS\user1 by enabling the UNIX/Windows user account mapping feature
- The account BUSINESS\user1 is a member of the services admin group
- In the lsf.sudoers file:

```
LSF_STARTUP_USERS="user1 user2 user3"
LSF_STARTUP_PATH=LSF_TOP/10.1.0/linux2.4-glibc2.3-x86/etc
LSF_EGO_ADMIN_USER=Admin
LSF_EGO_ADMIN_PASSWORD=Admin
```

Note: Change the Admin user password immediately after installation by using the command **egosh user modify**.

Figure 1. Example of LSF daemon startup control



Configuration to modify LSF daemon startup control

Not applicable. This feature has no parameters to modify behavior.

LSF daemon startup control commands

LSF daemon startup control commands include **bhosts**, **lsload**, **bctrlld start sbd**, **bctrlld start lim**, **bctrlld start res**, and **badmin showconf**

Commands for submission

Command	Description
N/A	<ul style="list-style-type: none"> This feature does not directly relate to job submission.

Commands to monitor

Command	Description
bhosts	<ul style="list-style-type: none"> Displays the host status of all hosts, specific hosts, or specific host groups.
lsload	<ul style="list-style-type: none"> Displays host status and load information.

Commands to control

Command	Description
bctrlld start sbd	<ul style="list-style-type: none"> Starts the sbatchd daemon on specific hosts or all hosts. Only root and users who are listed in the <code>LSF_STARTUP_USERS</code> parameter can successfully run this command.

Command	Description
bctrl start lim	<ul style="list-style-type: none"> Starts the lim daemon on specific hosts or all hosts in the cluster. Only root and users who are listed in the LSF_STARTUP_USERS parameter can successfully run this command.
bctrl start res	<ul style="list-style-type: none"> Starts the res daemon on specific hosts or all hosts in the cluster. Only root and users who are listed in the LSF_STARTUP_USERS parameter can successfully run this command.

Commands to display configuration

Command	Description
badmin showconf	<ul style="list-style-type: none"> Displays all configured parameters and their values set in the lsf.conf or ego.conf configuration files that affect the mbatchd and sbatchd daemons. Use a text editor to view other parameters in the lsf.conf or ego.conf configuration files. In an environment that uses LSF multicluster capability, displays the parameters of daemons on the local cluster.

Use a text editor to view the lsf.sudoers configuration file.

Commands to reconfigure your cluster

After you change parameters in LSF configuration files, you must run commands for LSF to reread the files to update the configuration.

Use the following commands to reconfigure a cluster:

- **lsadmin reconfig** to reconfigure the **lim** daemon
- **badmin reconfig** to reconfigure the **mbatchd** daemon without restarting
- **badmin mbdrestart** to restart the **mbatchd** daemon
- **bctrl restart sbd** to restart the **sbatchd** daemon

Note: After you change configuration, most LSF parameters require only reconfiguration (**lsadmin reconfig** or **badmin reconfig**). Several LSF parameters require restart (**badmin mbdrestart**). Which parameters require restart are indicated in the parameter description in the [configuration reference](#).

For most LSF parameters, the reconfiguration commands that you use depend on which files you change in LSF. The following table is a quick reference.

Table 1. Cluster reconfiguration commands

File changed	Command	Result
hosts	badmin reconfig	Reloads configuration files
lsb.applications	badmin reconfig	Reloads configuration files Pending jobs use new application profile definition. Running jobs are not affected.
lsb.hosts	badmin reconfig	Reloads configuration files
lsb.modules	badmin reconfig	Reloads configuration files
lsb.nqsmaps	badmin reconfig	Reloads configuration files
lsb.params	badmin reconfig	Reloads configuration files
lsb.queuees	badmin reconfig	Reloads configuration files
lsb.resources	badmin reconfig	Reloads configuration files
lsb.serviceclasses	badmin reconfig	Reloads configuration files
lsb.users	badmin reconfig	reloads configuration files

File changed	Command	Result
lsf.cluster.cluster_name	lsadmin reconfig AND badmin mbdrestart	restarts the lim daemon, reloads configuration files, and restarts the mbatchd daemon
lsf.conf	lsadmin reconfig AND badmin mbdrestart	Restarts the lim daemon, reloads configuration files, and restarts the mbatchd daemon
lsf.licensescheduler	bladmin reconfig lsadmin reconfig badmin mbdrestart	Restarts the bld daemon, restarts the lim daemon, reloads configuration files, and restarts the mbatchd daemon
lsf.shared	lsadmin reconfig AND badmin mbdrestart	Restarts the lim daemon, reloads configuration files, and restarts the mbatchd daemon
lsf.sudoers	badmin reconfig	Reloads configuration files

- [Reconfiguring the cluster with the lsadmin and badmin commands](#)
After you change a configuration file, use the **lsadmin reconfig** and **badmin reconfig** commands to reconfigure your cluster.
- [Reconfiguring the cluster by restarting the mbatchd daemon](#)
Use the **badmin mbdrestart** command to restart the **mbatchd** daemon on your cluster.
- [Viewing configuration errors](#)
Use the **lsadmin ckconfig -v** and **badmin ckconfig -v** commands to view configuration errors.

Reconfiguring the cluster with the lsadmin and badmin commands

After you change a configuration file, use the **lsadmin reconfig** and **badmin reconfig** commands to reconfigure your cluster.

About this task

To make a configuration change take effect, use this method to reconfigure the cluster.

Procedure

1. Log on to the host as `root` or the LSF administrator.
2. Run **lsadmin reconfig** to restart LIM:
`lsadmin reconfig`

The **lsadmin reconfig** command checks for configuration errors.

If no errors are found, you are prompted to either restart the **lim** daemon on management host candidates only, or to confirm that you want to restart the **lim** daemon on all hosts. If unrecoverable errors are found, reconfiguration is canceled.

3. Run the **badmin reconfig** command to reconfigure the **mbatchd** daemon:
`badmin reconfig`

The **badmin reconfig** command checks for configuration errors.

If unrecoverable errors are found, reconfiguration is canceled.

Reconfiguring the cluster by restarting the mbatchd daemon

Use the **badmin mbdrestart** command to restart the **mbatchd** daemon on your cluster.

About this task

To replay and recover the running state of the cluster, use this method to reconfigure the cluster.

Procedure

Run the **badmin mbdrestart** command to restart the **mbatchd** daemon:

```
badmin mbdrestart
```

The **badmin mbdrestart** command checks for configuration errors.

If no unrecoverable errors are found, you are asked to confirm the **mbatchd** daemon restart. If unrecoverable errors are found, the command exits and takes no action.

Tip: If the `lsb.events` file is large, or many jobs are running, restarting the **mbatchd** daemon can take some time. In addition, the **mbatchd** daemon is not available to service requests while it is restarted.

Viewing configuration errors

Use the **lsadmin ckconfig -v** and **badmin ckconfig -v** commands to view configuration errors.

Procedure

1. Run the **lsadmin ckconfig -v** command.
2. Run the **badmin ckconfig -v** command.

Results

These commands report all errors to your console.

Live reconfiguration

Use live reconfiguration to make configuration changes in LSF active memory that takes effect immediately. Live reconfiguration requests use the **bconf** command, and generate updated configuration files in the directory set by the `LSF_LIVE_CONFDIR` parameter in the `lsf.conf` file.

By default, the `LSF_LIVE_CONFDIR` parameter is set to `$LSB_SHAREDIR/cluster_name/live_confdir`. This directory is created automatically during LSF installation but remains empty until live reconfiguration requests write working configuration files into it later.

Live configuration changes that are made by the **bconf** command are recorded in the history file `liveconf.hist` located in the `$LSB_SHAREDIR/cluster_name/logdir` directory. Use the **bconf hist** command to query your changes. Not all configuration changes are supported by the **bconf** command and substantial configuration changes that are made by the **bconf** command might affect system performance for a few seconds.

When files exist in the directory set by the `LSF_LIVE_CONFDIR` parameter, all LSF restart and reconfiguration commands read the files in this directory instead of configuration files in configuration directory that are specified by the `LSF_CONFDIR` parameter. Merge the configuration files that are generated by **bconf** into `LSF_CONFDIR` regularly to avoid confusion. Alternatively, if you want the **bconf** command changes to overwrite original configuration files directly, set the `LSF_LIVE_CONFDIR` parameter to the same directory as the `LSF_CONFDIR` parameter.

For more information about the **bconf** command syntax and a complete list of configuration changes that are supported by live reconfiguration, see the **bconf** command man page or **bconf** in the *IBM® Spectrum LSF Command Reference*.

- [bconf command authentication](#)
Regular users can run the **bconf hist** command queries. Only cluster administrators and root can run all **bconf** commands.
- [Enabling live reconfiguration](#)
Enable live reconfiguration by defining the LSF_LIVE_CONFDIR parameter in the lsf.conf file.
- [Adding a user share to a fair share queue](#)
Use the **bconf addmember** command to add a user share to a fair share queue.
- [View bconf records](#)
- [Merge configuration files](#)

bconf command authentication

Regular users can run the **bconf hist** command queries. Only cluster administrators and root can run all **bconf** commands.

All requests by the **bconf** command must be made from static servers; **bconf** command requests from dynamic hosts or client hosts are not accepted.

User group administrators can do the following depending on their rights:

- With usershares rights, user group administrators can adjust user shares by using the **bconf update**, **addmember**, or **rmmember** commands
- With full rights, user group administrators can adjust both user shares and group members by using the **bconf update** command, delete the user group by using the **bconf delete** command, and create new user groups by using the **bconf create** command.

Note: User group admins with full rights can add a user group member to the user group only if they also have full rights for the member user group.

If a user group administrator adds a user group with the **bconf create** command, the user group administrator is automatically added to the GROUP_ADMIN parameter in the lsb.users file with full rights for the new user group.

For more information about user group administrators, see [User groups in LSF](#) and [lsb.users](#).

Enabling live reconfiguration

Enable live reconfiguration by defining the LSF_LIVE_CONFDIR parameter in the lsf.conf file.

Before you begin

- Ensure that all configuration files are free of warning messages when running the **badadmin reconfig** command.
- Merge multiple sections in configuration files where possible.
- Ensure that the configuration files follow the order and syntax that is given in the template files.

Procedure

1. Define the LSF_LIVE_CONFDIR parameter with an absolute path in the lsf.conf file.
2. Run the **lsadmin reconfig** and **badadmin mbdrestart** commands to apply the new parameter setting.
Running the **bconf** command creates updated copies of changed configuration files in the directory that is specified by the LSF_LIVE_CONFDIR parameter.

Important: When a file exists in the directory set by the LSF_LIVE_CONFDIR parameter, all LSF restart and reconfigure commands read the file in this directory instead of the equivalent configuration file in the LSF_CONFDIR directory.

Adding a user share to a fair share queue

Use the **bconf addmember** command to add a user share to a fair share queue.

About this task

You can add a member and share to a fair share queue in the `lsb.queues` file by using live reconfiguration.

Procedure

Run the **bconf addmember** command.

```
bconf addmember queue=queue_name "fair share=USER_SHARES[[user_name, share]]"
```

For example, if you have the following existing configuration in the `lsb.queues` file:

```
...
Begin queue
QUEUE_NAME=my_queue
fair share=USER_SHARES[[tina, 10] [default, 3]]
End Queue
...
```

Add a user group and share:

```
bconf addmember queue=my_queue "fair share=USER_SHARES[[ug1, 10]]"
bconf: Request for queue <my_queue> accepted
```

After it is accepted by the **bconf** command, the new share definition appears in the **bqueue -l** command output:

```
bqueues -l my_queue
...
USER_SHARES: [tina, 10] [ug1, 10] [default, 3]
...
```

Important: If `USER_SHARES=[]` is defined for the fair share queue and a share value is added to the `USER_SHARES` parameter, the value `[default, 1]` is also added automatically.

For example, if you have the following configuration in the `lsb.queues` file:

```
...
Begin Queue
QUEUE_NAME=queue16
fair share=USER_SHARES[]
End Queue
...
```

Add a share value:

```
bconf addmember queue=queue16 "fair share=USER_SHARES[[user3, 10]]"
bconf: Request for queue <queue16> accepted
```

After it is accepted by the **bconf** command, the new share definition appears in the **bqueue -l** command output:

```
bqueues -l queue16
...
USER_SHARES: [user3, 10] [default, 1]
...
```

View bconf records

About this task

All successful and partially successful **bconf** requests are recorded in the history file `liveconf.hist` located under `$LSB_SHAREDIR/cluster_name/logdir`.

Procedure

Run **bconf hist**.

All **bconf** requests made by the current user are displayed.

For example:

```
bconf hist
TIME          OBJECT  NAME  ACTION  USER  IMPACTED_OBJ
Nov 9 15:19:46 2009  limit  aaa    update  liam  limit=aaa
Nov 9 15:19:28 2009  queue  normal update  liam  queue=normal
```

View bconf records for a specific configuration file

Procedure

Run **bconf hist -f *config_file***

where *config_file* is one of `lsb.resources`, `lsb.queues`, `lsb.users`, `lsb.hosts`, `lsf.cluster.clustername`, or `lsb.serviceclasses`.

All entries in the bconf history file which changed the specified configuration file are listed. This includes changes made directly, such as changing a limit, and indirectly, such as deleting the usergroup which must then be removed from the limit.

For example:

```
bconf hist -u all -f lsb.resources
TIME          OBJECT  NAME  ACTION  USER  IMPACTED_OBJ
Nov 9 15:19:50 2009  limit  aaa    create  robby  limit=aaa
Nov 9 15:19:46 2009  limit  aaa    update  liam   limit=aaa
Nov 9 15:19:37 2009  usergroup ug1    delete  robby  queue=normal owners*
                                         limit=bbb
                                         usergroup=ug1
```

View bconf records for a specific type of object

Procedure

Run **bconf hist -o *object_type***

where *object_type* is one of: `user`, `usergroup`, `host`, `hostgroup`, `queue`, `limit`, `gpool`

All entries in the bconf history file which changed the specified object are listed.

For example:

```
bconf hist -u all -o queue
TIME          OBJECT  NAME  ACTION  USER  IMPACTED_OBJ
Nov 9 15:19:28 2009  queue  normal update  liam  queue=normal
Nov 9 15:19:37 2009  usergroup ug1    delete  robby  queue=normal owners*
                                         limit=bbb
                                         usergroup=ug1
```

Merge configuration files

About this task

Any changes made to configuration files using the **bconf** command result in changed configuration files written to the directory set by `LSF_LIVE_CONFDIR` in `lsf.conf`. LSF restarting and reconfiguration uses configuration files in `LSF_LIVE_CONFDIR` if they exist.

Make live reconfiguration changes permanent by copying changed configuration files into the `LSF_CONFDIR` directory.

Important:

Remove LSF_LIVE_CONFDIR configuration files or merge files into LSF_CONFDIR before disabling **bconf**, upgrading LSF, applying patches to LSF, or adding server hosts.

Procedure

1. Locate the live reconfiguration directory set in LSF_LIVE_CONFDIR in lsf.conf.
The bconf command can result in updated copies of the following configuration files:
 - lsb.resources
 - lsb.queues
 - lsb.users
 - lsb.hosts
 - lsf.cluster.*clustername*
2. Copy any existing configuration files from LSF_LIVE_CONFDIR to the main configuration file directory set by LSF_CONFDIR in lsf.conf.
3. Delete configuration files from LSF_LIVE_CONFDIR.
Running **badmin mbdrestart** or **lsadmin reconfig** now, LSF_LIVE_CONFDIR is empty, and the configuration files that are found in LSF_CONFDIR are used.

Adding cluster administrators

Add or change the list of administrators for your cluster.

About this task

Primary Cluster Administrator

Required. The first cluster administrator, specified during installation. The primary LSF administrator account owns the configuration and log files. The primary LSF administrator has permission to perform cluster-wide operations, change configuration files, reconfigure the cluster, and control jobs submitted by all users.

Other Cluster Administrators

Optional. Might be configured during or after installation.

Cluster administrators can perform administrative operations on all jobs and queues in the cluster. Cluster administrators have the same cluster-wide operational privileges as the primary LSF administrator except that they do not have permission to change LSF configuration files.

Procedure

1. In the `ClusterAdmins` section of the `lsf.cluster.cluster_name` file, specify the list of cluster administrators following `ADMINISTRATORS`, separated by spaces.
You can specify user names and group names.

The first administrator in the list is the primary LSF administrator. All others are cluster administrators.

```
Begin ClusterAdmins
ADMINISTRATORS = lsfadmin admin1 admin2
End ClusterAdmins
```

2. Save your changes.
3. Restart all LIMs for the server host LIMs to pick up the new LSF administrators.
4. Run **badmin mbdrestart** to restart **mbatchd**.

Working with hosts

Check the status of hosts in your cluster, view information about your hosts, control hosts. Add and remove hosts in your cluster.

- [Host status](#)
Host status describes the ability of a host to accept and run batch jobs in terms of daemon states, load levels, and administrative controls. The **bhosts** and **lsload** commands display host status.
- [View host information](#)
- [Controlling hosts](#)
- [Connecting to a job execution host or container](#)
Use the **battach** command to connect (attach) to a job execution host or container for debugging or general connectivity purposes.
- [Host names](#)
LSF needs to match host names with the corresponding Internet host addresses.

Host status

Host status describes the ability of a host to accept and run batch jobs in terms of daemon states, load levels, and administrative controls. The **bhosts** and **lsload** commands display host status.

bhosts

Displays the current status of the host:

STATUS	Description
ok	Host is available to accept and run new batch jobs.
unavail	Host is down, or LIM and sbatchd are unreachable.
unreach	LIM is running but sbatchd is unreachable.
closed	Host does not accept new jobs. Use bhosts -l to display the reasons.

bhosts -l

Displays the closed reasons (for details, see the **bhosts** command reference). A closed host does not accept new batch jobs:

```
bhosts
HOST_NAME      STATUS      JL/U      MAX      NJOBS      RUN      SSUSP      USUSP      RSV
hostA          ok          -         55       2          2        0          0          0
hostB          closed      -         20       16         16       0          0          0
...
```

```
bhosts -l hostB
HOST hostB
STATUS      CPUF      JL/U      MAX      NJOBS      RUN      SSUSP      USUSP      RSV      DISPATCH_WINDOW
closed_Adm   23.10     -         55       2          2        0          0          0        -
CURRENT LOAD USED FOR SCHEDULING:
           r15s    r1m    r15m    ut      pg      io      ls      it      tmp      swp      mem      slots
Total      1.0    -0.0  -0.0    4%     9.4    148    2       3    4231M  698M  233M    8
Reserved   0.0     0.0   0.0     0%     0.0     0      0       0      0M     0M    0M      8
LOAD THRESHOLD USED FOR SCHEDULING:
           r15s    r1m    r15m    ut      pg      io      ls      it      tmp      swp      mem
loadSched  -         -      -       -       -       -      -       -       -       -       -
loadStop   -         -      -       -       -       -      -       -       -       -       -
           cpuspeed  bandwidth
loadSched  -         -
loadStop   -         -
```

lsload

Displays the current state of the host:

Status	Description
ok	Host is available to accept and run batch jobs and remote tasks.
-ok	LIM is running but RES is unreachable.
busy	Does not affect batch jobs, only used for remote task placement (such as lsrun). The value of a load index exceeded a threshold (configured in <code>lsf.cluster.cluster_name</code> , displayed by lshosts -l). Indices that exceed thresholds are identified with an asterisk (*).
lockW	Does not affect batch jobs, only used for remote task placement (such as lsrun). Host is locked by a run window (configured in <code>lsf.cluster.cluster_name</code> , displayed by lshosts -l).
lockU	Does not accept new batch jobs or remote tasks. An LSF administrator or root explicitly locked the host by using lsadmin limlock , or an exclusive batch job (bsub -x) is running on the host. Running jobs are not affected. Use lsadmin limunlock to unlock LIM on the local host.
unavail	Host is down, or LIM is unavailable.

```
lsload
HOST NAME      status  r15s  r1m  r15m  ut    pg  ls    it    tmp    swp    mem
hostA          ok     0.0   0.0   0.0   4%    0.4  0    4316  10G   302M   252M
hostB          ok     1.0   0.0   0.0   4%    8.2  2    14   4231M  698M   232M
...
```

View host information

About this task

LSF uses some or all of the hosts in a cluster as execution hosts. The host list is configured by the LSF administrator.

Procedure

- Use the **bhosts** command to view host information.
- Use the **lsload** command to view host load information.

To view...	Run...
All hosts in the cluster and their status	bhosts
Condensed host groups in an uncondensed format	bhosts -X
Detailed server host information	bhosts -l and lshosts -l
Host load by host	lsload
Host architecture information	lshosts
Host history	badmin hhist
Host model and type information	lsinfo
Job exit rate and load for hosts	bhosts -l and bhosts -x
Dynamic host information	lshosts

- [Customize host information output](#)
- [Customize host load information output](#)

View all hosts in the cluster and their status

Procedure

Run **bhosts** to display information about all hosts and their status.

bhosts displays condensed information for hosts that belong to condensed host groups. When displaying members of a condensed host group, **bhosts** lists the host group name instead of the name of the individual host. For example, in a cluster with a condensed host group (*groupA*), an uncondensed host group (*groupB* containing *hostC* and *hostE*), and a host that is not in any host group (*hostF*), **bhosts** displays the following:

```
bhosts
```

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
groupA	ok	5	8	4	2	0	1	1
hostC	ok	-	3	0	0	0	0	0
hostE	ok	2	4	2	1	0	0	1
hostF	ok	-	2	2	1	0	1	0

Define condensed host groups in the HostGroups section of *lsb.hosts*.

View uncondensed host information

Procedure

Run **bhosts -X** to display all hosts in an uncondensed format, including those belonging to condensed host groups:

```
bhosts -X
```

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
hostA	ok	2	2	0	0	0	0	0
hostD	ok	2	4	2	1	0	0	1
hostB	ok	1	2	2	1	0	1	0
hostC	ok	-	3	0	0	0	0	0
hostE	ok	2	4	2	1	0	0	1
hostF	ok	-	2	2	1	0	1	0

View detailed server host information

Procedure

Run **bhosts -l host_name** and **lshosts -l host_name** to display all information about each server host such as the CPU factor and the load thresholds to start, suspend, and resume jobs:

```
bhosts -l hostB
HOST hostB
STATUS  CPUF  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV  DISPATCH_WINDOWS
ok      20.20 -    -    0      0      0      0      0    -
CURRENT LOAD USED FOR SCHEDULING:
          r15s  r1m   r15m  ut   pg   io   ls   it   tmp   swp   mem   slots
Total    0.1   0.1   0.1   9%  0.7  24   17   0   394M  396M  12M   8
Reserved 0.0   0.0   0.0   0%  0.0  0    0    0    0M   0M   0M   8
LOAD THRESHOLD USED FOR SCHEDULING:
          r15s  r1m   r15m  ut   pg   io   ls   it   tmp   swp   mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

          cpuspeed  bandwidth
loadSched          -          -
loadStop           -          -

lshosts -l hostB
HOST_NAME: hostB
type model cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores nthreads
LINUX86 PC6000 116.1      2      1  2016M  1983M  72917M      0    Yes    1    2    2

RESOURCES: Not defined
RUN_WINDOWS: (always open)

LOAD_THRESHOLDS:
          r15s  r1m   r15m  ut   pg   io   ls   it   tmp   swp   mem
          -    1.0   -    -    -    -    -    -    -    -    4M
```

View host load by host

About this task

The **lsload** command reports the current status and load levels of hosts in a cluster. The **lshosts -l** command shows the load thresholds.

Procedure

Run **lsload** to see load levels for each host:

```
lsload
HOST_NAME status r15s r1m r15m ut pg ls it tmp swp mem
hostD      ok      1.3 1.2 0.9 92% 0.0 2 20 5M 148M 88M
hostB      -ok     0.1 0.3 0.7 0%  0.0 1 67 45M 25M 34M
hostA      busy     8.0 *7.0 4.9 84% 4.6 6 17 1M 81M 27M
```

The first line lists the load index names, and each following line gives the load levels for one host.

View host architecture (type and model) information

About this task

The **lshosts** command displays configuration information about hosts. All these parameters are defined by the LSF administrator in the LSF configuration files, or determined by the LIM directly from the system.

Host types represent binary compatible hosts; all hosts of the same type can run the same executable. Host models give the relative CPU performance of different processors.

Procedure

Run **lshosts** to see configuration information about hosts:

```
lshosts
HOST_NAME  type    model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
hostD      SUNSOL  SunSparc  6.0    1    64M    112M    Yes    (solaris cserver)
hostM      RS6K    IBM350  7.0    1    64M    124M    Yes    (cserver aix)
hostC      RS6K    R10K   14.0   16   1024M  1896M    Yes    (cserver aix)
hostA      HPPA    HP715   6.0    1    98M    200M    Yes    (hpux fserver)
```

In the preceding example, the host type SUNSOL represents Sun SPARC systems running Solaris. The **lshosts** command also displays the resources available on each host.

type

The host CPU architecture. Hosts that can run the same binary programs should have the same type.

An UNKNOWN type or model indicates that the host is down, or LIM on the host is down.

When automatic detection of host type or model fails (the host type configured in `lsf.shared` cannot be found), the type or model is set to DEFAULT. LSF does work on the host, but a DEFAULT model might be inefficient because of incorrect CPU factors. A DEFAULT type may also cause binary incompatibility because a job from a DEFAULT host type can be migrated to another DEFAULT host type. automatic detection of host type or model has failed, and the host type configured in `lsf.shared` cannot be found.

View host history

Procedure

Run **badadmin hhist** to view the history of a host such as when it is opened or closed:

```
badadmin hhist hostB
Wed Nov 20 14:41:58: Host <hostB> closed by administrator <lsf>.
Wed Nov 20 15:23:39: Host <hostB> opened by administrator <lsf>.
```

View host model and type information

Procedure

1. Run **lsinfo -m** to display information about host models that exist in the cluster:

```
lsinfo -m
MODEL_NAME      CPU_FACTOR      ARCHITECTURE
PC1133          23.10           x6_1189_PentiumIIICoppermine
HP9K735         4.50            HP9000735_125
HP9K778         5.50            HP9000778
Ultra5S         10.30           SUNWUltra510_270_sparcv9
Ultra2          20.20           SUNWUltra2_300_sparc
Enterprise3000  20.00           SUNWUltraEnterprise_167_sparc
```

2. Run **lsinfo -M** to display all host models that are defined in `lsf.shared`:

```
lsinfo -M
MODEL_NAME      CPU_FACTOR      ARCHITECTURE
UNKNOWN_AUTO_DETECT 1.00           UNKNOWN_AUTO_DETECT
DEFAULT         1.00
LINUX133        2.50           x586_53_Pentium75
PC200           4.50           i86pc_200
Intel_IA64      12.00          ia64
Ultra5S         10.30          SUNWUltra5_270_sparcv9
PowerPC_G4      12.00          x7400G4
HP300           1.00
SunSparc        12.00
```

3. Run **lim -t** to display the type, model, and matched type of the current host. You must be the LSF administrator to use this command:

```
lim -t
Host Type       : NTX64
Host Architecture : EM64T_1596
Total NUMA Nodes : 1
Total Processors : 2
Total Cores     : 4
Total Threads   : 2
Matched Type    : NTX64
Matched Architecture : EM64T_3000
Matched Model    : Intel_EM64T
CPU Factor      : 60.0
```

View job exit rate and load for hosts

Procedure

1. Run **bhosts** to display the exception threshold for job exit rate and the current load value for hosts.
In the following example, `EXIT_RATE` for `hostA` is configured as four jobs per minute. `hostA` does not currently exceed this rate

```
bhosts -l hostA
HOST hostA
STATUS      CPUF  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV DISPATCH_WINDOW
ok          18.60 -    1    0      0      0      0      0      -

CURRENT LOAD USED FOR SCHEDULING:
      r15s  r1m  r15m  ut    pg    io    ls    it    tmp    swp    mem    slots
Total    0.0  0.0  0.0  0%   0.0    0    1    2   646M  648M  115M    8
Reserved  0.0  0.0  0.0  0%   0.0    0    0    0    0M   0M   0M    8

      share_rsrc host_rsrc
Total        3.0      2.0
Reserved     0.0      0.0

LOAD THRESHOLD USED FOR SCHEDULING:
      r15s  r1m  r15m  ut    pg    io    ls    it    tmp    swp    mem
loadSched  -    -    -    -    -    -    -    -    -    -    -
loadStop   -    -    -    -    -    -    -    -    -    -    -
```

	cpuspeed	bandwidth
loadSched	-	-
loadStop	-	-
THRESHOLD AND LOAD USED FOR EXCEPTIONS:		
	JOB_EXIT_RATE	
Threshold	4.00	
Load	0.00	

2. Use **bhosts -x** to see hosts whose job exit rate has exceeded the threshold for longer than `JOB_EXIT_RATE_DURATION`, and are still high. By default, these hosts are closed the next time LSF checks host exceptions and invokes **eadmin**.
If no hosts exceed the job exit rate, **bhosts -x** displays:

```
There is no exceptional host found
```

View dynamic host information

Procedure

Use **lshosts** to display information about dynamically added hosts.

An LSF cluster may consist of static and dynamic hosts. The **lshosts** command displays configuration information about hosts. All these parameters are defined by the LSF administrator in the LSF configuration files, or determined by the LIM directly from the system.

Host types represent binary compatible hosts; all hosts of the same type can run the same executable. Host models give the relative CPU performance of different processors. Server represents the type of host in the cluster. “Yes” is displayed for LSF servers, “No” is displayed for LSF clients, and “Dyn” is displayed for dynamic hosts.

For example:

```
lshosts
HOST_NAME  type    model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
hostA      SOL64   Ultra60F 23.5   1     64M    112M    Yes    ()
hostB      LINUX86 Opteron8 60.0   1     94M    168M    Dyn    ()
```

In the preceding example, `hostA` is a static host while `hostB` is a dynamic host.

Customize host information output

By default, the **bhosts** command displays a predefined set of host information. While you can use various **bhosts** options to display specific host information based on your needs, you can also customize the specific fields that **bhosts** displays. Customize output to create a specific **bhosts** output format that shows all the required information so you can easily parse the information by using custom scripts or to display the information in a predefined format.

Use the `LSB_BHOSTS_FORMAT` parameter in `lsf.conf` or the `LSB_BHOSTS_FORMAT` runtime environment variable to define the default **bhosts** output format for LSF:

```
LSB_BHOSTS_FORMAT="field_name[:[-][output_width]]... [delimiter='character']"
```

Use the **bhosts -o** option to define the custom output at the command level:

```
bhosts ... -o "field_name[:[-][output_width]]... [delimiter='character']"
```

The following alternative method of using **bhosts -o** is useful for special delimiter characters in a csh environment (for example, `$`):

```
bhosts ... -o 'field_name[:[-][output_width]]... [delimiter="character"]'
```

- Specify which **bhosts** fields (or aliases instead of the full field names), in which order, and with what width to display.
- Specify only the **bhosts** field name or alias to set its output to unlimited width and left justification.
- (Available starting in Fix Pack 14) Specify `all` to display all fields. Specify the colon (`:`) with an output width that applies to all fields.

- Specify the colon (:) without a width to set the output width to the recommended width for that field.
- Specify the colon (:) with a width to set the maximum number of characters to display for the field. When its value exceeds this width, **bhosts** truncates the ending characters.
- Specify a hyphen (-) to set right justification when **bhosts** displays the output for the specific field. If not specified, the default is to set left justification when **bhosts** displays the output for a field.
- Specify a second colon (:) with a unit to specify a unit prefix for the output for the following fields: `mem`, `max_mem`, `avg_mem`, `memlimit`, `swap`, `swaplimit`, `corelimit`, `stacklimit`, and `husage` (for `husage`, the unit prefix is for `mem` and `swap` resources only).

This unit is KB (or K) for kilobytes, MB (or M) for megabytes, GB (or G) for gigabytes, TB (or T) for terabytes, PB (or P) for petabytes, EB (or E) for exabytes, ZB (or Z) for zettabytes, or S to automatically adjust the value to a suitable unit prefix and remove the "bytes" suffix from the unit. The default is to automatically adjust the value to a suitable unit prefix, but keep the "bytes" suffix in the unit.

The display value keeps two decimals but rounds up the third decimal. For example, if the unit prefix is set to G, 10M displays as 0.01G.

The unit prefix specified here overrides the value of the `LSB_UNIT_FOR_JOBS_DISPLAY` environment variable, which also overrides the value of the `LSB_UNIT_FOR_JOBS_DISPLAY` parameter in the `lsf.conf` file.

- Use `delimiter=` to set the delimiting character to display between different headers and fields. This delimiter must be a single character. By default, the delimiter is a space.

The **bhosts -o** option overrides the `LSB_BHOSTS_FORMAT` environment variable, which overrides the `LSB_BHOSTS_FORMAT` setting in `lsf.conf`.

Output customization applies only to the output for certain **bhosts** options:

- `LSB_BHOSTS_FORMAT` and **bhosts -o** both apply to output for the **bhosts** command with no options, and for **bhosts** options with output that filter information, including the following options: `-a`, `-alloc`, `-cname`, `-R`, `-x`, `-X`.
- `LSB_BHOSTS_FORMAT` and **bhosts -o** do not apply to output for **bhosts** options that use a modified format, including the following options: `-aff`, `-e`, `-l`, `-s`, `-w`.

The following are the field names used to specify the **bhosts** fields to display, with valid widths and any supported aliases (which you can use instead of the field names). Units of measurement for the fields are an automatically chosen units of bytes (such as gigabytes, megabytes, and so on), depending on the field name.

Table 1. Output fields for bhosts

Field name	Width	Alias
host_name	20	hname
status	15	stat
cpuf	10	
jl_u	8	jlu
max	8	
njobs	8	
run	8	
ssusp	8	
ususp	8	
rsv	8	
dispatch_window	50	dispwin
ngpus	8	ng
ngpus_alloc	8	ngu
ngpus_excl_alloc	8	ngx
ngpus_shared_alloc	8	ngs
ngpus_shared_jexcl_alloc	8	ngsjx
ngpus_excl_avail	8	ngfx
ngpus_shared_avail	8	ngfs
attribute	50	attr
mig_alloc	5	

Field name	Width	Alias
comments Note: If combined with the bhosts -json option, this field displays full details of host closure events such as event time, administrator ID, lock ID, and comments, as shown in the bhosts -l option.	128	
available_mem (Available starting in Fix Pack 14)	15	
reserved_mem (Available starting in Fix Pack 14)	15	
total_mem (Available starting in Fix Pack 14)	15	

Field names and aliases are not case-sensitive. Valid values for the output width are any positive integer 1 - 4096.

Remove column headings from the host information output

Use the **bhosts -noheader** option to remove column headings from the **bhosts** output. When **bhosts -noheader** is specified, **bhosts** displays the values of the fields without displaying the names of the fields. This option is useful for script parsing, when column headings are not necessary.

This option applies to output for the **bhosts** command with no options, and to output for all **bhosts** options with output that uses column headings, including the following options: -a, -alloc, -cname, -e, -o, -R, -s, -w, -x, -X.

This option does not apply to output for **bhosts** options that do not use column headings, including the following options: -aff, -json, -l.

View customized host information in JSON format

Use the **bhosts -json** option to view the customized **bhosts** output in JSON format. Since JSON is a customized output format, you must use the **bhosts -json** option together with the -o option.

Customize host load information output

By default, the **lsload** command displays a predefined set of load information for hosts. While you can use various **lsload** options to display specific load information based on your needs, you can also customize the specific fields that **lsload** displays. Customize output to create a specific **lsload** output format that shows all the required information so you can easily parse the information by using custom scripts or to display the information in a predefined format.

Use the LSF_LSLOAD_FORMAT parameter in lsf.conf or the LSF_LSLOAD_FORMAT runtime environment variable to define the default **lsload** output format for LSF:

```
LSF_LSLOAD_FORMAT="field_name[:[-][output_width]][:unit]] ... [delimiter='character']"
```

Use the **lsload -o** option to define the custom output at the command level:

```
lsload ...-o field_name[:[-][output_width]][:unit]] ... [delimiter='character']"
```

- Specify which **lsload** fields, in which order, and with what width to display.
- Specify the asterisk wildcard character (*) in the field name to specify multiple external resource names. You can only specify one asterisk, but this asterisk can be at any position in the field name.
For example, running **lsload -o "gpu_mode*"** shows fields such as gpu_mode0, gpu_mode1, gpu_mode2, gpu_mode10, gpu_mode11, and gpu_mode12.
- Specify only the **lsload** field name to set its output to unlimited width and left justification.
- Specify the width colon (:) without a width to set the output width to the recommended width for that field.
- Specify the width colon (:) with a width to set the maximum number of characters to display for the field. When its value exceeds this width, **lsload** truncates the ending characters.

- Specify a hyphen (-) to set right justification when **lsload** displays the output for the specific field. If not specified, the default is to set left justification when **lsload** displays the output for a field.
- Specify the unit colon (:) with a unit to set the unit for the output of the specific field:
 - Specify **S** to use a built-in conversion for space or capacity, such as memory or disk space. Values are automatically scaled for M (MB), G (GB), and T (TB), where the default unit is M (MB).
For example, when displaying the mem field with a specified width of 3,
 - For a value of 30, running the **lsload -o "mem:3:S"** command shows 30.0M.
 - For a value of 4096, running the **lsload -o "mem:3:S"** command shows 4.0G.
 - For a value of 5000000, running the **lsload -o "mem:3:S"** command shows 4.8T.
 - Specify **D** to use a built-in conversion for duration or time, such as memory or disk space. Values are automatically scaled for s (seconds), m (minutes), h (hours), and d (days), where the default unit is s (seconds). The automatically scaled value is rounded up after the first decimal point.
For example, when displaying the external mytime resource field with a specified width of 5,
 - For a value of 30, running the **lsload -o "mytime:5:D"** command shows 30.0s.
 - For a value of 8000, running the **lsload -o "mytime:5:D"** command shows 2.2h.
 - For a value of 5000000, running the **lsload -o "mytime:5:D"** command shows 57.8d.
 - Specify any other string of 1 - 3 characters and the characters are used as is in the field value. The first character must be a letter (upper or lower case). The second and third characters must be an alphanumeric character.
For example, when displaying the external gpu_temp resource with a width of 3, running the **lsload -o "gpu_temp:3:C"** command for a value of 30 shows 30C
- Use **delimiter=** to set the delimiting character to display between different headers and fields. This delimiter must be a single character. By default, the delimiter is a space.

The **lsload -o** option overrides the LSF_LSLOAD_FORMAT environment variable, which overrides the LSF_LSLOAD_FORMAT setting in lsf.conf.

Output customization applies only to the output for certain **lsload** options:

- LSF_LSLOAD_FORMAT and **lsload -o** both apply to output for the **lsload** command with no options, and for **lsload** options with short form output that filter information, including the following options: -a, -E, -cname, -N, -n, -R.
- LSF_LSLOAD_FORMAT and **lsload -o** do not apply to output for **lsload** options that use a modified format, including the following options: -I, -l, -w, -s.

By default, the **lsload** command displays the built-in resource indices. You can also specify the names of external resources. The following are the field names for the built-in resource indices that are used to specify the **lsload** fields to display, recommended width, and units of measurement for the displayed field:

Table 1. Output fields for lsload

Field name	Width	Unit
HOST_NAME	20	
status	15	
r15s	6	
r1m	6	
r15m	6	
ut	6	
pg	6	
ls	6	
it	6	
io	6	
tmp	10	LSF_UNIT_FOR_LIMITS in lsf.conf (KB by default)
swp	10	LSF_UNIT_FOR_LIMITS in lsf.conf (KB by default)
mem	10	LSF_UNIT_FOR_LIMITS in lsf.conf (KB by default)
gpu_status* For example, gpu_status0 and gpu_status1 if there are two GPUs.	10	

Field name	Width	Unit
gpu_error* For example, gpu_error0 and gpu_error1 if there are two GPUs.	20	

Field names are case-sensitive. Valid values for the output width are any positive integer 1 - 4096.

View customized host load information in JSON format

Use the **lsload -json** option to view the customized **lsload** output in JSON format. Since JSON is a customized output format, you must use the **lsload -json** option together with the -o option.

Controlling hosts

About this task

Hosts are opened and closed by:

Procedure

- an LSF Administrator or root issuing a command
- configured dispatch windows

Closing a host

Procedure

Run **badadmin hclose**:

```
badadmin hclose hostB
Close <hostB> ..... done
```

If the command fails, it might be because the host is unreachable through network problems, or because the daemons on the host are not running.

Opening a host

Procedure

Run **badadmin hopen**:

```
badadmin hopen hostB
Open <hostB> ..... done
```

Configuring dispatch windows

About this task

A dispatch window specifies one or more time periods during which a host receive new jobs. The host does not receive jobs outside of the configured windows. Dispatch windows do not affect job submission and running jobs (they are allowed to run until completion). By default, dispatch windows are not configured.

To configure dispatch windows:

Procedure

1. Edit `lsb.hosts`.

- Specify one or more time windows in the DISPATCH_WINDOW column:

```
Begin Host
HOST_NAME      rlm      pg      ls      tmp      DISPATCH_WINDOW
...
hostB          3.5/4.5  15/    12/15   0        (4:30-12:00)
...
End Host
```

- Reconfigure the cluster:
 - Run **lsadmin reconfig** to reconfigure LIM.
 - Run **badadmin reconfig** to reconfigure mbatchd.
- Run **bhosts -l** to display the dispatch windows.

Login a comment when closing or opening a host

Procedure

- Use the **-C** option of **badadmin hclose** and **badadmin hopen** to log an administrator comment in lsb.events:

```
badadmin hclose -C "Weekly backup" hostB
```

The comment text `Weekly backup` is recorded in lsb.events. If you close or open a host group, each host group member displays with the same comment string.

A new event record is recorded for each host open or host close event. For example:

```
badadmin hclose -C "backup" hostA
```

followed by

```
badadmin hclose -C "Weekly backup" hostA
```

generates the following records in lsb.events:

```
"HOST_CTRL" "7.0 1050082346 1 "hostA" 32185 "lsfadmin" "backup"
"HOST_CTRL" "7.0 1050082373 1 "hostA" 32185 "lsfadmin" "Weekly backup"
```

- Use **badadmin hist** or **badadmin hhist** to display administrator comments for closing and opening hosts:

```
badadmin hhist
Fri Apr 4 10:35:31: Host <hostB> closed by administrator
<lsfadmin> Weekly backup.
```

bhosts -l also displays the comment text:

```
bhosts -l

HOST hostA
STATUS      CPUF    JL/U    MAX    NJOBS    RUN    SSUSP    USUSP    RSV    DISPATCH_WINDOW
closed_Adm  1.00    -      -      0        0        0        0        0      -

CURRENT LOAD USED FOR SCHEDULING:
              r15s    rlm    r15m    ut      pg      io      ls      it      tmp      swp      mem      slots
Total         0.0    0.0    0.0    2%     0.0     64      2     11    7117M    512M    432M      8
Reserved      0.0    0.0    0.0    0%     0.0      0      0      0      0M      0M      0M      8

LOAD THRESHOLD USED FOR SCHEDULING:
              r15s    rlm    r15m    ut      pg      io      ls      it      tmp      swp      mem
loadSched     -      -      -      -      -      -      -      -      -      -      -
loadStop      -      -      -      -      -      -      -      -      -      -      -

              cpuspeed    bandwidth
loadSched         -      -
loadStop          -      -

THRESHOLD AND LOAD USED FOR EXCEPTIONS:
              JOB_EXIT_RATE
Threshold       2.00
```

```
Load          0.00
ADMIN ACTION COMMENT: "Weekly backup"
```

Using lock IDs to specify multiple reasons for closing a host

About this task

Different users can close a host for multiple reasons by specifying a different lock ID for each reason. For example, userA might be updating an application while userB is configuring the operating system. The host remains closed until both users complete their updates and open the host using their specific lock IDs.

Procedure

1. Use the **-i** option of **badadmin hclose** when closing a host to specify a lock ID to attach to the closed host. Optionally, use the **-C** option to attach a comment to the lock ID that explains the closing reason in more detail.
`badadmin hclose -i lock_id [-C comment]`

If the host is already closed, this command stacks the new lock ID with any existing lock IDs on the closed host to ensure that the host remains closed if at least one lock ID is still attached to the host.

Each lock ID is a string that can contain up to 128 alphanumeric and underscore (`_`) characters. The keyword `all` is reserved and cannot be used as the lock ID.

userA closes the host to update application1:

```
badadmin hclose -i "lock_update_app1" -C "Updating application1"
```

userB closes the host to configure the operating system:

```
badadmin hclose -i "lock_config_os" -C "Configuring OS"
```

2. Use the **bhosts -l** command option to view all lock IDs and comments in tabular format, if there are any lock IDs that are attached to the host.

```
...
ADMIN ACTION COMMENTS:
LockId      EventTime      Admin      Message
lock_update_app1 Mon Dec  2 19:41:44    userA      Updating application1
lock_config_os  Mon Dec  2 19:51:03    userB      Configuring OS
...
```

3. Use the **-i** option of **badadmin hopen** to remove the specified lock ID from the closed host. Optionally, use the **-C** option to add comments.
`badadmin hopen -i "lock_id ... | all" [-C comment]`

Specify a space-separated list of lock IDs to remove multiple lock IDs, or use the `all` keyword to remove all lock IDs from the closed host. If there are no more lock IDs attached to the host, this command also opens the host.

userB finished configuring the operating system and removes the `lock_config_os` lock ID:

```
badadmin hopen -i "lock_config_os" -C "Finished OS configuration"
```

Since userA is still updating application1 and the lock ID is still attached to this host, the host remains closed. userA finished updating application1 and removes the `lock_update_app1` lock ID:

```
badadmin hopen -i "lock_update_app1" -C "Finished updating application1"
```

There are no more lock IDs attached to the host, so this command also opens the host.

How events display and are recorded in the lease model of the LSF multicluster capability

In the resource lease model of the LSF multicluster capability, host control administrator comments are recorded only in the `lsb.events` file on the local cluster. **badadmin hist** and **badadmin hhist** display only events that are recorded locally. Host control

messages are not passed between clusters in the lease model. For example, if you close an exported host in both the consumer and the provider cluster, the host close events are recorded separately in their local `lsb.events`.

Connecting to a job execution host or container

Use the **battach** command to connect (attach) to a job execution host or container for debugging or general connectivity purposes.

Procedure

1. Use the **battach** command with no options to connect to the job execution host or container for the specified job ID with the default options.

```
battach job_id
```

```
battach job_id[job_index]
```

Runs an interactive `/bin/sh` shell to connect to the first job execution host or container for the specified job.

2. Use the **battach -L** command option or the `LSB_BATTACH_DEFAULT_SHELL` environment variable to specify an alternate interactive shell process.

- `battach -L "shell_path" job_id`
- `battach -L "shell_path" job_id[job_index]`
- `LSB_BATTACH_DEFAULT_SHELL="shell_path"`

The path that you specify must be an absolute file path. Relative file paths are not valid. If you specify a binary file instead of a shell, **battach** runs the specified binary file and exits.

The **battach -L** command option overrides the `LSB_BATTACH_DEFAULT_SHELL` environment variable.

3. For Docker parallel jobs, use the **battach -m** command option to specify the job execution host or container.

```
battach -m "host_name" job_id
```

The host that you specify must be one of the execution hosts for the Docker parallel job.

For all other jobs (such as Docker sequential jobs or non-Docker jobs), you can only connect to the first job execution host or container.

Host names

LSF needs to match host names with the corresponding Internet host addresses.

LSF looks up host names and addresses the following ways:

- In the `/etc/hosts` file
- Sun Network Information Service/Yellow Pages (NIS or YP)
- Internet Domain Name Service (DNS).
DNS is also known as the Berkeley Internet Name Domain (BIND) or `named`, which is the name of the BIND daemon.

Each host is configured to use one or more of these mechanisms.

- [Hosts with multiple addresses](#)
- [Using IPv6 addresses](#)
- [Specifying host names with condensed notation](#)

Network addresses

Each host has one or more network addresses; usually one for each network to which the host is directly connected. Each host can also have more than one name.

Official host name

The first name configured for each address is called the official name.

Host name aliases

Other names for the same host are called aliases.

LSF uses the configured host naming system on each host to look up the official host name for any alias or host address.

This means that you can use aliases as input to LSF, but LSF always displays the official name.

Use host name ranges as aliases

The default host file syntax is as follows:

```
ip_address official_name [alias [alias ...]]
```

This syntax is powerful and flexible, but it is difficult to configure in systems where a single host name has many aliases, and in multihomed host environments.

In these cases, the hosts file can become very large and unmanageable, and configuration is prone to error.

The syntax of the LSF hosts file supports host name ranges as aliases for an IP address. This simplifies the host name alias specification.

To use host name ranges as aliases, the host names must consist of a fixed node group name prefix and node indices, specified in a form like:

```
host_name[index_x-index_y, index_m, index_a-index_b]
```

For example:

```
atlasD0[0-3,4,5-6, ...]
```

is equivalent to:

```
atlasD0[0-6, ...]
```

The node list does not need to be a continuous range (some nodes can be configured out). Node indices can be numbers or letters (both upper case and lower case).

Example

Some systems map internal compute nodes to single LSF host names. A host file might contain 64 lines, each specifying an LSF host name and 32 node names that correspond to each LSF host:

```
...  
177.16.1.1 atlasD0 atlas0 atlas1 atlas2 atlas3 atlas4 ... atlas31  
177.16.1.2 atlasD1 atlas32 atlas33 atlas34 atlas35 atlas36 ... atlas63  
...
```

In the new format, you still map the nodes to the LSF hosts, so the number of lines remains the same, but the format is simplified because you only have to specify ranges for the nodes, not each node individually as an alias:

```
...  
177.16.1.1 atlasD0 atlas[0-31]  
177.16.1.2 atlasD1 atlas[32-63]  
...
```

You can use either an IPv4 or an IPv6 format for the IP address (if you define the parameter `LSF_ENABLE_SUPPORT_IPV6` in `lsf.conf`).

Host name services

Solaris

On Solaris systems, the `/etc/nsswitch.conf` file controls the name service.

Other UNIX platforms

On other UNIX platforms, the following rules apply:

- If your host has an `/etc/resolv.conf` file, your host is using DNS for name lookups
- If the command `ypcat hosts` prints out a list of host addresses and names, your system is looking up names in NIS
- Otherwise, host names are looked up in the `/etc/hosts` file

For more information

The man pages for the `gethostbyname` function, the `ypbind` and `named` daemons, the `resolver` functions, and the `hosts`, `svc.conf`, `nsswitch.conf`, and `resolv.conf` files explain host name lookups in more detail.

Hosts with multiple addresses

Multi-homed hosts

Hosts that have more than one network interface usually have one Internet address for each interface. Such hosts are called *multi-homed hosts*. For example, dual-stack hosts are multi-homed because they have both an IPv4 and an IPv6 network address.

LSF identifies hosts by name, so it needs to match each of these addresses with a single host name. To do this, the host name information must be configured so that all of the Internet addresses for a host resolve to the same name.

There are two ways to do it:

- Modify the system hosts file (`/etc/hosts`) and the changes will affect the whole system
- Create an LSF hosts file (`LSF_CONFDIR/hosts`) and LSF will be the only application that resolves the addresses to the same host

Multiple network interfaces

Some system manufacturers recommend that each network interface, and therefore, each Internet address, be assigned a different host name. Each interface can then be directly accessed by name. This setup is often used to make sure NFS requests go to the nearest network interface on the file server, rather than going through a router to some other interface. Configuring this way can confuse LSF, because there is no way to determine that the two different names (or addresses) mean the same host. LSF provides a workaround for this problem.

All host naming systems can be configured so that host address lookups always return the same name, while still allowing access to network interfaces by different names. Each host has an official name and a number of aliases, which are other names for the same host. By configuring all interfaces with the same official name but different aliases, you can refer to each interface by a different alias name while still providing a single official name for the host.

Configure the LSF hosts file

If your LSF clusters include hosts that have more than one interface and are configured with more than one official host name, you must either modify the host name configuration, or create a private hosts file for LSF to use.

The LSF hosts file is stored in `LSF_CONFDIR`. The format of `LSF_CONFDIR/hosts` is the same as for `/etc/hosts`.

In the LSF hosts file, duplicate the system hosts database information, except make all entries for the host use the same official name. Configure all the other names for the host as aliases so that you can still refer to the host by any name.

Example

For example, if your `/etc/hosts` file contains:

```
AA.AA.AA.AA  host-AA host # first interface
BB.BB.BB.BB  host-BB      # second interface
```

then the LSF_CONFDIR/hosts file should contain:

```
AA.AA.AA.AA  host host-AA # first interface
BB.BB.BB.BB  host host-BB # second interface
```

Example /etc/hosts entries

No unique official name

The following example is for a host with two interfaces, where the host does not have a unique official name.

```
# Address      Official name  Aliases
# Interface on network A
AA.AA.AA.AA    host-AA.domain host-AA host
# Interface on network B
BB.BB.BB.BB    host-BB.domain host-BB host
```

Looking up the address AA.AA.AA.AA finds the official name host-AA.domain. Looking up address BB.BB.BB.BB finds the name host-BB.domain. No information connects the two names, so there is no way for LSF to determine that both names, and both addresses, refer to the same host.

To resolve this case, you must configure these addresses using a unique host name. If you cannot make this change to the system file, you must create an LSF hosts file and configure these addresses using a unique host name in that file.

Both addresses have the same official name

Here is the same example, with both addresses configured for the same official name.

```
# Address      Official name  Aliases
# Interface on network A
AA.AA.AA.AA    host.domain    host-AA.domain host-AA host
# Interface on network B
BB.BB.BB.BB    host.domain    host-BB.domain host-BB host
```

With this configuration, looking up either address returns host.domain as the official name for the host. LSF (and all other applications) can determine that all the addresses and host names refer to the same host. Individual interfaces can still be specified by using the host-AA and host-BB aliases.

Example for a dual-stack host

Dual-stack hosts have more than one IP address. You must associate the host name with both addresses, as shown in the following example:

```
# Address      Official name  Aliases
# Interface IPv4
AA.AA.AA.AA    host.domain    host-AA.domain
# Interface IPv6
BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB host.domain    host-BB.domain
```

With this configuration, looking up either address returns host.domain as the official name for the host. LSF (and all other applications) can determine that all the addresses and host names refer to the same host. Individual interfaces can still be specified by using the host-AA and host-BB aliases.

Sun Solaris example

For example, Sun NIS uses the /etc/hosts file on the NIS management host as input, so the format for NIS entries is the same as for the /etc/hosts file. Since LSF can resolve this case, you do not need to create an LSF hosts file.

DNS configuration

The configuration format is different for DNS. The same result can be produced by configuring two address (A) records for each Internet address. Following the previous example:

```
# name      class  type address
host.domain IN    A    AA.AA.AA.AA
host.domain IN    A    BB.BB.BB.BB
```

host-AA.domain	IN	A	AA.AA.AA.AA
host-BB.domain	IN	A	BB.BB.BB.BB

Looking up the official host name can return either address. Looking up the interface-specific names returns the correct address for each interface.

For a dual-stack host:

#	name	class	type	address
	host.domain	IN	A	AA.AA.AA.AA
	host.domain	IN	A	BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB
	host-AA.domain	IN	A	AA.AA.AA.AA
	host-BB.domain	IN	A	BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB

PTR records in DNS

Address-to-name lookups in DNS are handled using PTR records. The PTR records for both addresses should be configured to return the official name:

#	address	class	type	name
	AA.AA.AA.AA.in-addr.arpa	IN	PTR	host.domain
	BB.BB.BB.BB.in-addr.arpa	IN	PTR	host.domain

For a dual-stack host:

#	address	class	type	name
	AA.AA.AA.AA.in-addr.arpa	IN	PTR	host.domain
	BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB:BBBB.in-addr.arpa	IN	PTR	host.domain

If it is not possible to change the system host name database, create the hosts file local to the LSF system, and configure entries for the multi-homed hosts only. Host names and addresses not found in the hosts file are looked up in the standard name system on your host.

Using IPv6 addresses

About this task

IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format. You can use IPv6 addresses if you define the parameter `LSF_ENABLE_SUPPORT_IPV6=Y` in `lsf.conf`; you do not have to map IPv4 addresses to an IPv6 format.

Enabling both IPv4 and IPv6 support

Procedure

Configure the parameter `LSF_ENABLE_SUPPORT_IPV6=Y` in `lsf.conf`.

Configuring hosts for IPv6

About this task

Follow the steps in this procedure if you do not have an IPv6-enabled DNS server or an IPv6-enabled router. IPv6 is supported on some linux2.4 kernels and on all linux2.6 kernels.

Procedure

1. Configure the kernel.
 - a. Check that the entry `/proc/net/if_inet6` exists.
 - b. If it does not exist, as root run: `modprobe ipv6`
 - c. To check that the module loaded correctly, execute the command `lsmod | grep -w 'ipv6'`
2. Add an IPv6 address to the host by executing the following command as root: `/sbin/ifconfig eth0 inet6 add 3ffe:ffff:0:f101::2/64`

3. Display the IPv6 address using **ifconfig**.
4. Repeat all steps for other hosts in the cluster.
5. Add the addresses for all IPv6 hosts to `/etc/hosts` on each host.
Note: For IPv6 networking, hosts must be on the same subnet.
6. Test IPv6 communication between hosts using the command **ping6**.

Specifying host names with condensed notation

About this task

A number of commands often require you to specify host names. You can now specify host name ranges instead. You can use condensed notation with any commands that use the `-m` option or a host list to specify multiple host names, including the following commands:

- **bacct**
- **bhist**
- **bhost**
- **bjobs**
- **bkill**
- **blaunch**
- **blimits**
- **bmig**
- **bmod**
- **bpeek**
- **brestart**
- **brresume**
- **brsvadd**
- **brsvmod**
- **brsvs**
- **brun**
- **bstop**
- **bsub**
- **bswitch**
- **lsgrun**
- **lshosts**
- **lsload**

You must specify a valid range of hosts, where the start number is smaller than the end number.

Procedure

- Run the command you want and specify the host names as a range.
Use square brackets (`[]`) to enclose the multiple numbers, and use a hyphen (`-`) or colon (`:`) to specify a range of numbers. You can use multiple sets of square brackets in a host name.

For example:

- `bsub -m "host[1-100].example.com"`
The job is submitted to `host1.example.com`, `host2.example.com`, `host3.example.com`, all the way to `host100.example.com`.
- `bsub -m "host[01-03].example.com"`
The job is submitted to `host01.example.com`, `host02.example.com`, and `host03.example.com`.
- `bsub -m "host[5:200].example.com"`
The job is submitted to `host5.example.com`, `host6.example.com`, `host7.example.com`, all the way to `host200.example.com`.

- `bsub -m "host[05:09].example.com"`

The job is submitted to host05.example.com, host06.example.com, all the way to host09.example.com.

- `bsub -m "hostA[1-2]B[1-3].example.com"`

The job is submitted to hostA1B1.example.com, hostA1B2.example.com, hostA1B3.example.com, hostA2B1.example.com, hostA2B2.example.com, and hostA2B3.example.com.

- Run the command you want and specify host names as a combination of ranges and individuals. Use square brackets ([]) to enclose the multiple numbers, and use a hyphen (-) or colon (:) to specify a range of numbers. Use a comma (,) to separate multiple ranges of numbers or to separate individual numbers. You can use multiple sets of square brackets in a host name.

For example:

- `bsub -m "host[1-10,12,20-25].example.com"`

The job is submitted to host1.example.com, host2.example.com, host3.example.com, up to and including host10.example.com. It is also submitted to host12.example.com and the hosts between and including host20.example.com and host25.example.com.

- `bsub -m "host[1:10,20,30:39].example.com"`

The job is submitted to host1.example.com, host2.example.com, host3.example.com, up to and including host10.example.com. It is also submitted to host20.example.com and the hosts between and including host30.example.com and host39.example.com.

- `bsub -m "host[10-20,30,40:50].example.com"`

The job is submitted to host10.example.com, host11.example.com, host12.example.com, up to and including host20.example.com. It is also submitted to host30.example.com and the hosts between and including host40.example.com and host50.example.com.

- `bsub -m "host[01-03,05,07:09].example.com"`

The job is submitted to host01.example.com, up to and including host03.example.com. It is also submitted to host05.example.com, and the hosts between and including host07.example.com and host09.example.com.

- `bsub -m "hostA[1-2]B[1-3,5].example.com"`

The job is submitted to hostA1B1.example.com, hostA1B2.example.com, hostA1B3.example.com, hostA1B5.example.com, hostA2B1.example.com, hostA2B2.example.com, hostA2B3.example.com, and hostA2B5.example.com.

Job directories and data

Jobs use temporary directories for working files and temporary output. By default, IBM® Spectrum LSF uses the default operating system temporary directory. Use the LSF current working directory (CWD) feature to create and manage the job CWD dynamically based on configuration parameters, and any dynamic patterns included in the path. Use the flexible job output directory to create and manage the job output directory dynamically based on configuration parameters.

- [Directory for job output](#)

The output and error options (-o, -oo, -e, and -eo) of the **bsub** and **bmod** commands can accept a file name or directory path. LSF creates the standard output and standard error files in this directory. If you specify only a directory path, job output and error files are created with unique names based on the job ID so that you can use a single directory for all job output, rather than having to create separate output directories for each job.

- [Specifying a directory for job output](#)

- [Temporary job directories](#)

Jobs use temporary directories for working files and temporary output. By default, IBM Spectrum LSF uses the default operating system temporary directory.

- [About flexible job CWD](#)

The Current Working Directory (CWD) feature lets you create and manage the job CWD dynamically based on configuration parameters, and any dynamic patterns included in the path.

- [About flexible job output directory](#)

The flexible job output directory feature lets you create and manage the job output directory dynamically based on configuration parameters.

Directory for job output

The output and error options (-o, -oo, -e, and -eo) of the **bsub** and **bmod** commands can accept a file name or directory path. LSF creates the standard output and standard error files in this directory. If you specify only a directory path, job output and error files are created with unique names based on the job ID so that you can use a single directory for all job output, rather than having to create separate output directories for each job.

Note: The directory path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows.

Specifying a directory for job output

Procedure

Make the final character in the path a slash (/) on UNIX, or a double backslash (\\) on Windows. If you omit the trailing slash or backslash characters, LSF treats the specification as a file name.

If the specified directory does not exist, LSF creates it on the execution host when it creates the standard error and standard output files.

By default, the output files have the following format:

Standard output: output_directory/job_ID.out

Standard error: error_directory/job_ID.err

Example

The following command creates the directory /usr/share/lsf_out if it does not exist, and creates the standard output file *job_ID.out* in this directory when the job completes:

```
bsub -o /usr/share/lsf_out/ myjob
```

The following command creates the directory C:\lsf\work\lsf_err if it does not exist, and creates the standard error file *job_ID.err* in this directory when the job completes:

```
bsub -e C:\lsf\work\lsf_err\\ myjob
```

Temporary job directories

Jobs use temporary directories for working files and temporary output. By default, IBM® Spectrum LSF uses the default operating system temporary directory.

To enable and use temporary directories specific to each job, specify `LSF_TMPDIR=directory_name` in `lsf.conf`.

The name of the job-specific temporary directory has the following format:

- For regular jobs:
 - UNIX: `$LSF_TMPDIR/jobID.tmpdir`
 - Windows: `%LSF_TMPDIR%\jobID.tmpdir`
- For array jobs:
 - UNIX: `$LSF_TMPDIR/arrayID_arrayIndex.tmpdir`
 - Windows: `%LSF_TMPDIR%\arrayID_arrayIndex.tmpdir`

IBM Spectrum LSF can assign the value of the job-specific temporary directory to the `TMPDIR` environment variable, or to a custom environment variable. This allows user applications to use the job-specific temporary directory for each job. To assign

the value of the job-specific temporary directory, specify `LSB_SET_TMPDIR=y` in `lsf.conf`. To assign the value of the job-specific temporary directory to a custom environment variable, specify `LSB_SET_TMPDIR=env_var_name` in `lsf.conf`.

About flexible job CWD

The Current Working Directory (CWD) feature lets you create and manage the job CWD dynamically based on configuration parameters, and any dynamic patterns included in the path.

This feature is useful if you are running applications that have specific requirements for job CWD, such as copying data to the directory before the job starts running. The CWD feature ensures that this data will not be overwritten.

The CWD feature can be enabled and controlled through the following configuration parameters:

- **JOB_CWD_TTL** in `lsb.params` and `lsb.applications`: Specifies the time-to-live for the CWD of a job. LSF cleans created CWD directories after a job finishes based on the TTL value.
- **JOB_CWD** in `lsb.applications`: specifies the CWD for the job in the application profile. The path can be absolute or relative to the submission directory. The path can include dynamic directory patterns.
- **DEFAULT_JOB_CWD** in `lsb.params`: Specifies the cluster wide CWD for the job. The path can be absolute or relative to the submission directory. The path can include dynamic patterns.
- **LSB_JOB_CWD** environment variable: Specifies the directory on the execution host from where the job starts.

If the job is submitted with the `-app` option but without the `-cwd` option, and the `LSB_JOB_CWD` parameter is not defined, then the application profile defined in the `JOB_CWD` parameter will be used. If the `JOB_CWD` parameter is not defined in the application profile, then the value of the `DEFAULT_JOB_CWD` parameter is used.

You can also use the **`bsub -cwd`** command option to specify the current working directory. LSF cleans the created CWD based on the time to live value set in the `JOB_CWD_TTL` parameter.

Each specified CWD can be created as unique directory paths by using dynamic patterns. For example:

```
/scratch/%P will be shared for multiple jobs
/scratch/%P/%J_%I is unique
```

LSF creates CWD under the 700 permissions with the ownership of a submission user. If CWD creation fails, the `/tmp` directory is used. If the CWD path includes the user home directory and if it is not accessible on the execution host, it is replaced with the execution user home directory. If that directory is also not accessible, then `/tmp` is used.

When deleting a directory, LSF deletes only the last directory of the path which was created for the job. If that directory is shared by multiple jobs, data for other jobs may be lost. Therefore, it is recommended not to have shared CWD with enabled TTL.

If CWD was created for the job and then the **`brequeue`** command or the **`bmig`** command was run on the job, LSF will not delete CWD. For parallel jobs run with the **`blaunch`** command, LSF creates CWD only for the execution host and assumes that they are using a shared file system.

About flexible job output directory

The flexible job output directory feature lets you create and manage the job output directory dynamically based on configuration parameters.

This feature is useful if you are running applications that have specific requirements for job output directory, such as copying data to the directory after the job finishes. This feature ensures that this data will not be overwritten.

A job output directory can be specified through the `DEFAULT_JOB_OUTDIR` configuration parameter in the `lsb.params` file. The directory path can be absolute or relative to the submission directory and can include dynamic patterns. Once specified, the system creates the directory at the start of the job on the submission host and uses the new directory. The directory also applies to jobs that are check-pointed, migrated, re-queued or rerun.

LSF checks the directories from the beginning of the path. If a directory does not exist, the system tries to create that directory. If it fails to create that directory, then the system deletes all created directories and uses the submission directory for output.

LSF creates job output directory under the 700 permissions with the ownership of a submission user.

You can also use the **bsub -outdir output_directory** command to create the job output directory. The **-outdir** option supports dynamic patterns for the output directory. The job output directory specified with this command option, or specified in the `DEFAULT_JOB_OUTDIR` parameter, also applies when using the **bsub -f** command to copy files between the local (submission) host and the remote (execution) host.

The following assumptions and dependencies apply to the **-outdir** command option:

- The execution host has access to the submission host.
- The submission host should be running RES or it will use EGO_RSH to run a directory creation command. If this parameter is not defined, rsh will be used. RES should be running on the Windows submission host in order to create the output directory.

Job notification

By default, when a batch job completes or exits, LSF sends a job report by email to the submitting user account.

The job email report includes the following information:

- Standard output (stdout) of the job
- Standard error (stderr) of the job
- LSF job information such as CPU, process, and memory usage

The output from stdout and stderr are merged together in the order printed, as if the job was run interactively. The default standard input (stdin) file is the null device. The null device on UNIX is `/dev/null`.

Enable the `LSB_POSTEXEC_SEND_MAIL` parameter in the `lsf.conf` file to have LSF send a second email to the user that provides details of the post execution, if any. This includes any applicable output.

bsub notification options

-B

Sends email to the job submitter when the job is dispatched and begins running. The default destination for email is defined by the `LSB_MAILTO` parameter in the `lsf.conf` file.

-u user_name

If you want mail sent to another user, use the **-u user_name** option to the **bsub** command. Mail associated with the job will be sent to the named user instead of to the submitting user account.

-notify

If you want to be notified when the job reaches any of the specified states (`exit`, `done`, `start`, or `suspend`), use the **-notify** option. Use a space to separate multiple job states.

Note: Use this option with other integrations to handle notifications.

-N

If you want to separate the job report information from the job output, use the **-N** option to specify that the job report information should be sent by email.

-Ne

If you want the separate job report information to be sent only on a job error, use the **-Ne** option to specify that the job report information should be sent by email when the job exits.

Users can set the environment variable `LSB_JOB_REPORT_MAIL=N` at job submission to disable email notification. Users can also set the environment variable `LSB_JOB_REPORT_MAIL=ERROR` at job submission to ensure that job report information is sent only on a job error (same as the **-Ne** option).

Output and error file options (-o output_file, -e error_file, -oo output_file, and -eo error_file)

The output file created by the **-o** and **-oo** options to the **bsub** command normally contains job report information as well as the job output. This information includes the submitting user and host, the execution host, the CPU time (user plus system time)

used by the job, and the exit status.

If you specify a `-o output_file` or `-oo output_file` option and do not specify a `-e error_file` or `-eo error_file` option, the standard output and standard error are merged and stored in *output_file*. You can also specify the standard input file if the job needs to read input from stdin.

Note:

The file path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory, file name, and expanded values for `%J` (*job_ID*) and `%I` (*index_ID*).

The output files specified by the output and error file options are created on the execution host.

- [Disabling job email](#)
- [Size of job email](#)

Disabling job email

Procedure

- specify stdout and stderr as the files for the output and error options (`-o`, `-oo`, `-e`, and `-eo`).
For example, the following command directs stderr and stdout to file named `/tmp/job_out`, and no email is sent.

```
bsub -o /tmp/job_out sleep 5
```

- On UNIX, for no job output or email specify `/dev/null` as the output file:

```
bsub -o /dev/null sleep 5
```

Results

The following example submits `myjob` to the night queue:

```
bsub -q night -i job_in -o job_out -e job_err myjob
```

The job reads its input from file `job_in`. Standard output is stored in file `job_out`, and standard error is stored in file `job_err`.

By default, LSF sends email to users when their jobs finish. It may not be desirable to receive email after submitting a lot of jobs, and it may be difficult to change job scripts with short notice, especially if those job scripts are shared between users who want email and users who don't. Therefore, LSF provides a simple way to disable the sending of job level email notification from the cluster. When the administrator sets `LSB_JOB_REPORT_MAIL` in `lsf.conf`, email notification for all jobs is disabled. All **sbatchds** must be restarted on all hosts. However, end users can set the value for `LSB_JOB_REPORT_MAIL` in the job submission environment to disable email notification for only that particular job and not email for all jobs. In this case, there is no need to restart **sbatchd**.

If you define `LSB_JOB_REPORT_MAIL` as `N`, no mail will be sent by **sbatchd** and it doesn't affect email sent by **mbatchd**. It also means you do not have to change your job script.

When defining `LSB_JOB_REPORT_MAIL`, note the following:

- **esub**: If you submit a job using **bsub -a xxx** and don't want **sbatchd** to send email, you can set `LSB_JOB_REPORT_MAIL=N|n` before submitting the job. You can also change this parameter's value using `LSB_SUB_MODIFY_ENVFILE` in the **esub** script. However, when using **bmod** with **esub**, you cannot change the value of this parameter even if you use `LSB_SUB_MODIFY_ENVFILE` in the **esub** script.
- **Chunk job**: After the job is done, the submitter or mail user will receive email from **sbatchd**. If you set `LSB_JOB_REPORT_MAIL=N|n` before submitting the job, no email will be sent by **sbatchd**.
- **MultiCluster**: When a job is forwarded from the sending cluster to the execution cluster, **sbatchd** in the execution cluster sends email to the job's submitter or mail user. If you set `LSB_JOB_REPORT_MAIL=N|n` before submitting the job, no email will be sent by the execution cluster's **sbatchd**.

- Job re-run: When a job is scheduled to rerun on another host, **sbatchd** will send the email to the submitter or mail user. If you set `LSB_JOB_REPORT_MAIL=N` before submitting job, no email will be sent. If you change the value of `LSB_JOB_REPORT_MAIL` before rerunning the job, the new value will not affect **sbatchd**.
- Checkpoint job restart: If you set `LSB_JOB_REPORT_MAIL=N` before submitting a checkpoint job, no email will be sent by **sbatchd** when the job is done. If you want to restart the checkpoint job and don't want **sbatchd** to send email, set `LSB_JOB_REPORT_MAIL=N` before restarting the job.
- Pre-execution specified during job submission or in CLI: If you submit a job using **bsub -E pre-exec**, **sbatchd** will send an email to the job's submitter or mail user when the job is done. If you don't want **sbatchd** to send email, set `LSB_JOB_REPORT_MAIL=N` before submitting the job. If you change the value of `LSB_JOB_REPORT_MAIL` in the pre-execution script, the new value will not affect **sbatchd**'s sending mail action on the execution host.
- Pre-execution or job-starter at the queue level: If you submit a job using **bsub -q queueName**, **sbatchd** will send email to the job's submitter or mail user when the job is done. If you don't want **sbatchd** to send email, set `LSB_JOB_REPORT_MAIL=N` before submitting the job. If you change the value of `LSB_JOB_REPORT_MAIL` in the pre-execution or job-starter script, the new value will not affect **sbatchd**'s sending mail action on the execution host.

Size of job email

Some batch jobs can create large amounts of output. To prevent large job output files from interfering with your mail system, you can use the `LSB_MAILSIZE_LIMIT` parameter in `lsf.conf` to limit the size of the email containing the job output information.

By default, `LSB_MAILSIZE_LIMIT` is not enabled: no limit is set on size of batch job output email.

If the size of the job output email exceeds `LSB_MAILSIZE_LIMIT`, the output is saved to a file under `JOB_SPOOL_DIR`, or the default job output directory if `JOB_SPOOL_DIR` is undefined. The email informs users where the job output is located.

If the `-o` or `-oo` option of **bsub** is used, the size of the job output is not checked against `LSB_MAILSIZE_LIMIT`.

LSB_MAILSIZE environment variable

LSF sets `LSB_MAILSIZE` to the approximate size in KB of the email containing job output information, allowing a custom mail program to intercept output that is larger than desired. If you use the `LSB_MAILPROG` parameter to specify the custom mail program that can make use of the `LSB_MAILSIZE` environment variable, it is not necessary to configure `LSB_MAILSIZE_LIMIT`.

`LSB_MAILSIZE` is not recognized by the LSF default mail program. To prevent large job output files from interfering with your mail system, use `LSB_MAILSIZE_LIMIT` to explicitly set the maximum size in KB of the email containing the job information.

LSB_MAILSIZE values

The `LSB_MAILSIZE` environment variable can take the following values:

- A positive integer: if the output is being sent by email, `LSB_MAILSIZE` is set to the estimated mail size in KB.
- `-1`: if the output fails or cannot be read, `LSB_MAILSIZE` is set to `-1`, and the output is sent by email using `LSB_MAILPROG` if specified in `lsf.conf`.
- Undefined: If you use the output or error options (**-o**, **-oo**, **-e**, or **-eo**) of **bsub**, the output is redirected to an output file. Because the output is not sent by email in this case, `LSB_MAILSIZE` is not used and `LSB_MAILPROG` is not called.

If the `-N` option is used with the output or error options of **bsub**, `LSB_MAILSIZE` is not set.

Monitoring IBM Spectrum LSF cluster operations and health

Learn how to monitor cluster performance, job resource usage, and other information about queues, jobs, and users.

- [Monitor cluster performance](#)
Use **badadmin perfmon** to monitor cluster performance. Use **badadmin diagnose** to troubleshoot cluster problems.
- [Monitor job information](#)
Use **bjobs** and **bhist** to monitor the current and past status of jobs and job arrays. The **bjdepinfo** command displays any dependencies that jobs have, either jobs that depend on a job or jobs that your job depends on. View suspending conditions using **bhosts -l** and **bqueues -l**. Run **bjobs -lp** to see the reason a job was suspended. Run **bjobs -l** to see the scheduling thresholds that control when a job is resumed display.
- [Monitor applications by using external scripts](#)
Use the watchdog feature to regularly run external scripts that check application information and to pass on the job information as notifications.
- [View information about resources](#)
Use the **bhosts** command to view information about shared resources on hosts and load on a host. Use the **bjobs** command to see job resource usage. Use the **lsinfo** command to see overall cluster resources, and the **lshosts** command to see host-based resources and host load by resource.
- [View user and user group information](#)
Use the **busers** and **bugroup** commands to display information about LSF users and user groups.
- [View queue information](#)
The **bqueues** command displays information about queues. The **bqueues -l** option also gives current statistics about the jobs in a particular queue, such as the total number of jobs in the queue, the number of running and suspended jobs.

Monitor cluster performance

Use **badadmin perfmon** to monitor cluster performance. Use **badadmin diagnose** to troubleshoot cluster problems.

- [Monitor performance metrics in real time](#)
Enable performance metric collection, tune the metric sampling period, and use **badadmin perfmon view** to display current performance.
- [Enabling daemon log files for diagnostics](#)
LSF provides daemon monitoring mechanisms for **mbatchd** and, starting in Fix Pack 14, for **lsfproxyd**, to log information and help administer or support diagnosing problems with clusters when using these daemons. The LSF rate limiter is managed by the **lsfproxyd** daemon.
- [Diagnose scheduler buckets](#)
- [Monitor scheduler efficiency and overhead](#)
Use the **bacct** or **badadmin perfmon view** commands to monitor scheduler efficiency.

Monitor performance metrics in real time

Enable performance metric collection, tune the metric sampling period, and use **badadmin perfmon view** to display current performance.

Enable metric collection

Set the `SCHED_METRIC_ENABLE=Y` parameter in the `lsb.params` file to enable performance metric collection.

Start performance metric collection dynamically:

```
badadmin perfmon start sample_period
```

Optionally, you can set a sampling period, in seconds. If no sample period is specified, the default sample period set in the `SCHED_METRIC_SAMPLE_PERIOD` parameter in the `lsb.params` file is used.

Stop sampling:

```
badadmin perfmon stop
```

SCHED_METRIC_ENABLE and SCHED_METRIC_SAMPLE_PERIOD can be specified independently. That is, you can specify SCHED_METRIC_SAMPLE_PERIOD and not specify SCHED_METRIC_ENABLE. In this case, when you turn on the feature dynamically (using **badadmin perfmon start**), the sampling period valued defined in SCHED_METRIC_SAMPLE_PERIOD will be used.

badadmin perfmon start and **badadmin perfmon stop** override the configuration setting in lsb.params. Even if SCHED_METRIC_ENABLE is set, if you run **badadmin perfmon start**, performance metric collection is started. If you run **badadmin perfmon stop**, performance metric collection is stopped.

Tune the metric sampling period

Set SCHED_METRIC_SAMPLE_PERIOD in lsb.params to specify an initial cluster-wide performance metric sampling period.

Set a new sampling period in seconds:

```
badadmin perfmon setperiod sample_period
```

Collecting and recording performance metric data may affect the performance of LSF. Smaller sampling periods will result in the lsb.streams file growing faster.

Display current performance

Use the **badadmin perfmon view** command to view real-time performance metric information.

The following metrics are collected and recorded in each sample period:

- The number of queries handled by **mbatchd**
- The number of queries for each of jobs, queues, and hosts. (**bjobs**, **bqueues**, and **bhosts**, as well as other daemon requests)
- The number of jobs submitted (divided into job submission requests and jobs actually submitted)
- The number of jobs dispatched
- The number of jobs reordered, that is, the number of jobs that reused the resource allocation of a finished job (RELAX_JOB_DISPATCH_ORDER is enabled in lsb.params or lsb.queues)
- The number of jobs completed
- The number of jobs sent to remote cluster
- The number of jobs accepted from remote cluster
- Scheduler performance metrics:
 - A shorter scheduling interval means the job is scheduled more quickly
 - Number of different resource requirement patterns for jobs in use which may lead to different candidate host groups. The more matching hosts required, the longer it takes to find them, which means a longer scheduling session. The complexity increases with the number of hosts in the cluster.
 - Number of scheduler buckets in which jobs are put based on resource requirements and different scheduling policies. More scheduler buckets means a longer scheduling session.

```
badadmin perfmon view
```

```
Performance monitor start time: Fri Jan 19 15:07:54
```

```
End time of last sample period: Fri Jan 19 15:25:55
```

```
Sample period : 60 Seconds
```

Metrics	Last	Max	Min	Avg	Total
Processed requests: mbatchd	0	25	0	8	159
Jobs information queries	0	13	0	2	46
Hosts information queries	0	0	0	0	0
Queue information queries	0	0	0	0	0
Job submission requests	0	10	0	0	10
Jobs submitted	0	100	0	5	100
Jobs dispatched	0	0	0	0	0
Jobs reordered	0	0	0	0	0
Jobs completed	0	13	0	5	100
Jobs sent to remote cluster	0	12	0	5	100
Jobs accepted from remote cluster	0	0	0	0	0

File Descriptor Metrics	Free		Used	Total	

MBD file descriptor usage		800	424	1024
Scheduler Metrics	Last	Max	Min	Avg
Scheduling interval in seconds(s)	5	12	5	8
Host matching criteria	5	5	0	5
Job buckets	5	5	0	5

Scheduler metrics are collected at the end of each scheduling session.

Performance metrics information is calculated at the end of each sampling period. Running **badadmin perfmon view** before the end of the sampling period displays metric data collected from the sampling start time to the end of last sample period.

If no metrics have been collected because the first sampling period has not yet ended, **badadmin perfmon view** displays:

```
badadmin perfmon view
Performance monitor start time: Thu Jan 25 22:11:12
End time of last sample period: Thu Jan 25 22:11:12
Sample period : 120 Seconds
-----
No performance metric data available. Please wait until first sample period ends.
```

badadmin perfmon output

Sample Period

Current sample period

Performance monitor start time

The start time of sampling

End time of last sample period

The end time of last sampling period

Metric

The name of metrics

Total

This is accumulated metric counter value for each metric. It is counted from Performance monitor start time to End time of last sample period.

Last Period

Last sampling value of metric. It is calculated per sampling period. It is represented as the metric value per period, and normalized by the following formula:

$\text{LastPeriod} = (\text{Metric Counter Value of Last Period} / \text{Sample Period Interval}) * \text{Sample Period}$

Max

Maximum sampling value of metric. It is reevaluated in each sampling period by comparing Max and Last Period. It is represented as the metric value per period.

Min

Minimum sampling value of metric. It is reevaluated in each sampling period by comparing Min and Last Period. It is represented as the metric value per period.

Avg

Average sampling value of metric. It is recalculated in each sampling period. It is represented as the metric value per period, and normalized by the following formula:

$\text{Avg} = (\text{Total} / (\text{Last PeriodEndTime} - \text{SamplingStartTime})) * \text{Sample Period}$

Reconfigure your cluster with performance metric sampling enabled

If performance metric sampling is enabled dynamically with **badadmin perfmon start**, you must enable it again after running **badadmin mbdrestart**.

- If performance metric sampling is enabled by default, `StartTime` will be reset to the point **mbatchd** is restarted.
- Use the **badadmin mbdrestart** command when the `SCHED_METRIC_ENABLE` and `SCHED_METRIC_SAMPLE_PERIOD` parameters are changed. The **badadmin reconfig** command is the same as the **badadmin mbdrestart** command in this context.

Performance metric logging in lsb.streams

By default, collected metrics are written to lsb.streams.

However, performance metric can still be turned on even if `ENABLE_EVENT_STREAM=N` is defined. In this case, no metric data will be logged.

- If `EVENT_STREAM_FILE` is defined and is valid, collected metrics should be written to `EVENT_STREAM_FILE`.
- If `ENABLE_EVENT_STREAM=N` is defined, metrics data will not be logged.

Job arrays and job packs

Every job submitted in a job array or job pack is counted individually, except for the `Job submission requests` metric.

The entire job array or job pack counts as just one job submission request.

Job rerun

Job rerun occurs when execution hosts become unavailable while a job is running, and the job will be put to its original queue first and later will be dispatched when a suitable host is available.

In this case, only one submission request, one job submitted, and n jobs dispatched, n jobs completed are counted (n represents the number of times the job reruns before it finishes successfully).

Job queue

Requeued jobs may be dispatched, run, and exit due to some special errors again and again. The job data always exists in the memory, so LSF only counts one job submission request and one job submitted, and counts more than one job dispatched.

For jobs completed, if a job is requeued with **brequeue**, LSF counts two jobs completed, since requeuing a job first kills the job and later puts the job into pending list. If the job is automatically requeued, LSF counts one job completed when the job finishes successfully.

Job replay

When job replay is finished, submitted jobs are not counted in job submission and job submitted, but are counted in job dispatched and job finished.

Enabling daemon log files for diagnostics

LSF provides daemon monitoring mechanisms for **mbatchd** and, starting in Fix Pack 14, for **lsfproxyd**, to log information and help administer or support diagnosing problems with clusters when using these daemons. The LSF rate limiter is managed by the **lsfproxyd** daemon.

About this task

The **mbatchd** daemon log file is called `query_info.querylog.hostname`. The log files show information about **mbatchd** requests: who issued these requests, where the requests came from, and the data size of the query. The format of the information provided in the log file is **DATE TIME YEAR COMMAND,USER,HOSTNAME,SIZE,OPTION**.

The **lsfproxyd** daemon log file for the LSF rate limiter, and is called `query_info.queryproxylog.hostname`. The log files show information about **lsfproxyd** requests: where the requests came from, the data size of the request, the batch operation code, whether the request was rejected or accepted, and the time that the **lsfproxyd** daemon receives and processes the requests. The format of the information provided in the log file is **DATE TIME YEAR BATCH_OPCODE,USER,HOSTNAME,SIZE,ACCEPT,RECEIVE TIME,PROCESS TIME**.

There are two ways to enable daemon log files: statically and dynamically.

Procedure

- Statically:
 - For the **mbatchd** daemon log file, set [ENABLE_DIAGNOSE=query](#) in the lsb.params configuration file.
 - For the **lsfproxyd** daemon log file, set the `ENABLE_DIAGNOSE=lsfproxyd` in the lsb.params configuration file.
 - For both the **mbatchd** daemon and **lsfproxyd** daemon log files, set `ENABLE_DIAGNOSE="query lsfproxyd"` in the lsb.params configuration file.

The log files save to the default LSF log directory (LSF_LOGDIR). To change this log location, defined the [DIAGNOSE_LOGDIR](#) parameter in the lsb.params configuration file.

- Dynamically:
 - For the **mbatchd** daemon log file, run the **badmin diagnose -c query** command.
 - For the **lsfproxyd** daemon log file, run the **badmin diagnose -c lsfproxyd** command.

The dynamic method overrides the static settings. However, if you restart or reconfigure **mbatchd** or **lsfproxyd**, it switches back to the static diagnosis settings.

Related concepts

- [Offloading the mbatchd daemon using the LSF rate limiter \(lsfproxyd daemon\)](#)
- [daemons](#)

Related reference

- [ENABLE_DIAGNOSE](#)
- [DIAGNOSE_LOGDIR](#)

Diagnose scheduler buckets

LSF provides the ability to save a snapshot of the current contents of the scheduler buckets to help administrators diagnose problems with the scheduler. Jobs are put into scheduler buckets based on resource requirements and different scheduling policies. Saving the contents into a snapshot file is useful for data analysis by parsing the file or by performing a simple text search on its contents.

This feature is helpful if there is a sudden large performance impact on the scheduler that you want to examine. Use the snapshot file to identify any users with a large number of buckets or large attribute values.

To use this feature, run the **badmin diagnose -c jobreq** command.

This feature enables **mbschd** to write an active image of the scheduler job buckets into a snapshot file as raw data in XML or JSON format. There can be a maximum of one snapshot file generated in each scheduling cycle.

Use the `-f` option to specify a custom file name and path and the `-t` option to specify whether the file is in XML or JSON format.

By default, the name of the snapshot file is `jobreq_<hostname>_<dateandtime>.<format>`, where `<format>` is `xml` or `json`, depending on the specified format of the snapshot file. By default, the snapshot file is saved to the location specified in the `DIAGNOSE_LOGDIR` parameter.

Monitor scheduler efficiency and overhead

Use the **bacct** or **badmin perfmon view** commands to monitor scheduler efficiency.

When the amount of time that LSF spent scheduling a job is large compared to the run times of jobs, you will observe a low resource utilization in your cluster. For instance, if the average run time of jobs equals the average time required to fill a slot after a job finishes, the slot usage in the cluster will be approximately 50% of what it would be if scheduling overhead is zero.

It is not always clear whether low utilization is caused by scheduling performance or by configured policies (such as limits) that block jobs from accessing resources.

LSF has a scheduling efficiency metric in the **badmin perfmon** command that estimates how the slot and memory utilization of the cluster is affected by scheduling overhead. A value near 100% means that improving scheduler performance does not significantly improve resource utilization, while a lower percentage indicates how improving scheduler performance will improve resource utilization. For example, a value of 75% means that due to scheduling overhead, resource utilization is only 75% of what it could be if scheduling overhead were to be reduced to zero.

Run the **badmin perfmon view** command to view the scheduler efficiency for finished jobs within a sample period. This displays the scheduler efficiency numbers for both the set of finished jobs within the sample period and all finished jobs in the cluster.

Run the **bacct** command to view the scheduler efficiency for all finished jobs in a cluster.

Monitor job information

Use **bjobs** and **bhist** to monitor the current and past status of jobs and job arrays. The **bjdepinfo** command displays any dependencies that jobs have, either jobs that depend on a job or jobs that your job depends on. View suspending conditions using **bhosts -l** and **bqueues -l**. Run **bjobs -lp** to see the reason a job was suspended. Run **bjobs -l** to see the scheduling thresholds that control when a job is resumed display.

- [Viewing host-level and queue-level suspending conditions](#)
- [Viewing job-level suspending conditions](#)
- [Viewing resume thresholds](#)
- [View job priority information](#)
- [Viewing job dependencies](#)
- [View information about backfill jobs](#)
Monitor how resources are distributed in fair share scheduling policies.
- [Viewing job array information](#)
Use the **bjobs** and **bhist** commands to monitor the current and past status of job arrays.
- [View information about reserved job slots](#)

Viewing host-level and queue-level suspending conditions

Procedure

View suspending conditions using **bhosts -l** and **bqueues -l**.

Viewing job-level suspending conditions

About this task

The thresholds that apply to a particular job are the more restrictive of the host and queue thresholds.

Procedure

Run **bjobs -l**.

Viewing resume thresholds

Procedure

Run **bjobs -l**.

The scheduling thresholds that control when a job is resumed display.

View job priority information

Procedure

Use the following commands to view job history, the current status and system configurations:

- **bhist -l job_ID**

Displays the history of a job including changes in job priority.

- **bjobs -l [job_ID]**

Displays the current job priority and the job priority at submission time. Job priorities are changed by the job owner, LSF, and queue administrators, and automatically when automatic job priority escalation is enabled.

- **bparams -l**

Displays values for:

- The maximum user priority, MAX_USER_PRIORITY
- The default submission priority, MAX_USER_PRIORITY/2
- The value and frequency used for automatic job priority escalation, JOB_PRIORITY_OVER_TIME

Viewing job dependencies

About this task

The **bjdepinfo** command displays any dependencies that jobs have, either jobs that depend on a job or jobs that your job depends on.

By specifying **-r**, you get not only direct dependencies (job A depends on job B), but also indirect dependencies (job A depends on job B, job B depends on jobs C and D). You can also limit the number of levels returned using the **-l** option.

The **-l** option displays results in greater detail.

Procedure

- To display all jobs that this job depends on:

```
bjdepinfo 123
```

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
123	32522	RUN	JOB32522	1

- To display jobs that depend on a job, you specify (display child jobs):

```
bjdepinfo -c 300
```

JOBID	CHILD	CHILD_STATUS	CHILD_NAME	LEVEL
300	310	PEND	JOB310	1
300	311	PEND	JOB311	1

300	312	PEND	JOB312	1
-----	-----	------	--------	---

- To display the parent jobs that cause a job to pend:
bjdepinfo -p 100

These jobs are always pending because their dependency has not yet been satisfied.

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
100	99	PEND	JOB99	1
100	98	PEND	JOB98	1
100	97	PEND	JOB97	1
100	30	PEND	JOB30	1

- Display more information about job dependencies including whether the condition has been satisfied or not and the condition that is on the job:

```
bjdepinfo -l 32522
```

Dependency condition of job <32522> is not satisfied: done(23455)

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
32522	23455	RUN	JOB23455	1

- Display information about job dependencies that includes only direct dependencies and two levels of indirect dependencies:

```
bjdepinfo -r 3 -l 100
```

Dependency condition of job <100> is not satisfied: done(99) && ended(98) && done(97) && done(96)

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
100	99	PEND	JOB99	1
100	98	PEND	JOB98	1
100	97	PEND	JOB97	1
100	96	DONE	JOB96	1

Dependency condition of job <97> is not satisfied: done(89)

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
97	89	PEND	JOB89	2

Dependency condition of job <89> is not satisfied: ended(86)

JOBID	PARENT	PARENT_STATUS	PARENT_NAME	LEVEL
89	86	PEND	JOB86	3

View information about backfill jobs

Monitor how resources are distributed in fair share scheduling policies.

- [Viewing information about job start time](#)
- [Viewing the run limits for interruptible backfill jobs \(bjobs and bhist\)](#)
- [Displaying available slots for backfill jobs](#)

Viewing information about job start time

Procedure

Use **bjobs -l** to view the estimated start time of a job.

Viewing the run limits for interruptible backfill jobs (bjobs and bhist)

Procedure

1. Use **bjobs** to display the run limit calculated based on the configured queue-level run limit.
For example, the interruptible backfill queue *lazy* configures RUNLIMIT=60:

```
bjobs -l 135
Job <135>, User <user1>, Project <default>, Status <RUN>, Queue <lazy>, Command
<myjob>
Mon Nov 21 11:49:22 2009: Submitted from host <hostA>, CWD <${HOME}/H
PC/jobs>;
RUNLIMIT
59.5 min of hostA
Mon Nov 21 11:49:26 2009: Started on <hostA>, Execution Home </home
/user1>, Execution CWD </home/user1/HPC/jobs>;
...
```

2. Use **bhist** to display job-level run limit if specified.
For example, job 135 was submitted with a run limit of 3 hours:

```
bsub -n 1 -q lazy -W 3:0 myjob
Job <135> is submitted to queue <lazy>.
```

bhist displays the job-level run limit:

```
bhist -l 135
Job <135>, User <user1>, Project <default>, Command <myjob>
Mon Nov 21 11:49:22 2009: Submitted from host <hostA>, to Queue <lazy>, CWD
<${HOME}/HPC/jobs>;
RUNLIMIT
180.0 min of hostA
Mon Nov 21 11:49:26 2009: Dispatched to <hostA>;
Mon Nov 21 11:49:26 2009: Starting (Pid 2746);
Mon Nov 21 11:49:27 2009: Interruptible backfill runtime limit is 59.5 minutes;
Mon Nov 21 11:49:27 2009: Running with execution home </home/user1>, Execution CWD
...

```

Displaying available slots for backfill jobs

The **bslots** command displays slots reserved for parallel jobs and advance reservations. The available slots are not currently used for running jobs, and can be used for backfill jobs. The available slots displayed by **bslots** are only a snapshot of the slots currently not in use by parallel jobs or advance reservations. They are not guaranteed to be available at job submission.

By default, **bslots** displays all available slots, and the available run time for those slots. When no reserved slots are available for backfill, **bslots** displays "No reserved slots available."

The backfill window calculation is based on the snapshot information (current running jobs, slot reservations, advance reservations) obtained from **mbatchd**. The backfill window displayed can serve as reference for submitting backfill-able jobs. However, if you have specified extra resource requirements or special submission options, it does not insure that submitted jobs are scheduled and dispatched successfully.

bslots -R only supports the select resource requirement string. Other resource requirement selections are not supported.

If the available backfill window has no run time limit, its length is displayed as UNLIMITED.

Examples

Display all available slots for backfill jobs:

```
bslots
```

SLOTS RUNTIME

1 UNLIMITED

3 1 hour 30 minutes

5 1 hour 0 minutes

7 45 minutes

15 40 minutes

18 30 minutes

20 20 minutes

Display available slots for backfill jobs requiring 15 slots or more:

```
bslots -n 15
```

SLOTS RUNTIME

15 40 minutes

18 30 minutes

20 20 minutes

Display available slots for backfill jobs requiring a run time of 30 minutes or more:

```
bslots -W 30
```

SLOTS RUNTIME

3 1 hour 30 minutes

5 1 hour 0 minutes

7 45 minutes

15 40 minutes

18 30 minutes

```
bslots -W 2:45
```

No reserved slots available.

```
bslots -n 15 -W 30
```

SLOTS RUNTIME

15 40 minutes

18 30 minutes

Display available slots for backfill jobs requiring a host with more than 500 MB of memory:

```
bslots -R "mem>500"
```

SLOTS RUNTIME

7 45 minutes

15 40 minutes

Display the host names with available slots for backfill jobs:

```
bslots -l
```

SLOTS: 15

RUNTIME: 40 minutes

HOSTS: 1*hostB 1*hostE 3*hostC ...

3*hostZ

SLOTS: 15

RUNTIME: 30 minutes

HOSTS: 2*hostA 1*hostB 3*hostC ...

1*hostX

Viewing job array information

Use the **bjobs** and **bhist** commands to monitor the current and past status of job arrays.

Displaying job array status

The -A option of the **bjobs** command shows job array summary information.

Procedure

To display summary information about the currently running jobs submitted from a job array, use the -A option of the **bjobs** command.

For example, a job array of 10 jobs with job ID 123:

```
bjobs -A 123
JOBID   ARRAY_SPEC  OWNER   NJOBS  PEND  DONE  RUN  EXIT  SSUSP  USUSP  PSUSP
123     myArra[1-10]  user1    10     3     3     4     0     0     0     0
```

Displaying job array dependencies

The **bjdepinfo** command shows job dependency information for a job array.

Procedure

To display information for any job dependency information for an array, use the **bjdepinfo** command.

For example, a job array (with job ID 456) where you want to view the dependencies on the third element of the array:

```
bjdepinfo -c "456[3]"
JOBID  CHILD  CHILD_STATUS  CHILD_NAME  LEVEL
456[3] 300    PEND          job300      1
```

Displaying status of jobs submitted from an array

The **bjobs** command displays the status of the individual jobs submitted from a job array

Procedure

Use the **bjobs** command and specify the job array job ID to display the status of the individual jobs submitted from a job array. For jobs submitted from a job array, JOBID displays the job array job ID, and JOBNAME displays the job array name and the index value of each job.

For example, to view a job array with job ID 123:

```
bjobs 123
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
123	user1	DONE	default	hostA	hostC	myArray[1]	Feb 29 12:34
123	user1	DONE	default	hostA	hostQ	myArray[2]	Feb 29 12:34
123	user1	DONE	default	hostA	hostB	myArray[3]	Feb 29 12:34
123	user1	RUN	default	hostA	hostC	myArray[4]	Feb 29 12:34
123	user1	RUN	default	hostA	hostL	myArray[5]	Feb 29 12:34
123	user1	RUN	default	hostA	hostB	myArray[6]	Feb 29 12:34
123	user1	RUN	default	hostA	hostQ	myArray[7]	Feb 29 12:34
123	user1	PEND	default	hostA		myArray[8]	Feb 29 12:34
123	user1	PEND	default	hostA		myArray[9]	Feb 29 12:34
123	user1	PEND	default	hostA		myArray[10]	Feb 29 12:34

Displaying past job status

The **bhist** command displays historical information about array jobs.

Procedure

Use the **bhist** command and specify the job array job ID to display the past status of the individual jobs submitted from a job array.

For example, to view the history of a job array with job ID 456:

```
bhist 456
```

Summary of time in seconds spent in various states:

JOBID	USER	JOB_NAME	PEND	PSUSP	RUN	USUSP	SSUSP	UNKWN	TOTAL
456[1]	user1	*rray[1]	14	0	65	0	0	0	79
456[2]	user1	*rray[2]	74	0	25	0	0	0	99
456[3]	user1	*rray[3]	121	0	26	0	0	0	147
456[4]	user1	*rray[4]	167	0	30	0	0	0	197
456[5]	user1	*rray[5]	214	0	29	0	0	0	243
456[6]	user1	*rray[6]	250	0	35	0	0	0	285
456[7]	user1	*rray[7]	295	0	33	0	0	0	328
456[8]	user1	*rray[8]	339	0	29	0	0	0	368
456[9]	user1	*rray[9]	356	0	26	0	0	0	382
456[10]	user1	*rray[10]	375	0	24	0	0	0	399

Displaying the current status of a specific job

The **bjobs** command shows the current status of a specific array job element.

Procedure

Use the **bjobs** command to display the current status of a specific job submitted from a job array. Specify the job array job ID and an index value in quotes.

For example, the status of the 5th job in a job array with job ID 123:

```
bjobs "123[5]"
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
123	user1	RUN	default	hostA	hostL	myArray[5]	Feb 29 12:34

Displaying the past status of a specific job

The **bhist** command shows the historical status of a specific array job element.

Procedure

Use the **bhist** command to display the past status of a specific job submitted from a job array. Specify the job array job ID and an index value in quotes.

For example, the status of the 5th job in a job array with job ID 456:

```
bhist "456[5]"
Summary of time in seconds spent in various states:
JOBID  USER   JOB_NAME  PEND   PSUSP   RUN    USUSP   SSUSP   UNKWN   TOTAL
456[5] user1  *rray[5]  214    0       29     0       0       0       243
```

View information about reserved job slots

Procedure

Display reserved slots using **bjobs**.

The number of reserved slots can be displayed with the **bqueues**, **bhosts**, **bhpart**, and **busers** commands. Look in the **RSV** column.

- [Viewing configured job slot share](#)
- [Viewing slot allocation of running jobs](#)

Viewing configured job slot share

Procedure

Use **bqueues -l** to show the job slot share (SLOT_SHARE) and the hosts participating in the share pool (SLOT_POOL):

```
QUEUE: queue1
PARAMETERS/STATISTICS
PRIO NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SSUSP USUSP  RSV
50   20  Open:Active        -   -   -   -   0     0     0    0    0    0
Interval for a host to accept two jobs is 0 seconds

STACKLIMIT MEMLIMIT
2048 K      5000 K

SCHEDULING PARAMETERS
          r15s  r1m  r15m  ut      pg    io    ls    it    tmp    swp    mem
loadSched -    -    -    -      -    -    -    -    -    -    -
loadStop  -    -    -    -      -    -    -    -    -    -    -

          cpuspeed  bandwidth
loadSched -          -
loadStop  -          -

USERS:  all users
HOSTS:  groupA/
SLOT_SHARE: 50%
SLOT_POOL: poolA
```

Viewing slot allocation of running jobs

Procedure

Use **bhosts**, **bmgroup**, and **bqueues** to verify how LSF maintains the configured percentage of running jobs in each queue. The queues configurations above use the following hosts groups:

```
bmgroup -r
GROUP_NAME  HOSTS
groupA      hosta hostb hostc
groupB      hostd hoste hostf
```

Each host has a maximum job slot limit of 5, for a total of 15 slots available to be allocated in each group:

bhosts									
HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV	
hosta	ok	-	5	5	5	0	0	0	
hostb	ok	-	5	5	5	0	0	0	
hostc	ok	-	5	5	5	0	0	0	
hostd	ok	-	5	5	5	0	0	0	
hoste	ok	-	5	5	5	0	0	0	
hostf	ok	-	5	5	5	0	0	0	

Pool named `poolA` contains `queue1`, `queue2`, and `queue3`. `poolB` contains `queue4`, `queue5`, and `queue6`. The `bqueues` command shows the number of running jobs in each queue:

bqueues											
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
queue1	50	Open:Active	-	-	-	-	492	484	8	0	
queue2	48	Open:Active	-	-	-	-	500	495	5	0	
queue3	46	Open:Active	-	-	-	-	498	496	2	0	
queue4	44	Open:Active	-	-	-	-	985	980	5	0	
queue5	43	Open:Active	-	-	-	-	985	980	5	0	
queue6	42	Open:Active	-	-	-	-	985	980	5	0	

As a result: `queue1` has a 50% share and can run 8 jobs; `queue2` has a 30% share and can run 5 jobs; `queue3` has a 20% share and is entitled 3 slots, but since the total number of slots available must be 15, it can run 2 jobs; `queue4`, `queue5`, and `queue6` all share 30%, so 5 jobs are running in each queue.

Monitor applications by using external scripts

Use the watchdog feature to regularly run external scripts that check application information and to pass on the job information as notifications.

- [Creating external scripts to monitor applications](#)
Configure the application profile to enable the watchdog feature and specify the LSF Application Center Notifications server to receive notifications.
- [Configuring application profiles to use external scripts](#)
Configure the application profile to use an external watchdog script to monitor the application and specify the LSF Application Center Notifications server to receive notifications.
- [Using the application profiles to run the external scripts](#)
To use the watchdog scripts, submit a job to an application profile that has the WATCHDOG parameter enabled.

Creating external scripts to monitor applications

Configure the application profile to enable the watchdog feature and specify the LSF Application Center Notifications server to receive notifications.

Before you begin

To ensure that the watchdog scripts can send notifications to the LSF Application Center Notifications server, define the `LSF_AC_PNC_URL` parameter in the `lsf.conf` file.

Procedure

1. Create a watchdog script to monitor the application (by checking application data, logs, and other information) and send notification messages.
In the script, use the `bpost -N` command option to send a notification (with the message in the `-d` option and the specified error level) to the LSF Application Center Notifications server:


```
bpost -d "message" -N WARNING | ERROR | CRITICAL | INFO
```

All job environment variables are available to the watchdog scripts. In addition, the following LSF job-level resource consumption environment variables are available to the watchdog scripts:

- LSB_GPU_ALLOC_INFO
- LSB_JOB_AVG_MEM
- LSB_JOB_CPU_TIME
- LSB_JOB_MAX_MEM
- LSB_JOB_MEM
- LSB_JOB_NTHREAD
- LSB_JOB_PGIDS
- LSB_JOB_PIDS
- LSB_JOB_RUN_TIME
- LSB_JOB_SWAP

The watchdog script might have the following format:

```
#!/bin/sh
source <lsf_conf_dir>/profile.lsf
<application_checking_commands>
if <okay> then
    exit 0
else
    if <warning_level> then
        bpost -N WARNING -d "WARNING: <warning_message>"
        exit 0
    else
        bpost -N CRITICAL -d "FATAL: <critical_message>"
        exit 1
    end if
end if
```

Note: You must add a command to source the LSF environment at the beginning of the watchdog script.

2. Set the proper permissions for the script to ensure that the job submission user is able to execute the script.

Related reference

- [LSF_AC_PNC_URL parameter in the lsf.conf file](#)
- [bpost -N command option](#)

Configuring application profiles to use external scripts

Configure the application profile to use an external watchdog script to monitor the application and specify the LSF Application Center Notifications server to receive notifications.

Before you begin

To ensure that the watchdog scripts can send notifications to the LSF Application Center Notifications server, define the LSF_AC_PNC_URL parameter in the lsf.conf file.

Procedure

1. Edit the lsb.applications file and define the WATCHDOG parameter for the application profile that you want to monitor.
`WATCHDOG=script[file/path/to/script] init[init_delay] period[start_interval]`

This parameter uses the following keywords:

script

- Required. This keyword specifies the file path to the external watchdog script to check the application data and other information. This file must have the proper permissions for the job submission user to execute the script.
- init
- Optional. This keyword specifies the delay to start the watchdog script after the job starts, in seconds. Specify a number larger than 30 seconds. The default value is 60 seconds.
- period
- Optional. This keyword specifies the interval in which to start the watchdog script after the previous time that the watchdog script started, in seconds. Specify a number larger than 30 seconds. The default value is 60 seconds.
2. To send notifications to the LSF Application Center Notifications server, edit the `lsf.conf` file and define the `LSF_AC_PNC_URL` parameter with the URL and listen port of the LSF Application Center Notifications server.
`LSF_AC_PNC_URL=HTTP://server_address[:port_number]`
- If the listen port is not specified, the default port number is 8081.

Related reference

- [WATCHDOG parameter in the lsb.applications file](#)
- [LSF_AC_PNC_URL parameter in the lsf.conf file](#)

Using the application profiles to run the external scripts

To use the watchdog scripts, submit a job to an application profile that has the WATCHDOG parameter enabled.

Procedure

1. Use the **bsub -app** command option to submit a job to an application profile that has the watchdog feature enabled. If the WATCHDOG parameter is enabled for the `jobwatch1` application in the `lsb.applications` file, submit a job with the following command:


```
bsub -app jobwatch1
```


LSF dispatches the job and executes the external watchdog script at regular intervals according to the WATCHDOG parameter settings for the specified application profile.
2. Use the **bread -N** command option to see information on any notifications that the watchdog scripts send.

Related reference

- [WATCHDOG parameter in the lsb.applications file](#)
- [bread -N command option](#)

View information about resources

Use the **bhosts** command to view information about shared resources on hosts and load on a host. Use the **bjobs** command to see job resource usage. Use the **lsinfo** command to see overall cluster resources, and the **lshosts** command to see host-based resources and host load by resource.

- [Viewing job-level resource requirements](#)
- [Viewing queue-level resource requirements](#)
- [Viewing shared resources for hosts](#)
- [Viewing load on a host](#)
- [Viewing job resource usage](#)
- [View cluster resources \(lsinfo\)](#)
- [Viewing host resources \(lshosts\)](#)

- [View resource reservation information](#)
- [View information about resource allocation limits](#)
Your job might be pending because some configured resource allocation limits are reached. Use the **blimits** command to show the dynamic counters of resource allocation limits configured in Limit sections in the lsb.resources file. The **blimits** command displays the current resource usage to show what limits might be blocking your job.
- [View application profile information](#)
- [View fair share information](#)
Monitor how resources are distributed in fair share scheduling policies.
- [Viewing information about SLAs and service classes](#)
- [Viewing configured guaranteed resource pools](#)
Resource-type SLAs have the host or slot guarantee configured within the guaranteed resource pool.

Viewing job-level resource requirements

Procedure

1. Use **bjobs -l** to view resource requirements defined for the job:

```
bsub -R "type==any" -q normal myjob
Job <2533> is submitted to queue <normal>.
bjobs -l 2533
Job <2533>, User <user1>, Project <default>, Status <DONE>, Queue <normal>,
  Command <myjob>
Fri May 10 17:21:26 2009: Submitted from host <hostA>, CWD <${HOME}>, Requested
  Resources <{hname=hostB} || {hname=hostC}>;
Fri May 10 17:21:31 2009: Started on <hostB>, Execution Home </home/user1>,
  Execution CWD </home/user1>;
Fri May 10 17:21:47 2009: Done successfully. The CPU time used is 0.3 seconds.
...
```

2. After a job is finished, use **bhist -l** to view resource requirements defined for the job:

```
bhist -l 2533
Job <2533>, User <user1>, Project <default>, Command <myjob>
Fri May 10 17:21:26 2009: Submitted from host <hostA>, to Queue <normal>, CWD
  <${HOME}>, Requested Resources <{hname=hostB} || {hname=hostC}>;
Fri May 10 17:21:31 2009: Dispatched to <hostB>, <Effective RES_REQ <select[
  (hname = hostC ) && (type == any)] order[r15s:pg] >>;
Fri May 10 17:21:32 2009: Starting (Pid 1850232);
Fri May 10 17:21:33 2009: Running with execution home </home/user1>, Execution
  CWD </home/user1>, Execution Pid <1850232>;
Fri May 10 17:21:45 2009: Done successfully. The CPU time used is 0.3 seconds;
...
```

Note: If you submitted a job with multiple select strings using the **bsub** -R option, **bjobs -l** and **bhist -l** display a single, merged select string.

Viewing queue-level resource requirements

Procedure

Use **bqueues -l** to view resource requirements (RES_REQ) defined for the queue:

```
bqueues -l normal
QUEUE: normal
-- No description provided. This is the default queue.
...
RES_REQ: select[type==any]
rusage[mem=10,dynamic_rsrc=10:duration=2:decay=1]
...
```

Viewing shared resources for hosts

Procedure

Run **bhosts -s** to view shared resources for hosts. For example:

```
bhosts -s
RESOURCE      TOTAL    RESERVED  LOCATION
tot_lic        5         0.0    hostA hostB
tot_scratch    00         0.0    hostA hostB
avail_lic       2         3.0    hostA hostB
avail_scratch  100        400.0    hostA hostB
```

The **TOTAL** column displays the value of the resource. For dynamic resources, the **RESERVED** column displays the amount that has been reserved by running jobs.

Viewing load on a host

Procedure

Run **bhosts -l** to check the load levels on the host.

A dash (-) in the output indicates that the particular threshold is not defined.

```
bhosts -l hostB
HOST: hostB
STATUS      CPUF    JL/U    MAX NJOBS  RUN  SSUSP  USUSP  RSV
ok          20.00   2       2    0      0     0     0

CURRENT LOAD USED FOR SCHEDULING:
          r15s   r1m   r15m   ut    pg    io    ls    t    tmp   swp   mem   slots
Total    0.3    0.8   0.9   61%   3.8   72   26    0    6M   253M  97M    8
Reserved 0.0    0.0   0.0   0%    0.0   0    0     0    0M   0M   0M    8

LOAD THRESHOLD USED FOR SCHEDULING:
          r15s   r1m   r15m   ut    pg    io    ls    it   tmp   swp   mem
loadSched -     -     -     -     -     -     -     -     -     -     -
loadStop  -     -     -     -     -     -     -     -     -     -     -

          cpuspeed   bandwidth
loadSched           -           -
loadStop            -           -
```

Viewing job resource usage

Procedure

Run **bjobs -l** to display the current resource usage of the job.

Usage information is sampled by PIM every 30 seconds and collected by **sbatchd** at a maximum frequency of every SBD_SLEEP_TIME (configured in the lsb.params file) and sent to **mbatchd**.

An update occurs only if the value for the CPU time, resident memory usage, or virtual memory usage has changed by more than 10 percent from the previous update, or if a new process or process group has been created. Even if the usage does not change for more than 10%, SBD will still update it if 15 * SBD_SLEEP_TIME passed from last update.

View cluster resources (lsinfo)

Procedure

Use **lsinfo** to list the resources available in your cluster.
The **lsinfo** command lists all resource names and descriptions.

```
lsinfo
RESOURCE_NAME  TYPE      ORDER DESCRIPTION
r15s           Numeric   Inc    15-second CPU run queue length
rlm            Numeric   Inc    1-minute CPU run queue length (alias:cpu)
r15m           Numeric   Inc    15-minute CPU run queue length
ut             Numeric   Inc    1-minute CPU utilization (0.0 to 1.0)
pg             Numeric   Inc    Paging rate (pages/second)
io             Numeric   Inc    Disk IO rate (Kbytes/second)
ls             Numeric   Inc    Number of login sessions (alias: login)
it             Numeric   Dec    Idle time (minutes) (alias: idle)
tmp            Numeric   Dec    Disk space in /tmp (Mbytes)
swp            Numeric   Dec    Available swap space (Mbytes) (alias:swap)
mem            Numeric   Dec    Available memory (Mbytes)
ncpus          Numeric   Dec    Number of CPUs
nprocs         Numeric   Dec    Number of physical processors
ncores         Numeric   Dec    Number of cores per physical processor
nthreads       Numeric   Dec    Number of threads per processor
coredisks      Numeric   Dec    Number of local disks
maxmem         Numeric   Dec    Maximum memory (Mbytes)
maxswp         Numeric   Dec    Maximum swap space (Mbytes)
maxtmp         Numeric   Dec    Maximum /tmp space (Mbytes)
cpuf           Numeric   Dec    CPU factor
...
```

Viewing host resources (lshosts)

Procedure

Run **lshosts** for a list of the resources that are defined on a specific host:

```
lshosts hostA
HOST_NAME      type      model  cpuf  ncpus  maxmem  maxswp  server  RESOURCES
hostA          SOL732    Ultra2  20.2   2     256M    679M    Yes    ()
```

- [Viewing host load by resource \(lshosts -s\)](#)
The **lshosts -s** command shows host load by shared resource
- [Customize host resource information output](#)

Viewing host load by resource (lshosts -s)

The **lshosts -s** command shows host load by shared resource

Procedure

Run **lshosts -s** to view host load for static and dynamic shared resources:
The following **lshosts -s** output shows that the shared scratch directory currently contains 500 MB of space.

```
lshosts -s
RESOURCE      VALUE      LOCATION
```

tot_lic	5	host1 host2
tot_scratch	500	host1 host2

The **VALUE** field indicates the amount of that resource. The **LOCATION** column shows the hosts which share this resource.

Customize host resource information output

By default, the **lshosts** command displays a predefined set of resource information for hosts. While you can use various **lshosts** options to display specific host resource information based on your needs, you can also customize the specific fields that **lshosts** displays. Customize output to create a specific **lshosts** output format that shows all the required information so you can easily parse the information by using custom scripts or to display the information in a predefined format.

Use the **LSF_LSHOSTS_FORMAT** parameter in **lsf.conf** or the **LSF_LSHOSTS_FORMAT** runtime environment variable to define the default **lshosts** output format for LSF:

```
LSF_LSHOSTS_FORMAT="field_name[:-][output_width][:unit]] ... [delimiter='character']"
```

Use the **lshosts -o** option to define the custom output at the command level:

```
lshosts ...-o field_name[:-][output_width][:unit]] ... [delimiter='character']"
```

- Specify which **lshosts** fields (or aliases instead of the full field names), in which order, and with what width to display.
- Specify only the **lshosts** field name or alias to set its output to unlimited width and left justification.
- Specify the colon (:) without a width to set the output width to the recommended width for that field.
- Specify the colon (:) with a width to set the maximum number of characters to display for the field. When its value exceeds this width, **lshosts** truncates the ending characters.
- Specify a hyphen (-) to set right justification when **lshosts** displays the output for the specific field. If not specified, the default is to set left justification when **lshosts** displays the output for a field.
- Specify the unit colon (:) with a unit to set the unit for the output of the specific field:
 - Specify **S** to use a built-in conversion for space or capacity, such as memory or disk space. Values are automatically scaled for M (MB), G (GB), and T (TB), where the default unit is M (MB).
For example, when displaying the **mem** field with a specified width of 3,
 - For a value of 30, running the **lshosts -o "maxmem:3:S"** command shows 30.0M.
 - For a value of 4096, running the **lshosts -o "maxswp:3:S"** command shows 4.0G.
 - For a value of 5000000, running the **lshosts -o "maxtmp:3:S"** command shows 4.8T.
 - Specify **D** to use a built-in conversion for duration or time, such as memory or disk space. Values are automatically scaled for s (seconds), m (minutes), h (hours), and d (days), where the default unit is s (seconds). The automatically scaled value is rounded up after the first decimal point.
For example, when displaying the external **mytime** resource field with a specified width of 5,
 - For a value of 30, running the **lshosts -o "mytime:5:D"** command shows 30.0s.
 - For a value of 8000, running the **lshosts -o "mytime:5:D"** command shows 2.2h.
 - For a value of 5000000, running the **lshosts -o "mytime:5:D"** command shows 57.8d.
 - Specify any other string of 1 - 3 characters and the characters are used as is in the field value. The first character must be a letter (upper or lower case). The second and third characters must be an alphanumeric character.
For example, when displaying the external **gpu_temp** resource with a width of 3, running the **lshosts -o "gpu_temp:3:C"** command for a value of 30 shows 30C
- Use **delimiter=** to set the delimiting character to display between different headers and fields. This delimiter must be a single character. By default, the delimiter is a space.

The **lshosts -o** option overrides the **LSF_LSHOSTS_FORMAT** environment variable, which overrides the **LSF_LSHOSTS_FORMAT** setting in **lsf.conf**.

Output customization applies only to the output for certain **lshosts** options:

- LSF_LSHOSTS_FORMAT** and **lshosts -o** both apply to output for the **lshosts** command with no options, and for **lshosts** options with output that filter information, including the following options: **-a**, **-cname**.
- LSF_LSHOSTS_FORMAT** and **lshosts -o** do not apply to output for other **lshosts** options that use a modified format, including the following options: **-l**, **-w**.

Table 1. Output fields for **lshosts**

Field name	Width	Aliases	Unit
HOST_NAME	20	hname	
type	10		
model	10		
cpuf	10		
ncpus	8		
maxmem	10		LSF_UNIT_FOR_LIMITS in lsfc.conf (KB by default)
maxswp	10		LSF_UNIT_FOR_LIMITS in lsfc.conf (KB by default)
server	10		
RESOURCES	20	res	
ndisks	8		
maxtmp	10		LSF_UNIT_FOR_LIMITS in lsfc.conf (KB by default)
rexpri	10		
nprocs	8		
ncores	8		
nthreads	10		
RUN_WINDOWS	20	runwin	

Field names and aliases are case-sensitive. Valid values for the output width are any positive integer 1 - 4096.

View customized host load information in JSON format

Use the **lshosts -json** option to view the customized **lshosts** output in JSON format. Since JSON is a customized output format, you must use the **lshosts -json** option together with the **-o** option.

View resource reservation information

- [Viewing host-level resource information \(bhosts\)](#)
- [Viewing queue-level resource information \(bqueues\)](#)
- [Viewing reserved memory for pending jobs \(bjobs\)](#)
- [Viewing per-resource reservation \(bresources\)](#)

Viewing host-level resource information (bhosts)

Procedure

1. Use **bhosts -l** to show the amount of resources reserved on each host. In the following example, 143 MB of memory is reserved on `hostA`, and no memory is currently available on the host.

```
bhosts -l hostA
HOST hostA
STATUS    CPUF    JL/U    MAX    NJOBS    RUN    SSUSP    USUSP    RSV    DISPATCH_WINDOW
ok        20.00    -       4       2       1       0       0       1       -
CURRENT LOAD USED FOR SCHEDULING:
           r15s    r1m    r15m    ut     pg     io     ls     it     tmp     swp     mem
slots
Total     1.5     1.2     2.0    91%    2.5     7     49     0     911M    915M    0M
8
Reserved  0.0     0.0     0.0     0%     0.0     0     0     0     0M     0M     143M
8
```

2. Use **bhosts -s** to view information about shared resources.

Viewing queue-level resource information (bqueues)

Procedure

Use **bqueues -l** to see the resource usage that is configured at the queue level.

```
bqueues -l reservation
QUEUE: reservation
-- For resource reservation

PARAMETERS/STATISTICS
PRIO NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SSUSP USUSP  RSV
40    0 Open:Active      -   -   -   -   -    4    0    0    0    0    4

SCHEDULING PARAMETERS
          r15s  r1m   r15m  ut    pg    io    ls    it    tmp    swp    mem
loadSched -      -    -     -     -     -     -     -     -     -     -
loadStop  -      -    -     -     -     -     -     -     -     -     -

          cpuspeed  bandwidth
loadSched          -      -
loadStop           -      -

SCHEDULING POLICIES:  RESOURCE_RESERVE

USERS:  all users
HOSTS:  all

Maximum resource reservation time: 600 seconds
```

Viewing reserved memory for pending jobs (bjobs)

About this task

If the job memory requirements cannot be satisfied, **bjobs -l** shows the pending reason. **bjobs -l** shows both reserved slots and reserved memory.

Procedure

For example, the following job reserves 60 MB of memory on `hostA`:

```
bsub -m hostA -n 2 -q reservation -R"rusage[mem=60]" sleep 8888
Job <3> is submitted to queue <reservation>.
```

bjobs -l shows the reserved memory:

```
bjobs -lp
Job <3>, User <user1>, Project <default>, Status <PEND>, Queue <reservation>
, Command <sleep 8888>
Tue Jan 22 17:01:05 2010: Submitted from host <user1>, CWD </home/user1/>, 2 Processors
Requested, Requested Resources <rusage[mem=60]>, Specified Hosts <hostA>;
Tue Jan 22 17:01:15 2010: Reserved <1> job slot on host <hostA>;
Tue Jan 22 17:01:15 2010: Reserved <60> megabyte memory on host <60M*hostA>;
PENDING REASONS:  Not enough job slot(s): hostA;

SCHEDULING PARAMETERS
          r15s  r1m   r15m  ut    pg    io    ls    it    tmp    swp    mem
loadSched -      -    -     -     -     -     -     -     -     -     -
loadStop  -      -    -     -     -     -     -     -     -     -     -

          cpuspeed  bandwidth
loadSched          -      -
```



```
loadStop      -      -  
RESOURCE REQUIREMENT DETAILS:  
...
```

Viewing per-resource reservation (bresources)

Procedure

Use **bresources** to display per-resource reservation configurations from lsb.resources:

View information about resource allocation limits

Your job might be pending because some configured resource allocation limits are reached. Use the **blimits** command to show the dynamic counters of resource allocation limits configured in Limit sections in the lsb.resources file. The **blimits** command displays the current resource usage to show what limits might be blocking your job.

blimits command

The **blimits** command displays the following information:

- Configured policy name and information for limits that are being applied to running jobs.
- Configured policy name and information for all limits, even if they are not being applied to running jobs (**-a** option).
- Users (**-u** option)
- Queues (**-q** option)
- Hosts (**-m** option)
- Project names (**-P** option)
- Limits (SLOTS, MEM, TMP, SWP, JOBS)
- All resource configurations in lsb.resources (**-c** option). This command option is the same as **bresources** with no options.

Resources that have no configured limits or no limit usage are indicated by a dash (-). Limits are displayed in a USED or LIMIT format. For example, if a limit of 10 slots is configured and 3 slots are in use, then **blimits** displays the limit for SLOTS as 3/10.

If limits MEM, SWP, or TMP are configured as percentages, both the limit and the amount that is used are displayed in MB. For example, **lshosts** displays `maxmem` of 249 MB, and MEM is limited to 10% of available memory. If 10 MB out of 25 MB are used, **blimits** displays the limit for MEM as 10/25 (10 MB USED from a 25 MB LIMIT). MEM, SWP, and TMP can also be configured in other units set in LSF_UNIT_FOR_LIMITS in lsf.conf)

Configured limits and resource usage for built-in resources (slots, mem, tmp, and swp load indices, and number of running and suspended jobs) are displayed as INTERNAL RESOURCE LIMITS separately from custom external resources, which are shown as EXTERNAL RESOURCE LIMITS.

Limits are displayed for both the vertical tabular format and the horizontal format for Limit sections. If a vertical format Limit section has no name, **blimits** displays `NONAMEnnn` under the NAME column for these limits, where the unnamed limits are numbered in the order the vertical-format Limit sections appear in the lsb.resources file.

If a resource consumer is configured as all, the limit usage for that consumer is indicated by a dash (-).

PER_HOST slot limits are not displayed. The **bhosts** command displays these limits as MXJ limits.

In a multicluster environment, **blimits** returns the information about all limits in the local cluster.

Examples

For the following limit definitions:

```

Begin Limit
NAME = limit1
USERS = user1
PER_QUEUE = all
PER_HOST = hostA hostC
TMP = 30%
SWP = 50%
MEM = 10%
End Limit

Begin Limit
NAME = limit_ext1
PER_HOST = all
RESOURCE = ([user1_num,30] [hc_num,20])
End Limit

Begin Limit
NAME = limit2
QUEUES = short
JOBS = 200
End Limit

```

The **blimits** command displays the following information:

```

blimits
INTERNAL RESOURCE LIMITS:

```

NAME	USERS	QUEUES	HOSTS	PROJECTS	SLOTS	MEM	TMP	SWP
JOBS								
limit1	user1	q2	hostA@cluster1	-	-	10/25	-	10/258
limit1	user1	q3	hostA@cluster1	-	-	-	30/2953	-
limit1	user1	q4	hostC	-	-	40/590	-	-
limit2	-	short	-	-	-	-	-	50/200

```

EXTERNAL RESOURCE LIMITS:

```

NAME	USERS	QUEUES	HOSTS	PROJECTS	user1_num	hc_num
limit_ext1	-	-	hostA@cluster1	-	-	1/20
limit_ext1	-	-	hostC@cluster1	-	1/30	1/20

- In limit policy `limit1`, `user1` submitting jobs to `q2`, `q3`, or `q4` on `hostA` or `hostC` is limited to 30% tmp space, 50% swap space, and 10% available memory. No limits are reached, so the jobs from `user1` can run. For example, on `hostA` for jobs from `q2`, 10 MB of memory are used from a 25 MB limit and 10 MB of swap space are used from a 258 MB limit.
- In limit policy `limit_ext1`, external resource `user1_num` is limited to 30 per host and external resource `hc_num` is limited to 20 per host. Again, no limits are reached, so the jobs that request those resources can run.
- In limit policy `limit2`, the short queue can have at most 200 running and suspended jobs. Fifty jobs are running or suspended against the 200 job limit. The limit is not reached, so jobs can run in the `short` queue.

View application profile information

To view the...	Run...
Available application profiles	bapp
Detailed application profile information	bapp -l
Jobs associated with an application profile	bjobs -l -app application_profile_name
Accounting information for all jobs associated with an application profile	bacct -l -app application_profile_name
Job success and requeue exit code information	<ul style="list-style-type: none"> • bapp -l • bacct -l • bhist -l -app application_profile_name • bjobs -l

- [Viewing available application profiles](#)

Viewing available application profiles

Procedure

Run **bapp**. You can view a particular application profile or all profiles.

```
bapp
APPLICATION_NAME  NJOBS  PEND  RUN  SUSP
fluent            0      0    0    0
catia              0      0    0    0
```

A dash (-) in any entry means that the column does not apply to the row.

Viewing detailed application profile information

Procedure

To see the complete configuration for each application profile, run **bapp -l**.

bapp -l also gives current statistics about the jobs in a particular application profile, such as the total number of jobs in the profile, the number of jobs running, suspended, and so on.

Specify application profile names to see the properties of specific application profiles.

```
bapp -l fluent
APPLICATION_NAME: fluent
-- Application definition for Fluent v2.0
STATISTICS:
      NJOBS      PEND      RUN      SSUSP      USUSP      RSV
        0         0         0         0         0         0

PARAMETERS:
CPULIMIT
600.0 min of hostA
RUNLIMIT
200.0 min of hostA
TASKLIMIT
9
FILELIMIT DATALIMIT STACKLIMIT CORELIMIT MEMLIMIT SWAPLIMIT PROCESSLIMIT THREADLIMIT
  800 K      100 K      900 K      700 K      300 K      1000 K      400      500
RERUNNABLE: Y
```

Viewing jobs associated with application profiles

Procedure

Run **bjobs -l -app application_profile_name**.

```
bjobs -l -app fluent
Job <1865>, User <user1>, Project <default>, Application <fluent>,
                               Status <PSUSP>, Queue <normal>, Command <ls>
Tue Jun  6 11:52:05 2009: Submitted from host <hostA> with hold, CWD
                               </clusters/lsf10.1.0/work/cluster1/logdir>;

PENDING REASONS:
Job was suspended by LSF admin or root while pending;
SCHEDULING PARAMETERS:
      r15s  rlm  r15m  ut      pg      io      ls      it      tmp      swp      mem      tlu
loadSched  -   -   -   -   -   -   -   -   -   -   -   -
loadStop   -   -   -   -   -   -   -   -   -   -   -   -

      cpuspeed  bandwidth
loadSched      -   -
loadStop        -   -
...
```

A dash (-) in any entry means that the column does not apply to the row.

Accounting information for all jobs associated with an application profile

Procedure

Run **bacct -l -app** *application_profile_name*.

```
bacct -l -app fluent
```

Accounting information about jobs that are:

- submitted by users jchan,
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on all service classes.
- associated with application profiles: fluent

```
-----
Job <207>, User <user1>, Project <default>, Application <fluent>, Status <DONE>
, Queue <normal>, Command <dir>
Wed May 31 16:52:42 2009: Submitted from host <hostA>, CWD <$HOME/src/mainline/lsbatch/cmd>;
Wed May 31 16:52:48 2009: Dispatched to 10 Hosts/Processors <10*hostA>
Wed May 31 16:52:48 2009: Completed <done>.
Accounting information about this job:
      CPU_T      WAIT      TURNAROUND      STATUS      HOG_FACTOR      MEM      SWAP
      0.02         6         6         done         0.0035         2M         5M
-----
...
SUMMARY:      ( time unit: second )
Total number of done jobs:      15      Total number of exited jobs:      4
Total CPU time consumed:      0.4      Average CPU time consumed:      0.0
Maximum CPU time of a job:      0.0      Minimum CPU time of a job:      0.0
Total wait time in queues:      5305.0
Average wait time in queue:      279.2
Maximum wait time in queue:      3577.0      Minimum wait time in queue:      2.0
Average turnaround time:      306 (seconds/job)
Maximum turnaround time:      3577      Minimum turnaround time:      5
Average hog factor of a job:      0.00 ( cpu time / turnaround time )
Maximum hog factor of a job:      0.01      Minimum hog factor of a job:      0.00
Total throughput:      0.14 (jobs/hour) during 139.98 hours
Beginning time:      May 31 16:52      Ending time:      Jun 6 12:51
...

```

Viewing job success exit values and re-queue exit code information

Procedure

1. Run **bjobs -l** to see command-line re-queue exit values if defined.

```
bjobs -l
```

```
Job <405>, User <user1>, Project <default>, Status <PSUSP>,
Queue <normal>, Command <myjob 1234>
Tue Dec 11 23:32:00 2009: Submitted from host <hostA> with hold, CWD </scratch/d
ev/lsfjobs/user1/work>, Requeue Exit Values <2>;
...

```

2. Run **bapp -l** to see SUCCESS_EXIT_VALUES when the parameter is defined in an application profile.

```
bapp -l
APPLICATION NAME: fluent
-- Run FLUENT applications
```

```
STATISTICS:
      NJOBS      PEND      RUN      SSUSP      USUSP      RSV
      0         0         0         0         0         0
```

```
PARAMETERS:
```

```
SUCCESS_EXIT_VALUES: 230 222 12
...
```

3. Run **bhist -l** to show command-line specified re-queue exit values with **bsub** and modified re-queue exit values with **bmod**.

```
bhist -l
Job <405>, User <user1>, Project <default>, Command <myjob 1234>
Tue Dec 11 23:32:00 2009: Submitted from host <hostA> with hold, to Queue
<norma
                                1>, CWD </scratch/dev/lsfjobs/user1/work>, R
                                e-queue Exit Values <1>;
Tue Dec 11 23:33:14 2009: Parameters of Job are changed:
                                Requeue exit values changes to: 2;
...
```

4. Run **bhist -l** and **bacct -l** to see success exit values when a job is done successfully. If the job exited with default success exit value 0, **bhist** and **bacct** do not display the 0 exit value

```
bhist -l 405
Job <405>, User <user1>, Project <default>, Interactive pseudo-terminal mode, Co
mmmand <myjob 1234>
...
Sun Oct 7 22:30:19 2009: Done successfully. Success Exit Code: 230 222 12.
...

bacct -l 405
...
Job <405>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Comma
nd <myjob 1234>
Wed Sep 26 18:37:47 2009: Submitted from host <hostA>, CWD </scratch/dev/lsfjobs/user1/wo
rk>;
Wed Sep 26 18:37:50 2009: Dispatched to <hostA>;
Wed Sep 26 18:37:51 2009: Completed <done>. Success Exit Code: 230 222 12.
...
```

View fair share information

Monitor how resources are distributed in fair share scheduling policies.

- [View queue-level fair share information](#)
- [Viewing cross-queue fair share information](#)
- [Viewing hierarchical share information for a group](#)
Use **bugroup -l** to find out if you belong to a group, and what its share distribution.
- [Viewing hierarchical share information for a host partition](#)
By default, **bhpart** displays only the top-level share accounts associated with the partition.
- [Viewing host partition information](#)

View queue-level fair share information

Procedure

To find out if a queue is a fair share queue, run the **bqueues -l** command. If you see **USER_SHARES** in the output, then a fair share policy is configured for the queue.

Viewing cross-queue fair share information

Procedure

Run **bqueues -l** to know if a queue is part of cross-queue fair share.

The FAIRSHARE_QUEUES parameter indicates cross-queue fair share. The first queue that is listed in the FAIRSHARE_QUEUES parameter is the parent queue: the queue in which fair share is configured; all other queues listed inherit the fair share policy from the parent queue.

All queues that participate in the same cross-queue fair share display the same fair share information (SCHEDULING POLICIES, FAIRSHARE_QUEUES, USER_SHARES, and SHARE_INFO_FOR) when **bqueues -l** is used. Fair share information applies to all the jobs running in all the queues in the parent-child set.

bqueues -l also displays DISPATCH_ORDER in the parent queue if it is defined.

Viewing hierarchical share information for a group

Use **bugroup -l** to find out if you belong to a group, and what its share distribution.

Procedure

Run **bugroup -l**:

```
bugroup -l
GROUP_NAME: group1
USERS: group2/ group3/
SHARES: [group2,20] [group3,10]

GROUP_NAME: group2
USERS: user1 user2 user3
SHARES: [others,10] [user3,4]

GROUP_NAME: group3
USERS: all
SHARES: [user2,10] [default,5]
```

This command displays all the share trees that are configured, even if they are not used in any fair share policy.

Viewing hierarchical share information for a host partition

By default, **bhpart** displays only the top-level share accounts associated with the partition.

Procedure

Use **bhpart -r** to display the group information recursively.

The output lists all the groups in the share tree, starting from the top level, and displays the following information:

- Number of shares
- Dynamic share priority (LSF compares dynamic priorities of users who belong to same group, at the same level)
- Number of started jobs
- Number of reserved jobs
- CPU time, in seconds (cumulative CPU time for all members of the group, recursively)
- Run time, in seconds (historical and actual run time for all members of the group, recursively)

```
bhpart -r Partition1
HOST_PARTITION_NAME: Partition1
HOSTS: HostA
SHARE_INFO_FOR: Partition1/
USER/GROUP SHARES PRIORITY STARTED RESERVED CPU_TIME RUN_TIME
group1      40      1.867      5      0      48.4      17618
```

group2	20	0.775	6	0	607.7	24664
SHARE_INFO_FOR: Partition1/group2/						
USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
user1	8	1.144	1	0	9.6	5108
user2	2	0.667	0	0	0.0	0
others	1	0.046	5	0	598.1	19556

Viewing host partition information

Procedure

Use **bhpart** to view the following information:

- Host partitions configured in your cluster
- Number of shares (for each user or group in a host partition)
- Dynamic share priority (for each user or group in a host partition)
- Number of started jobs
- Number of reserved jobs
- CPU time, in seconds (cumulative CPU time for all members of the group, recursively)
- Run time, in seconds (historical and actual run time for all members of the group, recursively)

```
% bhpart Partition1
HOST_PARTITION_NAME: Partition1
HOSTS: hostA hostB hostC

SHARE_INFO_FOR: Partition1/
USER/GROUP SHARES PRIORITY STARTED RESERVED CPU_TIME RUN_TIME
group1 100 5.440 5 0 200.0 1324
```

Viewing information about SLAs and service classes

Monitoring the progress of an SLA (bsla)

Use **bsla** to display the properties of service classes configured in lsb.serviceclasses and dynamic state information for each service class. The following are some examples:

- One velocity goal of service class Tofino is active and on time. The other configured velocity goal is inactive.

```
% bsla
SERVICE CLASS NAME: Tofino
-- day and night velocity
PRIORITY = 20

GOAL: VELOCITY 30
ACTIVE WINDOW: (17:30-8:30)
STATUS: Inactive
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

GOAL: VELOCITY 10
ACTIVE WINDOW: (9:00-17:00)
STATUS: Active:On time
SLA THROUGHPUT: 10.00 JOBS/CLEAN_PERIOD

NJOBS PEND RUN SSUSP USUSP FINISH
300 280 10 0 0 10
```

- The deadline goal of service class Uclulet is not being met, and **bsla** displays status `Active:Delayed`.

```
% bsla
SERVICE CLASS NAME: Uclulet
-- working hours
```

```
PRIORITY = 20
```

```
GOAL: DEADLINE
ACTIVE WINDOW: (8:30-19:00)
STATUS: Active:Delayed
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD
ESTIMATED FINISH TIME: (Tue Oct 28 06:17)
OPTIMUM NUMBER OF RUNNING JOBS: 6
```

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
40	39	1	0	0	0

- The configured velocity goal of the service class Kyuquot is active and on time. The configured deadline goal of the service class is inactive.

```
% bsla Kyuquot
SERVICE CLASS NAME: Kyuquot
-- Daytime/Nighttime SLA
PRIORITY = 23
USER_GROUP: user1 user2
```

```
GOAL: VELOCITY 8
ACTIVE WINDOW: (9:00-17:30)
STATUS: Active:On time
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD
```

```
GOAL: DEADLINE
ACTIVE WINDOW: (17:30-9:00)
STATUS: Inactive
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD
```

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
0	0	0	0	0	0

- The throughput goal of service class Inuvik is always active. **bsla** displays:
 - Status as active and on time
 - An optimum number of 5 running jobs to meet the goal
 - Actual throughput of 10 jobs per hour based on the last CLEAN_PERIOD

```
% bsla Inuvik
SERVICE CLASS NAME: Inuvik
-- constant throughput
PRIORITY = 20
```

```
GOAL: THROUGHPUT 6
ACTIVE WINDOW: Always Open
STATUS: Active:On time
SLA THROUGHPUT: 10.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS: 5
```

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
110	95	5	0	0	10

Tracking historical behavior of an SLA (bacct)

Use **bacct** to display historical performance of a service class. For example, service classes Inuvik and Tuktoyaktuk configure throughput goals.

```
% bsla
SERVICE CLASS NAME: Inuvik
-- throughput 6
PRIORITY = 20
```

```
GOAL: THROUGHPUT 6
ACTIVE WINDOW: Always Open
STATUS: Active:On time
SLA THROUGHPUT: 10.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS: 5
```


NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
111	94	5	0	0	12

```

SERVICE CLASS NAME: Tuktoyaktuk
-- throughput 3
PRIORITY = 15

GOAL: THROUGHPUT 3
ACTIVE WINDOW: Always Open
STATUS: Active:On time
SLA THROUGHPUT: 4.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS: 4

```

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
104	96	4	0	0	4

These two service classes have the following historical performance. For SLA Inuvik, **bacct** shows a total throughput of 8.94 jobs per hour over a period of 20.58 hours:

```
% bacct -sla Inuvik
```

Accounting information about jobs that are:

- submitted by users user1,
 - accounted on all projects.
 - completed normally or exited
 - executed on all hosts.
 - submitted to all queues.
 - accounted on service classes Inuvik,
-

```

SUMMARY:      ( time unit: second )
Total number of done jobs:      183      Total number of exited jobs:      1
Total CPU time consumed:      40.0      Average CPU time consumed:      0.2
Maximum CPU time of a job:      0.3      Minimum CPU time of a job:      0.1
Total wait time in queues: 1947454.0
Average wait time in queue: 10584.0
Maximum wait time in queue: 18912.0      Minimum wait time in queue:      7.0
Average turnaround time:      12268 (seconds/job)
Maximum turnaround time:      22079      Minimum turnaround time:      1713
Average hog factor of a job: 0.00 ( cpu time / turnaround time )
Maximum hog factor of a job: 0.00      Minimum hog factor of a job: 0.00
Total throughput:      8.94 (jobs/hour) during 20.58 hours
Beginning time:      Oct 11 20:23      Ending time:      Oct 12 16:58

```

For SLA Tuktoyaktuk, **bacct** shows a total throughput of 4.36 jobs per hour over a period of 19.95 hours:

```
% bacct -sla
Tuktoyaktuk
```

```

naround time )
Maximum hog fa

```

Accounting information about jobs that are:

- submitted by users user1,
 - accounted on all projects.
 - completed normally or exited
 - executed on all hosts.
 - submitted to all queues.
 - accounted on service classes Tuktoyaktuk,
-

```

SUMMARY:      ( time unit: second )
Total number of done jobs:      87      Total number of exited jobs:      0
Total CPU time consumed:      18.0      Average CPU time consumed:      0.2
Maximum CPU time of a job:      0.3      Minimum CPU time of a job:      0.1
Total wait time in queues: 2371955.0
Average wait time in queue: 27263.8
Maximum wait time in queue: 39125.0      Minimum wait time in queue:      7.0
Average turnaround time:      30596 (seconds/job)

```

```

Maximum turnaround time:      44778      Minimum turnaround time:      3355
Average hog factor of a job:  0.00 ( cpu time / turctor of a job:  0.00      Minimum hog
factor of a job:  0.00
Total throughput:              4.36 (jobs/hour) during 19.95 hours
Beginning time:      Oct 11 20:50      Ending time:      Oct 12 16:47

```

Because the run times are not uniform, both service classes actually achieve higher throughput than configured.

- [Monitoring the progress of an SLA \(bsla\)](#)
The `bsla` command displays the properties of service classes configured in the `lsb.serviceclasses` file.

Monitoring the progress of an SLA (bsla)

The `bsla` command displays the properties of service classes configured in the `lsb.serviceclasses` file.

Procedure

Use the `bsla` command to display the properties of service classes configured in the `lsb.serviceclasses` file and dynamic information about the state of each configured service class.

Examples

- The guarantee SLA `bigMemSLA` has 10 slots guaranteed, limited to one slot per host.

```

bsla
SERVICE CLASS NAME:  bigMemSLA
--
ACCESS CONTROL: QUEUES[normal]
AUTO ATTACH: Y
GOAL:  GUARANTEE

POOL NAME              TYPE  GUARANTEED  USED
bigMemPool             slots        10      0

```

- One velocity goal of service class `Tofino` is active and on time. The other configured velocity goal is inactive.

```

bsla
SERVICE CLASS NAME:  Tofino
-- day and night velocity
PRIORITY: 20
GOAL:  VELOCITY 30
ACTIVE WINDOW: (17:30-8:30)
STATUS:  Inactive
SLA THROUGHPUT:  0.00 JOBS/CLEAN_PERIOD

GOAL:  VELOCITY 10
ACTIVE WINDOW: (9:00-17:00)
STATUS:  Active:On time
SLA THROUGHPUT:  10.00 JOBS/CLEAN_PERIOD

NJOBS  PEND  RUN   SSUSP  USUSP  FINISH
300    280   10    0      0     10

```

- The deadline goal of service class `Sooke` is not being met, and the **bsla** command displays status `Active:Delayed`:

```

bsla
SERVICE CLASS NAME:  Sooke
-- working hours
PRIORITY: 20
GOAL:  DEADLINE
ACTIVE WINDOW: (8:30-19:00)
STATUS:  Active:Delayed

```

```
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

ESTIMATED FINISH TIME: (Tue Oct 28 06:17)
OPTIMUM NUMBER OF RUNNING JOBS: 6
NJOBS    PEND    RUN    SSUSP    USUSP    FINISH
  40      39      1      0         0         0
```

- The configured velocity goal of the service class `Duncan` is active and on time. The configured deadline goal of the service class is inactive.

```
bsla Duncan
SERVICE CLASS NAME: Duncan
  -- Daytime/Nighttime SLA
PRIORITY: 23
USER_GROUP: user1 user2

GOAL: VELOCITY 8
ACTIVE WINDOW: (9:00-17:30)
STATUS: Active:On time
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

GOAL: DEADLINE
ACTIVE WINDOW: (17:30-9:00)
STATUS: Inactive
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

NJOBS    PEND    RUN    SSUSP    USUSP    FINISH
  0         0      0      0         0         0
```

- The throughput goal of service class `Sidney` is always active. The **bsla** command displays information about the service class:
 - Status as active and on time
 - An optimum number of 5 running jobs to meet the goal
 - Actual throughput of 10 jobs per hour based on the last `CLEAN_PERIOD`

```
bsla Sidney
SERVICE CLASS NAME: Sidney
  -- constant throughput
PRIORITY: 20

GOAL: THROUGHPUT 6
ACTIVE WINDOW: Always Open
STATUS: Active:On time
SLA THROUGHPUT: 10.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS: 5

NJOBS    PEND    RUN    SSUSP    USUSP    FINISH
 110      95      5      0         0        10
```

Viewing jobs running in an SLA (bjobs)

The **bjobs -sla** command shows jobs running in a service class.

Procedure

Use the **bjobs -sla** command to display jobs running in a service class:

```
bjobs -sla Sidney
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
 136   user1  RUN   normal  hostA      hostA      sleep 100  Sep 28 13:24
 137   user1  RUN   normal  hostA      hostB      sleep 100  Sep 28 13:25
```

For time-based SLAs, use the `-sla` option with the `-g` option to display job groups attached to a service class. Once a job group is attached to a time-based service class, all jobs submitted to that group are subject to the SLA.

Track historical behavior of an SLA (bacct)

The **bacct** command shows historical performance of a service class.

Procedure

Use the **bacct** command to display historical performance of a service class.
The service classes *Sidney* and *Surrey* configure throughput goals.

```
bsla
SERVICE CLASS NAME:  Sidney
  -- throughput 6
PRIORITY:  20

GOAL:  THROUGHPUT 6
ACTIVE WINDOW: Always Open
STATUS:  Active:On time
SLA THROUGHPUT:  10.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS:  5

-----
NJOBS    PEND    RUN    SSUSP    USUSP    FINISH
  111      94      5      0        0      12
-----
SERVICE CLASS NAME:  Surrey
  -- throughput 3
PRIORITY:  15

GOAL:  THROUGHPUT 3
ACTIVE WINDOW: Always Open
STATUS:  Active:On time
SLA THROUGHPUT:  4.00 JOBS/CLEAN_PERIOD
OPTIMUM NUMBER OF RUNNING JOBS:  4

-----
NJOBS    PEND    RUN    SSUSP    USUSP    FINISH
  104      96      4      0        0      4
-----
```

These two service classes have the following historical performance. For SLA *Sidney*, the **bacct** command shows a total throughput of 8.94 jobs per hour over a period of 20.58 hours:

```
bacct -sla Sidney
Accounting information about jobs that are:
  - submitted by users user1,
  - accounted on all projects.
  - completed normally or exited
  - executed on all hosts.
  - submitted to all queues.
  - accounted on service classes Sidney,
-----
SUMMARY:      ( time unit: second )
Total number of done jobs:      183      Total number of exited jobs:      1
Total CPU time consumed:      40.0      Average CPU time consumed:      0.2
Maximum CPU time of a job:      0.3      Minimum CPU time of a job:      0.1
Total wait time in queues: 1947454.0
Average wait time in queue: 10584.0
Maximum wait time in queue: 18912.0      Minimum wait time in queue:      7.0
Average turnaround time:      12268 (seconds/job)
Maximum turnaround time:      22079      Minimum turnaround time:      1713
Average hog factor of a job: 0.00 ( cpu time / turnaround time )
Maximum hog factor of a job: 0.00      Minimum hog factor of a job: 0.00
Total throughput:      8.94 (jobs/hour) during 20.58 hours
Beginning time:      Oct 11 20:23      Ending time:      Oct 12 16:58
```

For SLA *Surrey*, the **bacct** command shows a total throughput of 4.36 jobs per hour over a period of 19.95 hours:

```
bacct -sla Surrey
Accounting information about jobs that are:
  - submitted by users user1,
  - accounted on all projects.
  - completed normally or exited.
  - executed on all hosts.
  - submitted to all queues.
  - accounted on service classes Surrey,
-----
SUMMARY:      ( time unit: second )
Total number of done jobs:      87      Total number of exited jobs:      0
```

Total CPU time consumed:	18.0	Average CPU time consumed:	0.2
Maximum CPU time of a job:	0.3	Minimum CPU time of a job:	0.1
Total wait time in queues:	2371955.0		
Average wait time in queue:	27263.8		
Maximum wait time in queue:	39125.0	Minimum wait time in queue:	7.0
Average turnaround time:	30596 (seconds/job)		
Maximum turnaround time:	44778	Minimum turnaround time:	3355
Average hog factor of a job:	0.00 (cpu time / turnaround time)		
Maximum hog factor of a job:	0.00	Minimum hog factor of a job:	0.00
Total throughput:	4.36 (jobs/hour)	during	19.95 hours
Beginning time:	Oct 11 20:50	Ending time:	Oct 12 16:47

Because the run times are not uniform, both service classes actually achieve higher throughput than configured.

View parallel jobs in EGO enabled SLA

The **bsla -N** command shows job counter information by job slots for a service class

Procedure

Use the **bsla -N** command to display service class job counter information by job slots instead of number of jobs. NSLOTS, PEND, RUN, SSUSP, USUSP are all counted in slots rather than number of jobs:

```
user1@system-02-461: bsla -N SLA1
SERVICE CLASS NAME: SLA1
PRIORITY: 10
CONSUMER: sla1
EGO_RES_REQ: any host
MAX_HOST_IDLE_TIME: 120
EXCLUSIVE: N

GOAL: VELOCITY 1
ACTIVE WINDOW: Always Open
STATUS: Active:On time
SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD
  NSLOTS  PEND    RUN    SSUSP  USUSP
    42     28    14      0      0
```

Viewing configured guaranteed resource pools

Resource-type SLAs have the host or slot guarantee configured within the guaranteed resource pool.

Procedure

Use the **bresources -g -l -m** options to see details of the guaranteed resource pool configuration, including a list of hosts currently in the resource pool.

For example:

```
bresources -glm
GUARANTEED RESOURCE POOL: packagePool
guaranteed package policy, where each package comprises slots and memory together on a single host
```

```
TYPE: package[slots=1:mem=10000]
DISTRIBUTION: [sc1, 4] [sc2, 2]
LOAN_POLICIES: QUEUES[priority] DURATION[10]
HOSTS: hostA.example.com hostB.example.com hostC.example.com hostD.example.com
STATUS: ok
```

RESOURCE SUMMARY:

	slots	mem	packages
TOTAL	48	191.9G	18
FREE	22	168.3G	10
ALLOCATED	6	58.5G	6

OWNER USE	8	3.9G	-
LOAN USE	8	1.9G	-
OTHER USE	10	5.8G	-

OWNER	RESOURCE	ALLOCATED	OWNER USE	LOAN USE	OTHER USE
sc1	slots	4	8	8	10
	mem	39G	3.9G	1.9G	5.8G
	packages	4	-	-	-
sc2	slots	2	0	0	0
	mem	19.5G	0	0	0
	packages	2	-	-	-

					OWNER	LOAN	
OTHER	HOST	RESERVED	RESOURCE	TOTAL	FREE	USE	USE
USE	hostA.example.com	Y	slots	16	0	8	8
0			mem	63.9G	54.2G	3.9G	1.9G
0			packages	6	0	-	-
-							
hostB.example.com	-	slots	16	8	0	0	0
8		mem	63.9G	56G	0	0	0
3.9G		packages	6	5	-	-	-
-							
hostC.example.com	-	slots	16	14	0	0	0
2		mem	63.9G	58G	0	0	0
1.9G		packages	6	5	-	-	-
-							
hostD.example.com	-	-	-	-	-	-	-
-							

- [Viewing guarantee policy information](#)

Viewing guarantee policy information

About this task

Use the **bsla** command to view guarantee policy information from the point of view of a service class. For service classes with guarantee goals, the command lists configuration information for the service class, as well as dynamic information for the guarantees made to that service class in the various pools.

The following is an example of output from the **bsla** command:

```
bsla
```

```
SERVICE CLASS NAME: sla1
-- SLA ONE
ACCESS CONTROL: QUEUES[normal]
AUTO ATTACH: Y
GOAL: GUARANTEE
```

POOL NAME	TYPE	GUARANTEE CONFIG	GUARANTEE USED	TOTAL USED
mypack	package	74	0	0

```
SERVICE CLASS NAME: sla2
-- SLA TWO
ACCESS CONTROL: QUEUES[priority]
AUTO ATTACH: Y
GOAL: GUARANTEE
```

POOL NAME	TYPE	GUARANTEE CONFIG	GUARANTEE USED	TOTAL USED
mypack	package	18	0	0

bresources -g provides information on guarantee policies. It gives a basic summary of the dynamic info of the guarantee pools.

This can also be used together with the **-l** option: **bresources -g -l**. This displays more details about the guarantee policies, including showing what is guaranteed and in use by each of the service classes with a guarantee in the pool. For example:

```
bresources -gl
GUARANTEED RESOURCE POOL: packagePool
guaranteed package policy, where each package comprises slots and memory together on a single host
```

```
TYPE: package[slots=1:mem=10000]
DISTRIBUTION: [sc1, 4] [sc2, 2]
LOAN_POLICIES: QUEUES[priority] DURATION[10]
HOSTS: hostA.example.com hostB.example.com
STATUS: ok
```

RESOURCE SUMMARY:

	slots	mem	packages
TOTAL	32	127.9G	12
FREE	8	110.3G	5
ALLOCATED	6	58.5G	6
OWNER USE	8	3.9G	-
LOAN USE	8	1.9G	-
OTHER USE	8	3.9G	-

OWNER	RESOURCE	ALLOCATED	OWNER USE	LOAN USE	OTHER USE
sc1	slots	4	8	8	8
	mem	39G	3.9G	1.9G	3.9G
	packages	4	-	-	-
sc2	slots	2	0	0	0
	mem	19.5G	0	0	0
	packages	2	-	-	-

The **-m** option can be used together with **-g** and **-l** to get additional host information, including:

- Total packages on the host
- Currently available packages on the host
- Number of resources allocated on the host to jobs with guarantees in the pool
- Number of resources used by owner jobs
- Number of resources used by loaning jobs
- Number of resources used by other jobs

The following example shows hosts in a package pool:

```
bresources -glm
GUARANTEED RESOURCE POOL: packagePool
guaranteed package policy, where each package comprises slots and memory together on a single host
```

```
TYPE: package[slots=1:mem=10000]
DISTRIBUTION: [sc1, 4] [sc2, 2]
LOAN_POLICIES: QUEUES[priority] DURATION[10]
HOSTS: hostA.example.com hostB.example.com
STATUS: ok
```

RESOURCE SUMMARY:

	slots	mem	packages
TOTAL	32	127.9G	12
FREE	8	110.3G	5
ALLOCATED	6	58.5G	6
OWNER USE	8	3.9G	-
LOAN USE	8	1.9G	-
OTHER USE	8	3.9G	-

OWNER	RESOURCE	ALLOCATED	OWNER USE	LOAN USE	OTHER USE
sc1	slots	4	8	8	8
	mem	39G	3.9G	1.9G	3.9G
	packages	4	-	-	-
sc2	slots	2	0	0	0
	mem	19.5G	0	0	0
	packages	2	-	-	-

					OWNER	LOAN
OTHER	HOST	RESERVED	RESOURCE	TOTAL	FREE	USE
USE						USE
hostA.example.com		Y	slots	16	0	8
0			mem	63.9G	54.2G	3.9G
0			packages	6	0	-
-						
hostB.example.com		-	slots	16	8	0
8			mem	63.9G	56G	0
3.9G			packages	6	5	-
-						

View user and user group information

Use the **busers** and **bugroup** commands to display information about LSF users and user groups.

The **busers** command displays information about users and user groups. The default is to display information about the user who runs the command. The **busers** command displays the following information:

- Maximum number of jobs a user or group can run on a single processor
- Maximum number of job slots a user or group can use in the cluster
- Maximum number of pending jobs a user or group can have in the system.
- Total number of job slots required by all submitted jobs of the user
- Number of job slots in the PEND, RUN, SSUSP, and USUSP states

The **bugroup** command displays information about user groups and which users belong to each group.

The **busers** and **bugroup** commands have extra options. See the **busers(1)** and **bugroup(1)** man pages for more details. Restriction: The keyword all is reserved by LSF. Make sure that no actual users are assigned the user name all.

- [Viewing user information](#)
- [Viewing user pending job threshold information](#)
- [Customize user information output](#)
- [Viewing user group information](#)
- [Viewing user share information](#)
- [Viewing user group admin information](#)

Viewing user information

Procedure

Run **busers all**.

```
busers all
USER/GROUP  JL/P  MAX  NJOBS  PEND  RUN  SSUSP  USUSP  RSV
default     12    -    -      -    -    -      -      -
user9       1    12    34     22   10    2      0      0
groupA      -   100    20     7    11    1      1      0
```

Viewing user pending job threshold information

Procedure

Run **busers -w**, which displays the pending job threshold column at the end of the **busers all** output.

```
busers -w
USER/GROUP  JL/P  MAX  NJOBS  PEND  RUN  SSUSP  USUSP  RSV  MPEND
default     12    -    -      -    -    -      -      -    10
user9       1    12    34     22   10    2      0      0   500
groupA      -   100    20     7    11    1      1      0 200000
```

Customize user information output

By default, the **busers** command displays a predefined set of queue information. While you can use various **busers** options to display specific user information based on your needs, you can also customize the specific fields that **busers** displays. Customize output to create a specific **busers** output format that shows all the required information so you can easily parse the information by using custom scripts or to display the information in a predefined format.

Use the `LSB_BUSERS_FORMAT` parameter in `lsf.conf` or the `LSB_BUSERS_FORMAT` runtime environment variable to define the default **busers** output format for LSF:

```
LSB_BUSERS_FORMAT="field_name ..."
```

Use the **busers -o** option to define the custom output at the command level:

```
busers ... -o "field_name ..."
```

Specify which **busers** fields to display, and in which order.

The **busers -o** option overrides the `LSB_BUSERS_FORMAT` environment variable, which overrides the `LSB_BUSERS_FORMAT` setting in `lsf.conf`.

The following are the field names used to specify the **busers** fields to display:

- user (Name of the user or user group)
- jl/p
- max
- nstart (Current number of starting tasks for all of a users' jobs)
- pend
- run
- ssusp
- ususp
- rsv

- njobs
- pjobs
- mpend
- mpjobs
- priority
- ngpus (Number of physical GPUs that the users or user groups are using)
- ngpus_shared (Number of physical GPUs that the users or user groups are using in shared mode)
- ngpus_excl (Number of physical GPUs that the users or user groups are using in exclusive mode)
- ngpus_shared_jexcl (Number of physical GPUs that the users or user groups are using in shared mode, but the jobs of the user is exclusive)
- all (Displays all fields. Specify the colon (:) with an output width that applies to all fields. Available starting in Fix Pack 14.)

Note: The following resource limit field names are supported, but show the same content as their corresponding maximum resource limit fields (that is, the following resource limit field names are aliases): `corelimit`, `cpulimit`, `datalimit`, `filelimit`, `memlimit`, `processlimit`, `runlimit`, `stacklimit`, `swaplimit`, `tasklimit`, `threadlimit`. For example, `corelimit` is the same as `max_corelimit`.

Field names and aliases are not case-sensitive. Valid values for the output width are any positive integer from 1 to 4096.

Viewing user group information

Procedure

Run **bugroup**.

```
bugroup
GROUP_NAME      USERS
testers         user1 user2
engineers       user3 user4 user10 user9
develop         user4 user10 user11 user34 engineers/
system          all users
```

Viewing user share information

Procedure

Run **bugroup -l**, which displays user share group membership information in long format.

```
bugroup -l
GROUP_NAME:  testers
USERS:       user1 user2
SHARES:      [user1, 4] [others, 10]

GROUP_NAME:  engineers
USERS:       user3 user4 user10 user9
SHARES:      [others, 10] [user9, 4]

GROUP_NAME:  system
USERS:       all users
SHARES:      [user9, 10] [others, 15]

GROUP_NAME:  develop
USERS:       user4 user10 user11 engineers/
SHARES:      [engineers, 40] [user4, 15] [user10, 34] [user11, 16]
```

Viewing user group admin information

About this task

If user group administrators are configured in the UserGroup sections of lsb.users they appear in **bugroup** output.

Procedure

Run **bugroup -w**, which displays the user group configuration without truncating columns.

```
bugroup -w
GROUP_NAME  USERS                                GROUP_ADMIN
engineering  user2 groupX groupZ                adminA[usershares]
drafting     user1 user10 user12                adminA adminB[full]
```

View queue information

The **bqueues** command displays information about queues. The **bqueues -l** option also gives current statistics about the jobs in a particular queue, such as the total number of jobs in the queue, the number of running and suspended jobs.

View	Command
Available queues	bqueues
Queue status	bqueues
Detailed queue information	bqueues -l
State change history of a queue	badmin qhist
Queue administrators	bqueues -l for queue

- [Queue states](#)
Queue states, displayed by **bqueues**, describe the ability of a queue to accept and start batch jobs using a combination of the following states:
- [Viewing available queues and queue status](#)
- [Viewing detailed queue information](#)
- [Customize queue information output](#)
- [Viewing the state change history of a queue](#)
- [Viewing queue administrators](#)
- [Viewing exception status for queues \(bqueues\)](#)

Queue states

Queue states, displayed by **bqueues**, describe the ability of a queue to accept and start batch jobs using a combination of the following states:

- Open: queues accept new jobs
- Closed: queues do not accept new jobs
- Active: queues start jobs on available hosts
- Inactive: queues hold all jobs

State	Description
Open:Active	Accepts and starts new jobs—normal processing
Open:Inact	Accepts and holds new jobs—collecting
Closed:Active	Does not accept new jobs, but continues to start jobs-draining
Closed:Inact	Does not accept new jobs and does not start jobs—all activity is stopped

Queue state can be changed by an LSF administrator or **root**.

Queues can also be activated and inactivated by run windows and dispatch windows (configured in lsb.queues, displayed by **bqueues -l**).

bqueues -l displays Inact_Adm when explicitly inactivated by an Administrator (**badmin qinact**), and Inact_Win when inactivated by a run or dispatch window.

Viewing available queues and queue status

Procedure

Run **bqueues**. You can view the current status of a particular queue or all queues. The **bqueues** command also displays available queues in the cluster.

```
bqueues
QUEUE_NAME  PRIO  STATUS      MAX JL/U JL/P JL/H NJOBS  PEND  RUN  SUSP
interactive  400   Open:Active - - - - 2      0    2    0
priority     43    Open:Active - - - - 16     4    4    11  1
night        40    Open:Inactive - - - - 4      4    0    0
short        35    Open:Active - - - - 6      1    5    0
license      33    Open:Active - - - - 0      0    0    0
normal       30    Open:Active - - - - 0      0    0    0
idle         20    Open:Active - - - - 6      3    1    2
```

A dash (-) in any entry means that the column does not apply to the row. In this example no queues have per-queue, per-user, per-processor, or per host job limits configured, so the MAX, JL/U, JL/P, and JL/H entries are shown as a dash.

Job slots required by parallel jobs

Important: A parallel job with *N* components requires *N* job slots.

Viewing detailed queue information

Procedure

To see the complete status and configuration for each queue, run **bqueues -l**. Specify queue names to select specific queues. The following example displays details for the queue **normal**.

```
bqueues -l normal
QUEUE: normal
--For normal low priority jobs, running only if hosts are lightly loaded. This is the
default queue.
PARAMETERS/STATISTICS
PRIO NICE  STATUS      MAX JL/U JL/P NJOBS  PEND  RUN  SSUSP  USUSP
40   20    Open:Active 100 50  11  1      1    0    0    0
Migration threshold is 30 min.
CPULIMIT          RUNLIMIT
20 min of IBM350  342800 min of IBM350
FILELIMIT  DATALIMIT  STACKLIMIT  CORELIMIT  MEMLIMIT  TASKLIMIT
20000 K    20000 K    2048 K      20000 K    5000 K     3

SCHEDULING PARAMETERS
loadSched - 0.7 1.0 0.2 r15s r1m r15m ut pg io ls it tmp swp mem
loadStop - 1.5 2.5 - 8.0 240 - - - - -

cpuspeed bandwidth
loadSched - -
loadStop - -

SCHEDULING POLICIES: FAIRSHARE PREEMPTIVE PREEMPTABLE EXCLUSIVE
USER_SHARES: [groupA, 70] [groupB, 15] [default, 1]
```

```

DEFAULT_HOST_SPECIFICATION : IBM350

RUN_WINDOWS:  2:40-23:00 23:30-1:30
DISPATCH_WINDOWS:  1:00-23:50

USERS: groupA/ groupB/ user5
HOSTS: hostA, hostD, hostB
ADMINISTRATORS: user7
PRE_EXEC: /tmp/apex_pre.x > /tmp/preexec.log 2>&1
POST_EXEC: /tmp/apex_post.x > /tmp/postexec.log 2>&1
REQUEUE_EXIT_VALUES: 45
HOST_PRE_EXEC: /tmp/apex_pre.x > /tmp/preexec.log 2>&1
HOST_POST_EXEC: /tmp/apex_post.x > /tmp/postexec.log 2>&1

```

Customize queue information output

By default, the **bqueues** command displays a predefined set of queue information. While you can use various **bqueues** options to display specific queue information based on your needs, you can also customize the specific fields that **bqueues** displays. Customize output to create a specific **bqueues** output format that shows all the required information so you can easily parse the information by using custom scripts or to display the information in a predefined format.

Use the `LSB_BQUEUES_FORMAT` parameter in `lsf.conf` or the `LSB_BQUEUES_FORMAT` runtime environment variable to define the default **bqueues** output format for LSF:

```
LSB_BQUEUES_FORMAT="field_name[:[-][output_width]] ... [delimiter='character']"
```

Use the **bqueues -o** option to define the custom output at the command level:

```
bqueues ... -o "field_name[:[-][output_width]] ... [delimiter='character']"
```

The following alternative method of using **bqueues -o** is recommended for special delimiter characters in a csh environment (for example, \$):

```
bqueues ... -o 'field_name[:[-][output_width]] ... [delimiter="character"]'
```

- Specify which **bqueues** fields (or aliases instead of the full field names), in which order, and with what width to display.
- Specify only the **bqueues** field name or alias to set its output to unlimited width and left justification.
- (Available starting in Fix Pack 14) Specify `all` to display all fields. Specify the colon (:) with an output width that applies to all fields.
- Specify the colon (:) without a width to set the output width to the supported width for that field.
- Specify the colon (:) with a width to set the maximum number of characters to display for the field. When its value exceeds this width, **bqueues** truncates the ending characters.
- Specify a hyphen (-) to set right justification when **bqueues** displays the output for the specific field. If not specified, the default is to set left justification when **bqueues** displays output for a field.
- Use `delimiter=` to set the delimiting character to display between different headers and fields. This delimiter must be a single character. By default, the delimiter is a space.

The **bqueues -o** option overrides the `LSB_BQUEUES_FORMAT` environment variable, which overrides the `LSB_BQUEUES_FORMAT` setting in `lsf.conf`.

Output customization applies only to the output for certain **bqueues** options:

- `LSB_BQUEUES_FORMAT` and **bqueues -o** both apply to output for the **bqueues** command with no options, and for **bqueues** options with output that filter information, including the following options: `-alloc`, `-m`, `-u`.
- `LSB_BQUEUES_FORMAT` and **bqueues -o** do not apply to output for **bqueues** options that use a modified format, including the following options: `-l`, `-r`, `-w`.

This table outlines the **bqueues** fields to display, and their supported width, aliases you can use instead of field names, and units of measurement for the displayed field:

Table 1. Output fields for bqueues

Field name	Width	Aliases	Unit
queue_name	15	qname	
description	50	desc	
priority	10	prio	
status	12	stat	
max	10		
jl_u	10	jl_u	
jl_p	10	jl_p	
jl_h	10	jl_h	
njobs	10		
pend	10		
run	10		
susp	10		
rsv	10		
ususp	10		
ssusp	10		
nice	6		
max_corelimit	8	corelimit	
max_cpulimit	30	cpulimit	
default_cpulimit	30	def_cpulimit	
max_datalimit	8	datalimit	
default_datalimit	8	def_datalimit	
max_filelimit	8	filelimit	
max_memlimit	8	memlimit	
default_memlimit	8	def_memlimit	
max_processlimit	8	processlimit	
default_processlimit	8	def_processlimit	
max_runlimit	12	runlimit	
default_runlimit	12	def_runlimit	
max_stacklimit	8	stacklimit	
max_swaplimit	8	swaplimit	
max_tasklimit	6	tasklimit	
min_tasklimit	6		
default_tasklimit	6	def_tasklimit	
max_threadlimit	6	threadlimit	
default_threadlimit	6	def_threadlimit	
res_req	20		
hosts	50		
all (Available starting in Fix Pack 14)	Specify an output width that applies to all fields		

Note: The following resource limit field names are supported, but show the same content as their corresponding maximum resource limit fields (that is, the following resource limit field names are aliases): `corelimit`, `cpulimit`, `datalimit`, `filelimit`, `memlimit`, `processlimit`, `runlimit`, `stacklimit`, `swaplimit`, `tasklimit`, `threadlimit`. For example, `corelimit` is the same as `max_corelimit`.

Field names and aliases are not case-sensitive. Valid values for the output width are any positive integer from 1 to 4096.

Remove column headings from the queue information output

Use the **bqueues -noheader** option to remove column headings from the **bqueues** output. When **bqueues -noheader** is specified, **bqueues** displays the values of the fields without displaying the names of the fields. This option is useful for script parsing, when column headings are not necessary.

This option applies to output for the **bqueues** command with no options, and to output for all **bqueues** options with output that uses column headings, including the following: `-alloc`, `-m`, `-o`, `-u`, `-w`.

This option does not apply to output for **bqueues** options that do not use column headings, including the following: -json, -l, -r.

View customized queue information in JSON format

Use the **bqueues -json** option to view the customized **bqueues** output in JSON format. Since JSON is a customized output format, you must use the **bqueues -json** option together with the -o option.

Viewing the state change history of a queue

Procedure

Run **badmin qhist** to display the times when queues are opened, closed, activated, and inactivated.

```
badmin qhist
Wed Mar 31 09:03:14: Queue <normal> closed by user or administrator <root>.
Wed Mar 31 09:03:29: Queue <normal> opened by user or administrator <root>.
```

Viewing queue administrators

Procedure

Run **bqueues -l** for the queue.

Viewing exception status for queues (bqueues)

Procedure

Use **bqueues** to display the configured threshold for job exceptions and the current number of jobs in the queue in each exception state.

For example, queue **normal** configures JOB_IDLE threshold of 0.10, JOB_OVERRUN threshold of 5 minutes, and JOB_UNDERRUN threshold of 2 minutes. The following **bqueues** command shows no overrun jobs, one job that finished in less than 2 minutes (underrun) and one job that triggered an idle exception (less than idle factor of 0.10):

```
bqueues -l normal
QUEUE: normal
-- For normal low priority jobs, running only if hosts are lightly loaded. This is the
default queue.

PARAMETERS/STATISTICS
PRIO NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND  RUN  SSUSP USUSP  RSV
30   20  Open:Active      -   -   -   -   0     0    0    0    0    0

STACKLIMIT MEMLIMIT
2048 K      5000 K

SCHEDULING PARAMETERS
          r15s  r1m  r15m  ut      pg    io   ls    it    tmp    swp    mem
loadSched -    -    -    -      -    -    -    -    -    -    -
loadStop  -    -    -    -      -    -    -    -    -    -    -

          cpuspeed  bandwidth
loadSched          -          -
loadStop           -          -

JOB EXCEPTION PARAMETERS
```

	OVERRUN (min)	UNDERRUN (min)	IDLE (cputime/runtime)
Threshold	5	2	0.10
Jobs	0	1	1

USERS: all users
HOSTS: all

Managing IBM Spectrum LSF job execution

Learn how to manage your LSF jobs and job scheduling policies. View job information, control jobs, and manage job dependencies, job priorities, job arrays, interactive jobs, job pre-execution and post-execution, and job starters.

- [Managing job execution](#)
Learn about LSF job states, how to view information about your jobs, and control job execution by suspending, resuming, stopping, and signaling jobs.
- [Job file spooling](#)
LSF enables spooling of job input, output, and command files by creating directories and files for buffering input and output for a job. LSF removes these files when the job completes.
- [Job data management](#)
LSF provides different options to manage job data.
- [Job scheduling and dispatch](#)
Learn how jobs are scheduled and dispatched to hosts for execution.
- [Control job execution](#)
Use resource usage limits to control how much resource can be consumed by running jobs. Automatically suspend jobs based on the load conditions on the execution hosts. Use pre- and post-execution processing to run commands on an execution host before and after completion of a job. Use job starters to set up the runtime environment for a job. Job submission and execution controls use external, site-specific executable files to validate, modify, and reject jobs, transfer data, and modify the job execution environment.
- [Interactive jobs and remote tasks](#)
Run interactive jobs with the **bsub -I**, **bsub -Is**, and **bsub -Ip** commands to take advantage of batch scheduling policies and host selection features for resource-intensive jobs. Run tasks interactively and remotely with non-batch utilities such as **lsrun** and **lsgrun**.

Managing job execution

Learn about LSF job states, how to view information about your jobs, and control job execution by suspending, resuming, stopping, and signaling jobs.

- [About job states](#)
The **bjobs** command displays the current state of the job.
- [View job information](#)
The **bjobs** command is used to display job information. By default, **bjobs** displays information for the user who invoked the command. For more information about **bjobs**, see the *LSF Reference* and the **bjobs(1)** man page.
- [Force job execution](#)
You can use the **brun** command to force a pending or finished job to run. Only LSF administrators can run the **brun** command.
- [Suspend and resume jobs](#)
A job can be suspended by its owner or the LSF administrator. These jobs are considered user-suspended and are displayed by **bjobs** as USUSP.
- [Kill jobs](#)
The **bkill** command cancels pending batch jobs and sends signals to running jobs. By default, on UNIX, **bkill** sends the SIGKILL signal to running jobs.
- [Send a signal to a job](#)
LSF uses signals to control jobs to enforce scheduling policies, or in response to user requests. The principal signals LSF uses are SIGSTOP to suspend a job, SIGCONT to resume a job, and SIGKILL to terminate a job.

- [Data provenance](#)
LSF allows you to use data provenance tools to trace files that are generated by LSF jobs.

About job states

The **bjobs** command displays the current state of the job.

Normal job states

Most jobs enter only three states:

Job state	Description
PEND	Waiting in a queue for scheduling and dispatch
RUN	Dispatched to a host and running
DONE	Finished normally with a zero exit value

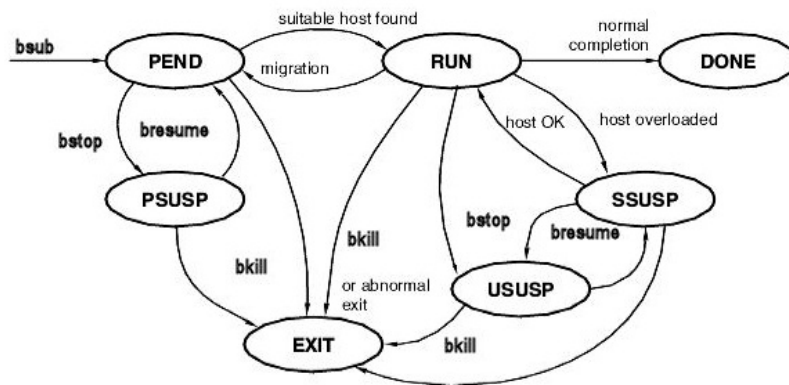
Suspended job states

If a job is suspended, it has three states:

Job state	Description
PSUSP	Suspended by its owner or the LSF administrator while in PEND state
USUSP	Suspended by its owner or the LSF administrator after being dispatched
SSUSP	Suspended by the LSF system after being dispatched

State transitions

A job goes through a series of state transitions until it eventually completes its task, fails, or is terminated. The possible states of a job during its life cycle are shown in the diagram.



Pending jobs

A job remains pending until all conditions for its execution are met. Some of the conditions are:

- Start time that is specified by the user when the job is submitted
- Load conditions on qualified hosts
- Dispatch windows during which the queue can dispatch and qualified hosts can accept jobs
- Run windows during which jobs from the queue can run
- Limits on the number of job slots that are configured for a queue, a host, or a user
- Relative priority to other users and jobs
- Availability of the specified resources
- Job dependency and pre-execution conditions

Maximum pending job threshold

If the user or user group submitting the job has reached the pending job or slots thresholds as specified by `MAX_PEND_JOBS` or `MAX_PEND_SLOTS` (either in the `User` section of `lsb.users`, or cluster-wide in `lsb.params`), LSF will reject any further job submission requests sent by that user or user group. The system will continue to send the job submission requests with the interval specified by `SUB_TRY_INTERVAL` in `lsb.params` until it has made a number of attempts equal to the `LSB_NTRIES` environment variable. If `LSB_NTRIES` is undefined and LSF rejects the job submission request, the system will continue to send the job submission requests indefinitely as the default behavior.

Pending job eligibility for scheduling

A job that is in an eligible pending state is a job that LSF would normally select for resource allocation, but is currently pending because its priority is lower than other jobs. It is a job that is eligible for scheduling and will be run if there are sufficient resources to run it.

An ineligible pending job remains pending even if there are enough resources to run it and is therefore ineligible for scheduling. Reasons for a job to remain pending, and therefore be in an ineligible pending state, include the following:

- The job has a start time constraint (specified with the `-b` option)
- The job is suspended while pending (in a `PSUSP` state).
- The queue of the job is made inactive by the administrator or by its time window.
- The job's dependency conditions are not satisfied.
- The job cannot fit into the run time window (`RUN_WINDOW`)
- Delayed scheduling is enabled for the job (`NEW_JOB_SCHED_DELAY` is greater than zero)
- The job's queue or application profile does not exist.

A job that is not under any of the ineligible pending state conditions is treated as an eligible pending job. In addition, for chunk jobs in `WAIT` status, the time spent in the `WAIT` status is counted as eligible pending time.

If `TRACK_ELIGIBLE_PENDINFO` in `lsb.params` is set to `Y` or `y`, LSF determines which pending jobs are eligible or ineligible for scheduling, and uses eligible pending time instead of total pending time to determine job priority for the following time-based scheduling policies:

- Automatic job priority escalation: Only increases job priority of jobs that have been in an eligible pending state instead of pending state for the specified period of time.
- Absolute priority scheduling (APS): The `JPRIORITY` sub-factor for the APS priority calculation uses the amount of time that the job spent in an eligible pending state instead of the total pending time.

In multicluster job forwarding mode, if the `MC_SORT_BY_SUBMIT_TIME` parameter is enabled in `lsb.params`, LSF counts all pending time before the job is forwarded as eligible for a forwarded job in the execution cluster.

In addition, the following LSF commands also display the eligible or ineligible pending information of jobs if `TRACK_ELIGIBLE_PENDINFO` is set to `Y` or `y`:

- **bjobs**
 - **bjobs -l** shows the total amount of time that the job is in the eligible and ineligible pending states.
 - **bjobs -pei** shows pending jobs divided into lists of eligible and ineligible pending jobs.
 - **bjobs -pe** only shows eligible pending jobs.
 - **bjobs -pi** only shows ineligible pending jobs.

- **bjobs -o** has the `pendstate`, `ependtime`, and `ipendtime` fields that you can specify to display jobs' pending state, eligible pending time, and ineligible pending time, respectively.
- **bacct**
 - **bacct** uses total pending time to calculate the wait time, turnaround time, expansion factor (turnaround time/run time), and hog factor (CPU time or turnaround time).
 - **bacct -E** uses eligible pending time to calculate the wait time, turnaround time, expansion factor (turnaround time/run time), and hog factor (CPU time or turnaround time).

If `TRACK_ELIGIBLE_PENDINFO` is disabled and LSF did not log any eligible or ineligible pending time, the ineligible pending time is zero for **bacct -E**.

- **bhist**
 - **bhist -l** shows the total amount of time that the job spent in the eligible and ineligible pending states after the job started.

mbschd saves eligible and ineligible pending job data to disk every five minutes. This allows the eligible and ineligible pending information to be recovered when **mbatchd** restarts. When **mbatchd** restarts, some ineligible pending time may be lost since it is recovered from the snapshot file, which is dumped periodically at set intervals. The lost time period is counted as eligible pending time under such conditions. To change this time interval, specify the `ELIGIBLE_PENDINFO_SNAPSHOT_INTERVAL` parameter, in minutes, in `lsb.params`.

Suspended jobs

A job can be suspended at any time. A job can be suspended by its owner, by the LSF administrator, by the root user (superuser), or by LSF.

After a job is dispatched and started on a host, it can be suspended by LSF. When a job is running, LSF periodically checks the load level on the execution host. If any load index is beyond either its per-host or its per-queue suspending conditions, the lowest priority batch job on that host is suspended.

If the load on the execution host or hosts becomes too high, batch jobs could be interfering among themselves or could be interfering with interactive jobs. In either case, some jobs should be suspended to maximize host performance or to guarantee interactive response time.

LSF suspends jobs according to the priority of the job's queue. When a host is busy, LSF suspends lower priority jobs first unless the scheduling policy associated with the job dictates otherwise.

Jobs are also suspended by the system if the job queue has a run window and the current time goes outside the run window.

A system-suspended job can later be resumed by LSF if the load condition on the execution hosts falls low enough or when the closed run window of the queue opens again.

WAIT state (chunk jobs)

If you have configured chunk job queues, members of a chunk job that are waiting to run are displayed as WAIT by **bjobs**. Any jobs in WAIT status are included in the count of pending jobs by **bqueues** and **busers**, even though the entire chunk job has been dispatched and occupies a job slot. The **bhosts** command shows the single job slot occupied by the entire chunk job in the number of jobs shown in the `NJOBS` column.

You can switch (**bswitch**) or migrate (**bmig**) a chunk job member in WAIT state to another queue.

Exited jobs

An exited job that is ended with a non-zero exit status.

A job might terminate abnormally for various reasons. Job termination can happen from any state. An abnormally terminated job goes into EXIT state. The situations where a job terminates abnormally include:

- The job is canceled by its owner or the LSF administrator while pending, or after being dispatched to a host.
- The job is not able to be dispatched before it reaches its termination deadline that is set by **bsub -t**, and thus is terminated by LSF.
- The job fails to start successfully. For example, the wrong executable is specified by the user when the job is submitted.

- The application exits with a non-zero exit code.

You can configure hosts so that LSF detects an abnormally high rate of job exit from a host.

Post-execution states

Some jobs may not be considered complete until some post-job processing is performed. For example, a job may need to exit from a post-execution job script, clean up job files, or transfer job output after the job completes.

The DONE or EXIT job states do not indicate whether post-processing is complete, so jobs that depend on processing may start prematurely. Use the `post_done` and `post_err` keywords on the **bsub -w** command to specify job dependency conditions for job post-processing. The corresponding job states POST_DONE and POST_ERR indicate the state of the post-processing.

After the job completes, you cannot perform any job control on the post-processing. Post-processing exit codes are not reported to LSF.

View job information

The **bjobs** command is used to display job information. By default, **bjobs** displays information for the user who invoked the command. For more information about **bjobs**, see the *LSF Reference* and the **bjobs(1)** man page.

- [Viewing all jobs for all users](#)

- [View job IDs](#)

In an multicluster environment, the execution cluster assigns forwarded jobs with different job IDs from the submission cluster. You can use the local job ID or `src_job_id@src_cluster_name` to query the job (for example, **bjobs 123@submission_cluster_name**).

- [Viewing jobs for specific users](#)

- [Viewing running jobs](#)

- [Viewing done jobs](#)

- [Viewing pending job information](#)

When you submit a job, it can be held in the queue before it starts running and it might be suspended while it is running. You can find out why jobs are pending or in suspension with the **bjobs -p** option.

- [Viewing job suspend reasons](#)

When you submit a job, it may be held in the queue before it starts running and it may be suspended while running.

- [Viewing post-execution states](#)

- [Viewing exception status for jobs \(bjobs\)](#)

- [Viewing unfinished job summary information](#)

- [View the job submission environment](#)

Use the **bjobs -env** command option to view a job's environment variables or the **bjobs -script** command option to view the job script file.

- [Customize job information output](#)

Viewing all jobs for all users

Procedure

Run **bjobs -u all** to display all jobs for all users.

Job information is displayed in the following order:

- Running jobs
- Pending jobs in the order in which they are scheduled
- Jobs in high-priority queues are listed before those in lower-priority queues

For example:

```
bjobs -u all
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
1004	user1	RUN	short	hostA	hostA	job0	Dec 16 09:23
1235	user3	PEND	priority	hostM		job1	Dec 11 13:55
1234	user2	SSUSP	normal	hostD	hostM	job3	Dec 11 10:09
1250	user1	PEND	short	hostA		job4	Dec 11 13:59

View job IDs

In an multicluster environment, the execution cluster assigns forwarded jobs with different job IDs from the submission cluster. You can use the local job ID or **src_job_id@src_cluster_name** to query the job (for example, **bjobs 123@submission_cluster_name**).

The advantage of using **src_job_id@src_cluster_name** instead of a local job ID in the execution cluster is that you do not have to know the local job ID in the execution cluster. The **bjobs** output is identical no matter which job ID you use (local job ID or **src_job_id@src_cluster_name**).

Viewing jobs for specific users

Procedure

Run **bjobs -u user_name** to display jobs for a specific user:

```
bjobs -u user1
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
2225	user1	USUSP	normal	hostA		job1	Nov 16 11:55
2226	user1	PSUSP	normal	hostA		job2	Nov 16 12:30
2227	user1	PSUSP	normal	hostA		job3	Nov 16 12:31

Viewing running jobs

Procedure

Run **bjobs -r** to display running jobs.

Viewing done jobs

Procedure

Run **bjobs -d** to display recently completed jobs.

Viewing pending job information

When you submit a job, it can be held in the queue before it starts running and it might be suspended while it is running. You can find out why jobs are pending or in suspension with the **bjobs -p** option.

Procedure

1. Run **bjobs -p**.
Displays information for pending jobs (PEND state) and their reasons. There can be more than one reason why the job is pending.

The pending reasons also display the number of hosts for each condition.
2. To get specific host names along with pending reasons, run **bjobs -lp**.
3. To view the pending reasons for all users, run **bjobs -p -u all**.
4. Run **bjobs -psum** to display the summarized number of jobs, hosts, and occurrences for each pending reason.
5. Run **busers -w all** to see the maximum pending job threshold for all users.

Viewing job suspend reasons

When you submit a job, it may be held in the queue before it starts running and it may be suspended while running.

Procedure

1. Run the **bjobs -s** command.
Displays information for suspended jobs (SUSP state) and their reasons. There can be more than one reason why the job is suspended.

The pending reasons also display the number of hosts for each condition.
2. Run **bjobs -ls** to see detailed information about suspended jobs, including specific host names along with the suspend reason.
The load threshold that caused LSF to suspend a job, together with the scheduling parameters, is displayed.
Note: The STOP_COND parameter affects the suspending reasons as displayed by the **bjobs** command. If the STOP_COND parameter is specified in the queue and the `loadStop` thresholds are not specified, the suspending reasons for each individual load index are not displayed.
3. To view the suspend reasons for all users, run **bjobs -s -u all**.

Viewing post-execution states

Procedure

Run **bhist -l** to display the POST_DONE and POST_ERR states.
The resource usage of post-processing is not included in the job resource usage.

Viewing exception status for jobs (bjobs)

Procedure

Run **bjobs** to display job exceptions. **bjobs -l** shows exception information for unfinished jobs, and **bjobs -x -l** shows finished along with unfinished jobs.

For example, the following **bjobs** command shows that job 1 is running longer than the configured JOB_OVERRUN threshold, and is consuming no CPU time. **bjobs** displays the job idle factor, and both job overrun and job idle exceptions. Job 1 finished before the configured JOB_UNDERRUN threshold, so **bjobs** shows exception status of underrun:

```
bjobs -x -l -a
Job <1>, User <user1>, Project <default>, Status <RUN>, Queue <normal>, Command
    <sleep 600>
Wed Aug 13 14:23:35 2009: Submitted from host <hostA>, CWD <$HOME>, Output File
    </dev/null>, Specified Hosts <hostB>;
```

```

Wed Aug 13 14:23:43 2009: Started on <hostB>, Execution Home </home/user1>, Execution
                          CWD </home/user1>;
Resource usage collected.
                          IDLE_FACTOR(cputime/runtime): 0.00
                          MEM: 3 Mbytes; SWAP: 4 Mbytes; NTHREAD: 3
                          PGID: 5027; PIDs: 5027 5028 5029

MEMORY USAGE:
MAX MEM: 8 Mbytes; AVG MEM: 4 Mbytes

SCHEDULING PARAMETERS:
      r15s  r1m  r15m  ut      pg      io      ls      it      tmp      swp      mem
loadSched  -    -    -    -      -      -      -      -      -      -      -
loadStop   -    -    -    -      -      -      -      -      -      -      -

      cpuspeed  bandwidth
loadSched      -          -
loadStop       -          -

EXCEPTION STATUS:  overrun  idle

RESOURCE REQUIREMENT DETAILS:
Combined : {4*{select[type == local] order[r15s:pg] span[ptile=2]}} || {2*{select
           [type == local] order[r15s:pg] span[hosts=1]}}
Effective : 2*{select[type == local] order[r15s:pg] span[hosts=1] }

```

Use **bacct -l -x** to trace the history of job exceptions.

Viewing unfinished job summary information

Procedure

Run **bjobs -sum** to display summary information about unfinished jobs.

bjobs -sum displays the count of job slots for the following states: running (RUN), system suspended (SSUSP), user suspended (USUSP), UNKNOWN, pending (PEND), and forwarded to remote clusters and pending (FWD_PEND).

bjobs -sum displays the job slot count only for the user's own jobs.

```

% bjobs -sum
RUN      SSUSP      USUSP      UNKNOWN      PEND      FWD_PEND
123      456      789      5      5      3

```

Use **-sum** with other options (like **-m**, **-P**, **-q**, and **-u**) to filter the results. For example, **bjobs -sum -u user1** displays job slot counts just for user `user1`.

```

% bjobs -sum -u user1
RUN      SSUSP      USUSP      UNKNOWN      PEND      FWD_PEND
20      10      10      0      5      0

```

View the job submission environment

Use the **bjobs -env** command option to view a job's environment variables or the **bjobs -script** command option to view the job script file.

About this task

You cannot specify the **-env** option together with the **-script** option. In addition, you cannot specify any other **bjobs** options with the **-env** or **-script** options.

Procedure

- To view the environment variables for a specified job, run the **bjobs -env** command option.

```
bjobs -env job_id
```

You must specify a single job ID or job array element when using the -env command option. Multiple job IDs are not supported.

- To view the specified job's job script file, run the **bjobs -script** command option.

```
bjobs -script job_id
```

You must specify a single job ID or job array element when using the -script command option. Job arrays and multiple job IDs are not supported.

Customize job information output

By default, the **bjobs** command displays a predefined set of job information. While you can use various **bjobs** options to display specific job information based on your needs, you can also customize the specific fields that **bjobs** displays. Customize output to create a specific **bjobs** output format that shows all the required information so you can easily parse the information by using custom scripts or to display the information in a predefined format.

Use the LSB_BJOBS_FORMAT parameter in lsf.conf or the LSB_BJOBS_FORMAT runtime environment variable to define the default **bjobs** output format for LSF:

```
LSB_BJOBS_FORMAT="field_name[:-][output_width][:unit_prefix] ... [delimiter='character']"
```

Use the **bjobs -o** option to define the custom output at the command level:

```
bjobs ... -o "field_name[:-][output_width][:unit_prefix] ... [delimiter='character']"
```

The following alternative method of using **bjobs -o** is recommended for special delimiter characters in a csh environment (for example, \$):

```
bjobs ... -o 'field_name[:-][output_width][:unit_prefix] ... [delimiter="character"]'
```

- Specify which **bjobs** fields (or aliases instead of the full field names), in which order, and with what width to display.
- Specify only the **bjobs** field name or alias to set its output to unlimited width and left justification.
- (Available starting in Fix Pack 14) Specify **all** to display all fields. Specify the colon (:) with an output width that applies to all fields.
- Specify the colon (:) without an output width to set the output width to the recommended width for that field.
- Specify the colon (:) with an output width to set the maximum number of characters to display for the field. When its value exceeds this width, **bjobs** truncates the output:
 - For the JOB_NAME field, **bjobs** removes the header characters and replaces them with an asterisk (*)
 - For other fields, **bjobs** truncates the ending characters
- Specify a hyphen (-) to set right justification when **bjobs** displays the output for the specific field. If not specified, the default is to set left justification when **bjobs** displays the output for a field.
- Specify a second colon (:) with a unit to specify a unit prefix for the output for the following fields: **mem**, **max_mem**, **avg_mem**, **memlimit**, **swap**, **swaplimit**, **corelimit**, **stacklimit**, and **husage** (for **husage**, the unit prefix is for **mem** and **swap** resources only).

This unit is KB (or K) for kilobytes, MB (or M) for megabytes, GB (or G) for gigabytes, TB (or T) for terabytes, PB (or P) for petabytes, EB (or E) for exabytes, ZB (or Z) for zettabytes, or S to automatically adjust the value to a suitable unit prefix and remove the "bytes" suffix from the unit. The default is to automatically adjust the value to a suitable unit prefix, but keep the "bytes" suffix in the unit.

The display value keeps two decimals but rounds up the third decimal. For example, if the unit prefix is set to G, 10M displays as 0.01G.

The unit prefix specified here overrides the value of the LSB_UNIT_FOR_JOBS_DISPLAY environment variable, which also overrides the value of the LSB_UNIT_FOR_JOBS_DISPLAY parameter in the lsf.conf file.

- Use `delimiter=` to set the delimiting character to display between different headers and fields. This delimiter must be a single character. By default, the delimiter is a space.

The **bjobs -o** option overrides the `LSB_BJOBS_FORMAT` environment variable, which overrides the `LSB_BJOBS_FORMAT` setting in `lsf.conf`.

Output customization applies only to the output for certain **bjobs** options:

- `LSB_BJOBS_FORMAT` and **bjobs -o** both apply to output for the **bjobs** command with no options, and for **bjobs** options with short form output that filter information, including the following options: `-a`, `-app`, `-d`, `-g`, `-G`, `-J`, `-Jd`, `-Lp`, `-m`, `-P`, `-q`, `-r`, `-sla`, `-u`, `-x`, `-X`.
- `LSB_BJOBS_FORMAT` does not apply to output for **bjobs** options that use a modified format and filter information, but you can use **bjobs -o** to customize the output for these options. These options include the following options: `-fwd`, `-N`, `-p`, `-s`.
- `LSB_BJOBS_FORMAT` and **bjobs -o** do not apply to output for **bjobs** options that use a modified format, including the following options: `-A`, `-aff`, `-aps`, `-l`, `-UF`, `-ss`, `-sum`, `-UF`, `-w`, `-W`, `-WF`, `-WL`, `-WP`.

The following are the field names used to specify the **bjobs** fields to display, recommended width, aliases you can use instead of field names, and units of measurement for the displayed field:

Table 1. Output fields for bjobs

Field name	Width	Alias	Unit	Description	Category
jobid	7	id		The job ID.	Common
jobindex	8			The array job's element index.	
stat	5			The job's status.	
user	7			The user that submitted the job.	
user_group	15	ugroup		The user group for the user.	
queue	10			The queue for the job.	
job_name	10	name		The job's name.	
job_description	17	description		A description of the job.	
proj_name	11	proj, project		The project to which the job is submitted.	
application	13	app		The application to which the job is submitted.	
service_class	13	sla		The service class to which the job is submitted.	
job_group	10	group		The job group to which the job is submitted.	
job_priority	12	priority		The job's priority.	
rsvd	40			The advance reservation ID for the job.	
esub	20			The job's esub external submission name.	

Field name	Width	Alias	Unit	Description	Category
kill_reason	50			The reason why the job was killed.	
suspend_reason	50			The reason why the job was suspended.	
resume_reason	50			The reason why the job resumed.	
kill_issue_host	50			The host that issued killing the job.	
suspend_issue_host	50			The host that issued suspending the job.	
resume_issue_host	50			The host that issued resuming the job.	
dependency	15			The job's dependency information.	
pend_reason This displays the single key pending reason, including custom messages, regardless of the default pending reason level as specified in the LSB_BJOBS_PENDREASON_LEVEL parameter in the lsf.conf file.	11			The job's pending reason.	
charged_saap	50	saap		The path to which the job's user-based fair share charged.	
command	15	cmd		The job's command.	Command
pre_exec_command	16	pre_cmd		The job's pre-execution command.	
post_exec_command	17	post_cmd		The job's post-execution command.	
resize_notification_command	27	resize_cmd		The job's resize notification command.	
pids	20			The job's PID number.	
exit_code	10			The job's exit code.	
exit_reason	50			The job's exit reason.	
interactive	11			Whether or not the job is interactive.	

Field name	Width	Alias	Unit	Description	Category
from_host	11			The host from where the job is submitted.	Host
first_host	11			The first execution host for the job.	
exec_host	11			All execution hosts for the job.	
nexec_host Note: If the allocated host group or compute unit is condensed, this field does not display the real number of hosts. Use bjobs -X -o to view the real number of hosts in these situations.	10			The number of execution hosts for the job.	
ask_hosts	30			The hosts specified by the job in the submission options.	
alloc_slot	20			List of execution hosts, and the number of slots allocated to the job on each host.	
nalloc_slot	10			The number of slots allocated to the job.	
host_file	10			A user-specified host file while the job was submitted.	
exclusive	13			Whether or not the job is exclusive.	
nreq_slot	10			The number of slots requested by the job	
submit_time	15		time stamp	The time when the job was submitted.	Time
start_time	15		time stamp	The time when the job was started to run.	
estimated_start_time	20	estart_time	time stamp	The job's estimated start time.	
specified_start_time	20	sstart_time	time stamp	The specified start time of the job while submitted.	
specified_terminate_time	24	sterminate_time	time stamp	Specifies the job termination deadline.	

Field name	Width	Alias	Unit	Description	Category
time_left	11		seconds	Amount of time left running based on the run limit of the job.	
finish_time	16		time stamp	The time that the job finished.	
estimated_run_time	20	ertime	seconds	The estimated job run time based on the run limit.	
ru_utime	12		seconds	CPU user time cost by executing the job.	
ru_stime	12		seconds	CPU system time cost by executing the job.	
%complete	11			The percentage job completeness based on run time and run limit.	
warning_action	15	warn_act		The job action to be taken before a job control action occurs.	
action_warning_time	19	warn_time		The job action warning time.	
pendstate (IPEND/EPEND/NOTPEND)	9			The job's pending state: eligible or ineligible pending status.	
pend_time	12		seconds	Amount of time that a job is kept in pending status.	
ependtime	12		seconds	Amount of time that a job is kept in eligible pending status.	
ipendtime	12		seconds	Amount of time that a job is kept in ineligible pending status.	
estimated_sim_start_time	24	esstart_time	time stamp	The job start time, predicted by a simulation-based estimator.	
effective_plimit (run with bjobs - p to show information for pending jobs only)	18		seconds	Effective pending time limit.	

Field name	Width	Alias	Unit	Description	Category
plimit_remain (run with bjobs -p to show information for pending jobs only) A negative number indicates the amount of time in which the job exceeded the pending time limit, while a positive number shows that the time remaining until the pending time limit is reached.	15		seconds	Pending time remaining, based on the effective pending time limit.	
effective_eplimit (run with bjobs -p to show information for pending jobs only)	19		seconds	Effective eligible pending time limit.	
eplimit_remain (run with bjobs -p to show information for pending jobs only)	16		seconds	Effective pending time remaining, based on the pending time limit.	
cpu_used A negative number indicates the amount of time in which the job exceeded the pending time limit, while a positive number shows that the time remaining until the pending time limit is reached.	10			The amount of CPU used time while executing the job.	CPU
run_time	15		seconds	The duration that the job has been running.	
idle_factor	11			The idle factor used for job exception handling.	
exception_status	16	except_stat		The job's exception status.	
slots	5			The number of slots used by the job.	
mem	15		As defined for LSF_UNIT_FOR_LIMITS in lsf.conf.	The amount of memory used by the job.	
max_mem	15		As defined for LSF_UNIT_FOR_LIMITS in lsf.conf.	The maximum memory used by the job.	
avg_mem	15		As defined for LSF_UNIT_FOR_LIMITS in lsf.conf.	The average amount of memory used by the job.	
memlimit	15		As defined for LSF_UNIT_FOR_LIMITS in lsf.conf.	The limit to the amount of memory that can be used by the job.	

Field name	Width	Alias	Unit	Description	Category
swap	15		As defined for LSF_UNIT_FOR_LIMITS in lsf.conf.	The amount of swap memory used by the job.	
swaplimit	15		As defined for LSF_UNIT_FOR_LIMITS in lsf.conf.	The limit to the amount of swap memory that can be used by the job.	
gpu_num	10	gnum		The number of GPUs used by the job.	GPU
gpu_mode	20	gmode		The GPU mode used for the job.	
j_exclusive	15	j_excl		Whether or not the job is using GPUs in job exclusive mode.	
gpu_alloc	30	galloc		GPU allocation for the job.	
nthreads	10			The number of threads created by the job.	Resource usage
husage	50			The amount of host-based resources used by the job.	
min_req_proc	12			The minimum number of processors requested by the parallel job.	Resource requirement
max_req_proc	12			The maximum number of processors requested by the parallel job.	
effective_resreq	17	eresreq		The effective resource requirements of the job.	
combined_resreq	20	cresreq		The combined resource requirements of the job.	
network_req	15			IBM® Parallel Environment (IBM PE) network requirements of the job.	
filelimit	10			The file limit value for the job.	Resource limits
corelimit	15			The core limit value for the job.	

Field name	Width	Alias	Unit	Description	Category
stacklimit	15			The stack limit value for the job.	
processlimit	12			The process limit value for the job.	
runtime limit	12			The runtime limit value for the job.	
plimit	10		seconds	The pending time limit value for the job.	
eplimit	10		seconds	The eligible pending time limit value for the job.	
input_file	10			The specified input file for the job.	File
output_file	11			The specified output file for the job.	
error_file	10			The specified standard error output file for the job.	
output_dir	15			The specified standard output file for the job.	Directory
sub_cwd	10			The path location from where the job was submitted.	
exec_home	10			The home directory for the job on the execution host.	
exec_cwd	10			The current working directory for the job on the execution host.	
licproject	20			The license project specified for the job.	License
forward_cluster	15	fwd_cluster		The name of the cluster that forwarded the job.	MultiCluster
forward_time	15	fwd_time	time stamp	The time when the job was forwarded.	
srcjobid	8			The job ID assigned by the cluster that accepted the job.	

Field name	Width	Alias	Unit	Description	Category
dstjobid	8			The job ID assigned by the cluster that forwarded the job.	
source_cluster	15	srcluster		The name of the cluster that accepted the job.	
energy			Joule	The amount of energy used by the job.	Energy
gpfsio Job disk usage (I/O) data on IBM Spectrum Scale.				The size of input or output disk usage data on IBM Spectrum Scale for the job.	
block (Available starting in Fix Pack 13)	5			Blocking mode. Used for getting jobs submitted with the bsub -K command.	
cpu_peak (Available starting in Fix Pack 13)	10			The peak number of CPUs used by the job.	
cpu_efficiency (for Fix Pack 13) cpu_peak_efficiency (for Fix Pack 14)	10			The peak number of CPUs used by the job compared to the number of requested CPUs, expressed in percentage.	
mem_efficiency (Available starting in Fix Pack 13)	10			The maximum size of memory used by the job compared to the requested memory size for the job, expressed in percentage.	
average_cpu_efficiency (Available starting in Fix Pack 14)	10			The average number of CPUs used by job compared to the number of requested CPUs, expressed in percentage.	
cpu_peak_reached_duration (Available starting in Fix Pack 14)	10			The time it takes for the number of CPU jobs to reach the peak number.	

Field name	Width	Alias	Unit	Description	Category
all (Available starting in Fix Pack 14)	Specify an output width that applies to all fields			All of the fields supported in the bjobs -o command.	

Field names and aliases are not case-sensitive. Valid values for the output width are any positive integer 1 - 4096. If the jobid field is defined with no output width and LSB_JOBID_DISP_LENGTH is defined in lsfc.conf, the LSB_JOBID_DISP_LENGTH value is used for the output width. If jobid is defined with a specified output width, the specified output width overrides the LSB_JOBID_DISP_LENGTH value.

Remove column headings from the job information output

Use the **bjobs -noheader** option to remove column headings from the **bjobs** output. When **bjobs -noheader** is specified, **bjobs** displays the values of the fields without displaying the names of the fields. This option is useful for script parsing, when column headings are not necessary.

This option applies to output for the **bjobs** command with no options, and to output for all **bjobs** options with short form output except for **-aff**, **-l**, **-UF**, **-N**, **-h**, and **-V**.

View customized job information in JSON format

Use the **bjobs -json** option to view the customized **bjobs** output in JSON format. Since JSON is a customized output format, you must use the **bjobs -json** option together with the **-o** option.

Force job execution

You can use the **brun** command to force a pending or finished job to run. Only LSF administrators can run the **brun** command.

You can force a job to run on a particular host to run until completion, and other restrictions. For more information, see the **brun** command.

When a job is forced to run, any other constraints that are associated with the job such as resource requirements or dependency conditions are ignored.

In this situation, some job slot limits, such as the maximum number of jobs that can run on a host, might be violated. A job that is forced to run cannot be preempted.

- [Forcing a pending job to run](#)

Forcing a pending job to run

Procedure

Run **brun -m hostnamejob_ID** to force a pending or finished job to run.
You must specify the host on which the job is to run.

For example, the following command forces the sequential job 104 to run on hostA:

```
brun -m hostA 104
```

Suspend and resume jobs

A job can be suspended by its owner or the LSF administrator. These jobs are considered user-suspended and are displayed by **bjobs** as USUSP.

If a user suspends a high priority job from a non-preemptive queue, the load may become low enough for LSF to start a lower priority job in its place. The load that is created by the low priority job can prevent the high priority job from resuming. This can be avoided by configuring preemptive queues.

- [Suspending a job](#)
- [Resuming a job](#)

Suspending a job

Procedure

Run **bstop** *job_ID*.

Your job goes into USUSP state if the job is already started, or into PSUSP state if it is pending.

```
bstop 3421
Job <3421> is being stopped
```

The preceding example suspends job 3421.

Example

UNIX

bstop sends the following signals to the job:

- SIGTSTP for parallel or interactive jobs—SIGTSTP is caught by the parent process and passed to all the child processes running on other hosts.
- SIGSTOP for sequential jobs—SIGSTOP cannot be caught by user programs. The SIGSTOP signal can be configured with the LSB_SIGSTOP parameter in lsf.conf.

Windows

bstop causes the job to be suspended.

Resuming a job

Procedure

Run **bresume** *job_ID*:

```
bresume 3421
Job <3421> is being resumed
```

Resumes job 3421.

Resuming a user-suspended job does not put your job into RUN state immediately. If your job was running before the suspension, **bresume** first puts your job into SSUSP state and then waits for **sbatchd** to schedule it according to the load conditions.

Kill jobs

The **bkill** command cancels pending batch jobs and sends signals to running jobs. By default, on UNIX, **bkill** sends the SIGKILL signal to running jobs.

Before SIGKILL is sent, SIGINT and SIGTERM are sent to give the job a chance to catch the signals and clean up. The signals are forwarded from **mbatchd** to **sbatchd**. **sbatchd** waits for the job to exit before reporting the status. Because of these delays, for a short period of time after the **bkill** command has been issued, **bjobs** may still report that the job is running.

On Windows, job control messages replace the SIGINT and SIGTERM signals, and termination is implemented by the TerminateProcess() system call.

- [Killing a job](#)
- [Killing multiple jobs](#)
- [Killing jobs by status](#)
- [Killing and recording jobs as DONE status](#)
- [Forcefully removing a job from LSF](#)

Run the **bkill -r** command to remove a job from the LSF system without waiting for the job to terminate in the operating system. This sends the same series of signals as the **bkill** command without **-r**, except that the job is removed from the system immediately. If the job is not in **ZOMBI** state, **bkill -r** will mark the job as **ZOMBI** state, and send a kill signal to the **sbatchd** daemon on the job's execution host. The **mbatchd** daemon will change this job in **ZOMBI** state to **EXIT** state as soon as LSF receives the first signal. If the **bkill -r** command finds a job already in **ZOMBI** state, the **mbatchd** daemon will directly change the job's state to **EXIT**.

- [Removing hung jobs from LSF](#)

A dispatched job becomes hung if its execution host (or first execution host for parallel jobs) goes to either **unreach** or **unavail** state. For jobs with a specified runlimit, LSF considers a job to be hung once the runlimit expires and **mbatchd** attempts to signal **sbatchd** to kill the job, but **sbatchd** is unable to kill the job. During this time, any resources on other hosts held by the job are unavailable to the cluster for use by other pending jobs. This results in poor utilization of cluster resources.

- [Orphan job termination](#)

When one job depends on the result of another job and the dependency condition is never satisfied, the dependent job never runs and remains in the system as an *orphan job*. LSF can automatically terminate jobs that are orphaned when a job they depend on fails.

Killing a job

Procedure

Run **bkill job_ID**. For example, the following command kills job 3421:

```
bkill 3421
Job <3421> is being terminated
```

Killing multiple jobs

Procedure

Run **bkill 0** to kill all pending jobs in the cluster or use **bkill 0** with the **-g**, **-J**, **-m**, **-q**, or **-u** options to kill all jobs that satisfy these options.

The following command kills all jobs dispatched to **the hostA** host:

```
bkill -m hostA 0
Job <267> is being terminated
Job <268> is being terminated
Job <271> is being terminated
```

The following command kills all jobs in the **groupA** job group:

```
bkill -g groupA 0
Job <2083> is being terminated
Job <2085> is being terminated
```

Killing multiple jobs rapidly

About this task

Killing multiple jobs with **bkill 0** and other commands is usually sufficient for moderate numbers of jobs. However, killing a large number of jobs (approximately greater than 1000 jobs) can take a long time to finish.

Procedure

Run **bkill -b** to kill a large number of jobs faster than with normal means. However, jobs that are killed in this manner are not logged to `lsb.acct`.

Local pending jobs are killed immediately and cleaned up as soon as possible, ignoring the time interval that is specified by `CLEAN_PERIOD` in `lsb.params`. Other jobs are killed as soon as possible but cleaned up normally (after the `CLEAN_PERIOD` time interval).

If the **-b** option is used with **bkill 0**, it kills all applicable jobs and silently skips the jobs that cannot be killed.

The **-b** option is ignored if used with **-r** or **-s**.

Killing jobs by status

Procedure

Run **bkill -stat** to kill jobs in the specified status.

This command option kills large numbers of jobs in the specified status as soon as possible.

This option kills all applicable jobs and silently skips the jobs that LSF cannot kill. LSF kills local pending jobs immediately and cleans up the jobs as soon as possible, ignoring the time interval that is specified by `CLEAN_PERIOD` in `lsb.params`. Jobs that are killed in this manner are not logged to the `lsb.acct` file. LSF kills other jobs, such as running jobs, as soon as possible and cleans up these jobs normally.

When running the **bkill -stat** command option, you do not need the job ID, nor do you need one of the **-m**, **-u**, **-q**, **-J**, **-g**, **-sla**, or **-app** options.

The **bkill -stat run** command option kills all running jobs that you can kill.

The **bkill -stat pend** command option only works with three signals that are specified by the **-s** option: `INT`, `KILL`, or `TERM`.

The **-stat** option cannot be used with the **-b** option.

Killing and recording jobs as DONE status

Procedure

Run **bkill -d** to kill jobs and record the jobs as `DONE` after the jobs exit.

Use the **-d** option when working with remote clusters.

The **-d** option is ignored if used with the **-r** or **-s** options.

The **-d** option only takes effect for started jobs that are in the `RUN`, `USUSP`, or `SSUSP` state. Otherwise, the option is ignored.

Forcefully removing a job from LSF

Run the **bkill -r** command to remove a job from the LSF system without waiting for the job to terminate in the operating system. This sends the same series of signals as the **bkill** command without **-r**, except that the job is removed from the system immediately. If the job is not in `ZOMBI` state, **bkill -r** will mark the job as `ZOMBI` state, and send a kill signal to the **sbatchd** daemon on the job's execution host. The **mbatchd** daemon will change this job in `ZOMBI` state to `EXIT` state as soon as LSF receives the first signal. If the **bkill -r** command finds a job already in `ZOMBI` state, the **mbatchd** daemon will directly change the job's state to `EXIT`.

Removing hung jobs from LSF

A dispatched job becomes hung if its execution host (or first execution host for parallel jobs) goes to either `unreach` or `unavail` state. For jobs with a specified runlimit, LSF considers a job to be hung once the runlimit expires and **mbatchd** attempts to signal **sbatchd** to kill the job, but **sbatchd** is unable to kill the job. During this time, any resources on other hosts held by the job are unavailable to the cluster for use by other pending jobs. This results in poor utilization of cluster resources.

About this task

It is possible to manually remove hung jobs with **bkill -r**, but this requires LSF administrators to actively monitor for jobs in `UNKNOWN` state. Instead of manually removing jobs or waiting for the hosts to come back, LSF can automatically terminate the job after reaching a timeout. After removing the job, LSF moves the job to the `EXIT` state to free up resources for other workload, and logs a message in the **mbatchd** log file.

Jobs with a runlimit specified may hang for the following reasons:

- Host status is `unreach`: **sbatchd** on the execution host (or first execution host for parallel jobs) is down. Jobs running on an execution host when **sbatchd** goes down go into the `UNKNOWN` state. These `UNKNOWN` jobs continue to occupy shared resources, making the shared resources unavailable for other jobs.
- Host status is `unavail`: **sbatchd** and LIM on the execution host (or first execution host for parallel jobs) are down (that is, the host status is `unavail`). Jobs running on an execution host when **sbatchd** and LIM go down go into the `UNKNOWN` state.
- Reasons specific to the operating system on the execution host. Jobs that cannot be killed due to an issue with the operating system remain in the `RUN` state even after the run limit has expired.

To enable hung job management, set the `REMOVE_HUNG_JOBS_FOR` parameter in `lsb.params`. When `REMOVE_HUNG_JOBS_FOR` is set, LSF automatically removes hung jobs and frees host resources for other workload. An optional timeout can also be specified for hung job removal. Hung jobs are removed under the following conditions:

- `HOST_UNAVAIL`: Hung jobs are automatically removed if the first execution host is unavailable and a timeout is reached as specified by `wait_time` in the parameter configuration. The default value of `wait_time` is 10 minutes. Hung jobs of any status will be a candidate for removal by LSF when the timeout is reached.
- `runlimit`: Remove the hung job after the job's run limit was reached. You can use the `wait_time` option to specify a timeout for removal after reaching the runlimit. The default value of `wait_time` is 10 minutes. For example, if `REMOVE_HUNG_JOBS_FOR` is defined with `runlimit`, `wait_time=5` and `JOB_TERMINATE_INTERVAL` is not set, the job is removed by **mbatchd** 5 minutes after the job runlimit is reached. Hung jobs in `RUN` status are considered for removal if the `runlimit + wait_time` have expired.

For backwards compatibility with earlier versions of LSF, `REMOVE_HUNG_JOBS_FOR = runlimit` is handled as previously: The grace period is 10 mins + MAX(6 seconds, `JOB_TERMINATE_INTERVAL`) where `JOB_TERMINATE_INTERVAL` is specified in `lsb.params`. The grace period only begins once a job's run limit has been reached.

- `ALL`: Specifies hung job removal for all conditions (both `runlimit` and `host_unavail`). The hung job is removed when the first condition is satisfied. For example, if a job has a run limit, but it becomes hung because a host is unavailable before

the run limit is reached, jobs (running, suspended, etc.) will be removed after 10 minutes after the host is unavailable. Job is placed in `EXIT` status by `mbatchd`.

The output for hung job removal can be shown with the `bhist` command. For example:

```
Job <5293>, User <user1>, Project <default>, Job Group </default/user1>,
Command <sleep 1000>
Tue May 21 00:59:43 2013: Submitted from host <hostA>, to Queue <normal>, CWD
<$HOME>, Specified Hosts <abc210>;
Tue May 21 00:59:44 2013: Dispatched to <abc210>, Effective RES_REQ <select
[type == any] order[r15s:pg] >;
Tue May 21 00:59:44 2013: Starting (Pid 27216);
Tue May 21 00:59:49 2013: Running with execution home </home/user1>, Execution
CWD </home/user1>, Execution Pid <27216>;
Tue May 21 01:05:59 2013: Unknown; unable to reach the execution host;
Tue May 21 01:10:59 2013: Exited; job has been forced to exit with exit code 2.
The CPU time used is unknown;
Tue May 21 01:10:59 2013: Completed <exit>; TERM_REMOVE_HUNG_JOB: job removed from the
LSF system

Summary of time in seconds spent in various states by Tue May 21 13:23:06 2013
  PEND    PSUSP    RUN    USUSP    SSUSP    UNKWN    TOTAL
44147      0    375      0      0      81    44603
```

Where exit code 1 is for jobs removed by the runlimit condition and exit code 2 is for those removed by the `host_unavail` condition.

When defining `REMOVE_HUNG_JOBS_FOR`, note the following:

- **mbatchd restart** and **badmin reconfig** will reset the timeout value for jobs with a `HOST_UNAVAIL` condition.
- Rerunnable jobs are not removed from LSF since they can be dispatched to other hosts.
- The job exit rate for a hung job is considered in the exit rate calculation when the exit rate type is `JOBEXIT`.
- **mbatchd** removes entire running chunk jobs and waiting chunk jobs if a `HOST_UNAVAIL` condition is satisfied. If a runlimit condition is satisfied, only `RUNNING` or `UNKNOWN` members of chunk jobs will be removed.
- When using the LSF multicluster capability, an unavailable host condition (`HOST_UNAVAIL`) works for local hosts and jobs. The forwarded job is handled by the execution cluster depending on how `REMOVE_HUNG_JOBS_FOR` is configured in the execution cluster.
- When the LSF Advanced Edition LSF/XL feature is defined, if the remote host is unavailable, **mbatchd** removes the job based on the timeout value specified in the execution cluster.
- If both `HOST_UNAVAIL` and `runlimit` are defined (or `ALL`), the job is removed for whichever condition is satisfied first.

Related reference

- [REMOVE_HUNG_JOBS_FOR](#)

Orphan job termination

When one job depends on the result of another job and the dependency condition is never satisfied, the dependent job never runs and remains in the system as an *orphan job*. LSF can automatically terminate jobs that are orphaned when a job they depend on fails.

Often, complex workflows are required with job dependencies for proper job sequencing and job failure handling. A parent job can have child jobs that depend on its state before they can start. If one or more conditions are not satisfied, a child job remains pending. However, if the parent job is in a state that prevents a dependent child job from ever running, the child becomes an orphan job. For example, if a child job has a `DONE` dependency on the parent job but the parent ends abnormally, the child can never run because the parent job did not finish normally. The child job becomes an orphan job. Orphaned jobs remain pending in the LSF system.

Keeping orphan jobs in the system can cause performance degradation. The pending orphan jobs consume unnecessary system resources and add unnecessary loads to the daemons, which can impact their ability to do useful work. You might use external scripts for monitoring and terminating orphan jobs, but that would add more work to **mbatchd**.

Enable orphan job termination

Enable orphan job termination two ways:

- An LSF administrator enables the feature at the cluster level by defining a cluster-wide termination grace period with the parameter `ORPHAN_JOB_TERM_GRACE_PERIOD` in the `lsb.params` file. The cluster-wide termination grace period applies to all dependent jobs in the cluster.
- Use the `-ti` suboption of jobs with job dependencies that are specified by `bsub -w` to enforce immediate automatic orphan termination on a per-job basis even if the feature is disabled at the cluster level. Dependent jobs that are submitted with this option that later become orphans are subject to immediate termination without the grace period even if it is defined.

Define a cluster-wide termination grace period

To avoid prematurely killing dependent jobs that users might still want to keep, LSF terminates a dependent job only after a configurable grace period elapses. The orphan termination grace period is the minimum amount of time that the child job must wait before it is eligible for automatic orphan termination. The grace period starts from the point when a child job's dependency becomes invalid.

mbatchd periodically scans the job list and determines jobs for which the dependencies can never be met. The number of job dependencies to evaluate per session is controlled by the cluster-wide parameter `EVALUATE_JOB_DEPENDENCY` in the `lsb.params` file. If an orphan job is detected and it meets the grace period criteria, the **mbatchd** daemon kills the orphan as part of dependency evaluation processing.

Due to various runtime factors (such as how busy **mbatchd** is serving other requests), the actual elapsed time before LSF automatically kills dependent jobs can be longer than the specified grace period. But LSF ensures that the dependent jobs are terminated only after at least the grace period elapses.

To avoid taking a long time to terminate all dependent jobs in a large dependency tree, the grace period is not repeated at each dependency level. When a job is killed, its entire subtree of orphaned dependents can be killed after the grace period is expired.

The elapsed time for the `ORPHAN_JOB_TERM_GRACE_PERIOD` parameter is carried over after LSF restarts so that the grace period is not restarted when LSF restarts.

For example, to use a cluster-wide termination grace period:

1. Set the `ORPHAN_JOB_TERM_GRACE_PERIOD=90` parameter in the `lsb.params` file.
2. Run the **badmin reconfig** command to reconfigure the cluster.
3. Submit a parent job.

```
bsub -J "JobA" sleep 100
```

4. Submit child jobs.

```
bsub -w "done(JobA)" sleep 100
```

5. (Optional) Use commands such as **bjobs -l**, **bhist -l**, or **bparams -l** to query orphan termination settings.

```
bparams -l
Grace period for the automatic termination of orphan jobs:
ORPHAN_JOB_TERM_GRACE_PERIOD = 90 (seconds)
```

6. The parent job is killed. Some orphan jobs must wait for the grace period to expire before they can be terminated by LSF.
7. Use commands such as **bjobs -l**, **bhist -l**, or **bacct -l** to query orphaned jobs that are terminated by LSF.

```
bacct -l <dependent job ID/name>:
Job <job ID>, User <user1>, Project <default>, Status <EXIT>, Queue <normal>,
Command <sleep 100>
Thu Jan 23 14:26:27: Submitted from host <hostA>, CWD <${HOME}/lsfcluster/conf>;
Thu Jan 23 14:26:56: Completed <exit>; TERM_ORPHAN_SYSTEM: orphaned job
                        terminated automatically by LSF.
```

```
Accounting information about this job:
```

CPU_T	WAIT	TURNAROUND	STATUS	HOG_FACTOR	MEM	SWAP
0.00	29	29	exit	0.0000	0M	0M

Note: The **bhist** command on LSF 9.1.2 or earlier shows the message `Signal <KILL> requested by user or administrator <system>`. This message is equivalent to `Signal <KILL> requested by LSF` on LSF 9.1.3 and later. Both messages mean that the orphan job was terminated automatically by LSF.

Enforce automatic orphan termination on a per-job basis

The **-ti** sub option of **bsub -w** command (that is, **bsub -w 'dependency_expression' [-ti]**) indicates that an orphan job is eligible for automatic termination, without waiting for the grace period to expire. The behavior is enforced even if automatic orphan termination is not enabled at the cluster level. LSF terminates a job only as soon as **mbatchd** can detect it, evaluate its dependency and determine it to be an orphan. For this reason, the job might not terminate immediately.

For the **bmod** command, the **-ti** option is not a suboption, and you do not need to respecify the original **bsub -w** dependency expression.

For example, to enforce automatic orphan job termination on a per-job basis:

1. Submit a parent job.

```
bsub -J "JobA" sleep 100
```

2. Submit child jobs with the **-ti** option to ignore the grace period.

```
bsub -w "done(JobA)" -J "JobB" -ti sleep 100
```

3. (Optional) Use commands such as **bjobs -l** or **bhist -l** to query orphan termination settings.

```
bhist -l <dependent job ID/name>:
Job <135>, Job Name <JobB>, User <user1>, Project <default>, Command <sleep 100>
Thu Jan 23 13:25:35: Submitted from host <hostA>, to Queue <normal>, CWD
                     <$HOME/lsfcluster/conf>, Dependency Condition <done(JobA)>
                     - immediate orphan termination for job <Y>;
```

4. The parent job is killed. LSF immediately and automatically kills the orphan jobs that are submitted with the **-ti** suboption.
5. Use commands such as **bjobs -l** or **bhist -l** to query orphaned jobs that are terminated by LSF.

```
bjobs -l <dependent job ID/name>:
Job <135>, Job Name <JobB>, User <user1>, Project <default>, Status <EXIT>,
Queue <normal>, Command <sleep 100>
Thu Jan 23 13:25:42: Submitted from host <hostA>, CWD <$HOME/lsfcluster/conf/
                     sbatch/lsfcluster/configdir>, Dependency Condition
                     <done(JobA)> - immediate orphan termination for job <Y>;
Thu Jan 23 13:25:49: Exited
Thu Jan 23 13:25:49: Completed <exit>; TERM_ORPHAN_SYSTEM:
                     orphaned job terminated automatically by LSF.
```

How LSF uses automatic orphan job termination

- LSF takes a best-effort approach to discovering orphaned jobs in a cluster. Some jobs might not be identified and reported as orphans.
- Orphan jobs that are terminated automatically by LSF are logged in `lsb.events` and `lsb.acct` files. For example, you might see the following event in `lsb.events`:

```
JOB_SIGNAL" "9.12" 1390855455 9431 -1 1 "KILL" 0 "system" "" -1 "" -1
```

- Similar to the **-w** option, the **-ti** suboption is not valid for forwarded remote jobs.
- For automatic orphan termination, if the dependency was specified with a job name and other jobs have the same name, evaluating the status of a child job depends on the `JOB_DEP_LAST_SUB` parameter:
 - If set to 1, a child job's dependency is evaluated based on the most recently submitted parent job with that name. So killing an older parent with that job name does not affect the child and does not cause it to become an orphan.

- If not set, a child job's dependency is evaluated based on all previous parent jobs with that name. So killing any previous parent with that job name impacts the child job and causes it to become an orphan.
- When you manually requeue a running, user-suspended, or system-suspended parent job, the automatic orphan termination mechanism does not prematurely terminate temporary orphans. When you manually requeue an exited or done parent job, the job's dependents might become orphans and be terminated automatically. You must requeue the parent job and any terminated orphan jobs to restart the job flow.

If automatic requeue is configured for a parent job, when the parent job finishes, automatic orphan termination does not prematurely terminate its temporary orphan jobs while the parent job is requeued.

- The **bjdepinfo** command does not consider the running state of the dependent job. It is based on the current dependency evaluation. You can get a reason such as `is invalid`, `never satisfied`, or `not satisfied` even for a running or finished job.
- If a parent job is checkpointed, its dependents might become orphans. If automatic orphan termination is enabled, these orphans can be terminated by LSF before a user restarts the parent job.
- Orphan jobs that are automatically terminated by the system are logged with the exit code `TERM_ORPHAN_SYSTEM` and cleaned from **mbatchd** memory after the time interval specified by the `CLEAN_PERIOD` parameter.

Send a signal to a job

LSF uses signals to control jobs to enforce scheduling policies, or in response to user requests. The principal signals LSF uses are `SIGSTOP` to suspend a job, `SIGCONT` to resume a job, and `SIGKILL` to terminate a job.

Occasionally, you may want to override the default actions. For example, instead of suspending a job, you might want to kill or checkpoint it. You can override the default job control actions by defining the `JOB_CONTROLS` parameter in your queue configuration. Each queue can have its separate job control actions.

You can also send a signal directly to a job. You cannot send arbitrary signals to a pending job; most signals are only valid for running jobs. However, LSF does allow you to kill, suspend, and resume pending jobs.

You must be the owner of a job or an LSF administrator to send signals to a job.

You use the **bkill -s** command to send a signal to a job. If you issue **bkill** without the **-s** option, a `SIGKILL` signal is sent to the specified jobs to kill them. Twenty seconds before `SIGKILL` is sent, `SIGTERM` and `SIGINT` are sent to give the job a chance to catch the signals and clean up.

On Windows, job control messages replace the `SIGINT` and `SIGTERM` signals, but only customized applications are able to process them. Termination is implemented by the `TerminateProcess()` system call.

- [Signals on different platforms](#)
LSF translates signal numbers across different platforms because different host types may have different signal numbering. The real meaning of a specific signal is interpreted by the machine from which the **bkill** command is issued.
- [Sending a signal to a job](#)

Signals on different platforms

LSF translates signal numbers across different platforms because different host types may have different signal numbering. The real meaning of a specific signal is interpreted by the machine from which the **bkill** command is issued.

For example, if you send signal 18 from a SunOS 4.x host, it means `SIGTSTP`. If the job is running on HP-UX and `SIGTSTP` is defined as signal number 25, LSF sends signal 25 to the job.

Sending a signal to a job

About this task

On most versions of UNIX, signal names and numbers are listed in the **kill(1)** or **signal(2)** man pages. On Windows, only customized applications are able to process job control messages that are specified with the **-s** option.

Procedure

Run **bkill -s signal job_id**, where *signal* is either the signal name or the signal number:

```
bkill -s TSTP 3421
Job <3421> is being signaled
```

The preceding example sends the TSTP signal to job 3421.

Data provenance

LSF allows you to use data provenance tools to trace files that are generated by LSF jobs.

You can use LSF data provenance tools to navigate your data to find where the data is coming from and how it is generated. In addition, you can use data provenance information to reproduce your data results when using the same job input and steps.

LSF includes the following scripts to support data provenance:

- **tag.sh**: This post-execution script marks provenance data in the job-generated files.
- **esub.dprov**: This **esub** application automatically enables data provenance. The parameters are the input file names, which are recorded in the job output file as part of the provenance data.
- **showhist.py**: This script generates a picture to show the relationship of the job data files.

Data provenance marks output files that are generated by LSF (as specified by the **bsub -o**, **-oo**, **-e**, and **-eo** option arguments, and as the destination file that is specified as an argument for the **bsub -f** option) and any files in the current working directory for the job that are newer than the job execution time.

- [Prerequisites for LSF data provenance](#)
LSF data provenance requires the following applications to function correctly.
- [Using data provenance tools](#)
Specify the LSF data provenance tools as **bsub** job submission options.

Prerequisites for LSF data provenance

LSF data provenance requires the following applications to function correctly.

You must use IBM® Spectrum Scale (GPFS) as the file system to support the extended attribute specification of files that is required for data provenance. These attributes are attached to the files, which allows LSF to maintain the files with the provenance data. For more details, refer to <https://www.ibm.com/support/knowledgecenter/en/STXKQY/>.

You must use Graphviz, which is an open source graph visualization software, for the LSF data provenance tools to generate pictures to show the relationship of your data files. For more details, refer to <https://www.graphviz.org/>.

Using data provenance tools

Specify the LSF data provenance tools as **bsub** job submission options.

About this task

Procedure

1. Enable data provenance by defining `LSB_DATA_PROVENANCE=Y` as an environment variable or by using the **esub.dprov** application.
The **esub.dprov** script automatically defines `LSB_DATA_PROVENANCE=Y` for the job and takes input file names as parameters.

- To specify the environment variable at job submission time:
`bsub -e LSB_DATA_PROVENANCE=y ... command`
- To specify the **esub.dprov** application at job submission time:
`bsub -a 'dprov(/path/to/input.file)' ... command`
- To specify the **esub.dprov** application as a mandatory **esub** for all job submissions, add `dprov` to the list of applications in the `LSB_ESUB_METHOD` parameter in the `lsf.conf` file:
`LSB_ESUB_METHOD="dprov"`

2. Attach provenance data to the job-generated files by using the predefined script **tag.sh** as a post-execution script.

- To specify the `tag.sh` post-execution script at job submission time:
`bsub -Ep 'tag.sh' ... command`
- To specify the `tag.sh` post-execution script at the application- or queue-level, specify `POST_EXEC=tag.sh` in the `lsb.applications` or `lsb.queue` file.

For example,

- `bsub -e LSB_DATA_PROVENANCE=y -Ep 'tag.sh' myjob`
- `bsub -a 'dprov(/home/userA/test1)' -Ep 'tag.sh' myjob`

You can edit the **tag.sh** script to customize data provenance for your specific environment.

All environment variables that are set for a job are also set when data provenance for a job.

The following additional environment variables apply only to the data provenance environment (that is, the following environment variables are available to the predefined `tag.sh` script that is used for data provenance):

- `LSB_DP_SUBCMD`: The **bsub** job submission command.
 - `LSB_DP_STDINFILE`: The standard input file for the job, as defined in the **bsub -i** option.
 - `LSB_DP_SUBFILES_index`: The source files on the submission host (to be copied to the execution host), as defined in the **bsub -f** option.
 - `LSB_DP_EXECFILES_index`: The destination files on the execution host (copied from the submission host), as defined in the **bsub -f** option.
 - `LSB_DP_FILES`: The number of files to be copied, as defined in the **bsub -f** option.
 - `LSB_DP_INPUTFILES_index`: The files that are defined in the **esub.dprov** script.
 - `LSB_DP_INPUTFILES`: The number files that are defined in the **esub.dprov** script.
3. Optional. Use the **showhist.py** script to show the history information of the job data file.

`showhist.py file_name`

showhist.py generates a picture to show the relationship of the data files.

What to do next

The data provenance script files (`esub.dprov`, `tag.sh`, and `showhist.py`) are all located in the `LSF_TOP/10.1/misc/examples/data_prov` directory. Optionally, you can edit these files to customize the data provenance for your specific environment.

Job file spooling

LSF enables spooling of job input, output, and command files by creating directories and files for buffering input and output for a job. LSF removes these files when the job completes.

- [File spooling for job input, output, and command files](#)
LSF enables *spooling* of job input, output, and command files by creating directories and files for buffering input and output for a job. LSF removes these files when the job completes.
- [Job spooling directory \(JOB_SPOOL_DIR\)](#)
The JOB_SPOOL_DIR in lsb.params sets the job spooling directory. If defined, JOB_SPOOL_DIR should be:
- [Specifying a job command file \(bsub -Zs\)](#)
- [Remote file access with non-shared file space](#)
LSF is usually used in networks with shared file space. When shared file space is not available, use the **bsub -f** command to have LSF copy needed files to the execution host before running the job, and copy result files back to the submission host after the job completes.
- [Job submission option files](#)
LSF enables you to create and use files that contain job submission options.

File spooling for job input, output, and command files

LSF enables *spooling* of job input, output, and command files by creating directories and files for buffering input and output for a job. LSF removes these files when the job completes.

You can make use of file spooling when submitting jobs with the **-is** and **-Zs** options to **bsub**. Use similar options in **bmod** to modify or cancel the spool file specification for the job. Use the file spooling options if you need to modify or remove the original job input or command files before the job completes. Removing or modifying the original input file does not affect the submitted job.

Note: The file path for spooling job input, output, and command files can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory, file name, and expanded values for %J (*job_ID*) and %I (*index_ID*). File spooling is not supported across multiple clusters.

- [Specifying a job input file](#)
- [Changing the job input file](#)

Specifying a job input file

Procedure

- Use **bsub -i input_file** and **bsub -is input_file** to get the standard input for the job from the file path name specified by *input_file*.
input_file can be an absolute path or a relative path to the current working directory, and can be any type of file though it is typically a shell script text file.

The **-is** option spools the input file to the directory specified by the JOB_SPOOL_DIR parameter in lsb.params, and uses the spooled file as the input file for the job.

Note: With **bsub -i** you can use the special characters %J and %I in the name of the input file. %J is replaced by the job ID. %I is replaced by the index of the job in the array, if the job is a member of an array, otherwise by 0 (zero).

- Use **bsub -is** to change the original input file before the job completes. Removing or modifying the original input file does not affect the submitted job.

Results

LSF first checks the execution host to see if the input file exists, and if so uses this file as the input file for the job. Otherwise, LSF attempts to copy the file from the submission host to the execution host. For the file copy to be successful, you must allow remote copy (**rcp**) access, or you must submit the job from a server host where RES is running. The file is copied from the submission host to a temporary file in the directory specified by the JOB_SPOOL_DIR parameter in lsb.params, or your \$HOME/.lsbatch directory on the execution host. LSF removes this file when the job completes.

Changing the job input file

Procedure

- Use **bmod -i input_file** and **bmod -is input_file** to specify a new job input file.
- Use **bmod -in** and **bmod -isn** to cancel the last job input file modification made with either -i or -is.

Job spooling directory (JOB_SPOOL_DIR)

The JOB_SPOOL_DIR in lsb.params sets the job spooling directory. If defined, JOB_SPOOL_DIR should be:

- A shared directory accessible from the management host and the submission host.
- A valid path up to a maximum length up to 4094 characters on UNIX and Linux or up to 255 characters for Windows.
- Readable and writable by the job submission user.

Except for **bsub -is** and **bsub -Zs**, if JOB_SPOOL_DIR is not accessible or does not exist, output is spooled to the default job output directory .lsbatch.

For **bsub -is** and **bsub -Zs**, JOB_SPOOL_DIR must be readable and writable by the job submission user. If the specified directory is not accessible or does not exist, **bsub -is** and **bsub -Zs** cannot write to the default directory and the job will fail.

JOB_SPOOL_DIR specified:

- The job input file for **bsub -is** is spooled to JOB_SPOOL_DIR/lsf_indir. If the lsf_indir directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.
- The job command file for **bsub -Zs** is spooled to JOB_SPOOL_DIR/lsf_cmddir. If the lsf_cmddir directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.

JOB_SPOOL_DIR not specified:

- The job input file for **bsub -is** is spooled to LSB_SHAREDIR/cluster_name/lsf_indir. If the lsf_indir directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.
- The job command file for **bsub -Zs** is spooled to LSB_SHAREDIR/cluster_name/lsf_cmddir. If the lsf_cmddir directory does not exist, LSF creates it before spooling the file. LSF removes the spooled file when the job completes.

If you want to use job file spooling without specifying JOB_SPOOL_DIR, the LSB_SHAREDIR/cluster_name directory must be readable and writable by all the job submission users. If your site does not permit this, you must manually create lsf_indir and lsf_cmddir directories under LSB_SHAREDIR/cluster_name that are readable and writable by all job submission users.

Specifying a job command file (bsub -Zs)

Procedure

- Use **bsub -Zs** to spool a job command file to the directory specified by the JOB_SPOOL_DIR parameter in lsb.params. LSF uses the spooled file as the command file for the job.

Note: The bsub -Zs option is not supported for embedded job commands because LSF is unable to determine the first command to be spooled in an embedded job command.

- Use **bmod -Zs** to change the command file after the job has been submitted. Changing the original input file does not affect the submitted job.
- Use **bmod -Zsn** to cancel the last spooled command file and use the original spooled file.
- Use **bmod -Z** to modify a command submitted without spooling

Remote file access with non-shared file space

LSF is usually used in networks with shared file space. When shared file space is not available, use the **bsub -f** command to have LSF copy needed files to the execution host before running the job, and copy result files back to the submission host after the job completes.

LSF attempts to run a job in the directory where the **bsub** command was invoked. If the execution directory is under the user's home directory, **sbatchd** looks for the path relative to the user's home directory. This handles some common configurations, such as cross-mounting user home directories with the `/net automount` option.

If the directory is not available on the execution host, the job is run in `/tmp`. Any files created by the batch job, including the standard output and error files created by the `-o` and `-e` options to **bsub**, are left on the execution host.

LSF provides support for moving user data from the submission host to the execution host before executing a batch job, and from the execution host back to the submitting host after the job completes. The file operations are specified with the `-f` option to **bsub**.

LSF uses the **lsrcp** command to transfer files. **lsrcp** contacts RES on the remote host to perform file transfer. If RES is not available, the UNIX **rcp** command is used or, if it is set, the command and options specified by setting `LSF_REMOTE_COPY_COMMAND` in `lsf.conf`.

- [Copying files from the submission host to execution host](#)
- [Specifying an input file](#)
- [Copying output files back to the submission host](#)

Copying files from the submission host to execution host

Procedure

Use **bsub -f "[local_fileoperator [remote_file]]"**

To specify multiple files, repeat the `-f` option.

local_file is the file on the submission host, *remote_file* is the file on the execution host.

local_file and *remote_file* can be absolute or relative file path names. You must specify at least one file name. When the file *remote_file* is not specified, it is assumed to be the same as *local_file*. Including *local_file* without the operator results in a syntax error.

Valid values for *operator* are:

>

local_file on the submission host is copied to *remote_file* on the execution host before job execution. *remote_file* is overwritten if it exists.

<

remote_file on the execution host is copied to *local_file* on the submission host after the job completes. *local_file* is overwritten if it exists.

<<

remote_file is appended to *local_file* after the job completes. *local_file* is created if it does not exist.

><, <>

Equivalent to performing the > and then the < operation. The file *local_file* is copied to *remote_file* before the job executes, and *remote_file* is copied back, overwriting *local_file*, after the job completes. <> is the same as ><

LSF tries to change the directory to the same path name as the directory where the **bsub** command was run. If this directory does not exist, the job is run in your home directory on the execution host.

Note:

Specify *remote_file* as a file name with no path when running in non-shared file systems; this places the file in the job's current working directory on the execution host. This way the job will work correctly even if the directory where the **bsub** command is run does not exist on the execution host.

Examples

To submit `myjob` to LSF, with input taken from the file `/data/data3` and the output copied back to `/data/out3`, run the command:

```
bsub -f "/data/data3 > data3" -f "/data/out3 < out3" myjob data3 out3
```

To run the job `batch_update`, which updates the `batch_data` file in place, you need to copy the file to the execution host before the job runs and copy it back after the job completes:

```
bsub -f "batch_data <>" batch_update batch_data
```

Specifying an input file

Procedure

Use **bsub -i** *input_file*.

If the input file specified is not found on the execution host, it is copied from the submission host using the LSF remote file access facility and is removed from the execution host after the job finishes.

Copying output files back to the submission host

About this task

The output files specified with the `bsub -o` and `bsub -e` are created on the execution host, and are not copied back to the submission host by default.

Procedure

Use the remote file access facility to copy these files back to the submission host if they are not on a shared file system. For example, the following command stores the job output in the `job_out` file and copies the file back to the submission host:

```
bsub -o job_out -f "job_out <" myjob
```

Job submission option files

LSF enables you to create and use files that contain job submission options.

- [Specifying a JSON file with job submission options](#)
Use the **bsub -json** command option to submit a job with a JSON file to specify job submission options.
- [Specifying a YAML file with job submission options](#)
Use the **bsub -yaml** command option to submit a job with a YAML file to specify job submission options.
- [JSDL files with job submission options](#)
Use the **bsub -jsdl** command option to submit a job with a JSDL file to specify job submission options.

Specifying a JSON file with job submission options

Use the **bsub -json** command option to submit a job with a JSON file to specify job submission options.

Procedure

1. Create a JSON file with the job submission options.

In the JSON file, specify the **bsub** option name or alias and the value as the key-value pair. To specify job command or job script, use the `command` option name with the name of the command or job script as the value. For options that have no values (flags), use `null` or (for string-type options) an empty value. Specify the key-value pairs under the category name of the option.

For more information on the syntax of the key names and values in the JSON file, see [bsub -json](#).

For the following job submission command:

```
bsub -r -H -N -Ne -i /tmp/input/jobfile.sh -outdir /tmp/output -C 5 -c 2022:12:12 -cn_mem 256 -hostfile /tmp/myHostFile.txt -q normal -G myUserGroup -u "user@example.com" myjob
```

The following JSON file specifies the equivalent job submission command:

```
{
  "io": {
    "inputFile": "/tmp/input/jobfile.sh",
    "outputDir": "/tmp/output"
  },
  "limit": {
    "coreLimit": 5,
    "cpuTimeLimit": "2022:12:12"
  },
  "resource": {
    "computeNodeMem": 256,
    "hostFile": "/tmp/myHostFile.txt"
  },
  "properties": {
    "queueName": "normal",
    "rerun": null
  },
  "schedule": {
    "hold": "",
    "userGroup": "myUserGroup"
  },
  "notify": {
    "notifyJobDone": "",
    "notifyJobExit": "",
    "mailUser": "user@example.com"
  },
  "command": "myjob"
}
```

2. Use the **bsub -json** command to submit a job with the JSON file that you created.
`bsub -json file_name`

Related reference

- [bsub -json command option](#)

Specifying a YAML file with job submission options

Use the **bsub -yaml** command option to submit a job with a YAML file to specify job submission options.

Procedure

1. Create a YAML file with the job submission options.

For more information on the syntax of the key names and values in the YAML file, see [bsub -yaml](#).

For the following job submission command:

```
bsub -r -H -N -Ne -i /tmp/input/jobfile.sh -outdir /tmp/output -C 5 -c 2022:12:12 -cn_mem 256 -hostfile /tmp/myHostFile.txt -q normal -G myUserGroup -u "user@example.com" myjob
```

The following YAML file specifies the equivalent job submission command:

```
io:
  inputFile: /tmp/input/jobfile.sh
  outputDir: /tmp/output
limit:
  coreLimit: 5
  cpuTimeLimit: 2022:12:12
resource:
  computeNodeMem: 256
  hostFile: /tmp/myHostFile.txt
properties:
  queueName: normal
  rerun: null
schedule:
  hold: ""
  userGroup: myUserGroup
notify:
  notifyJobDone: ""
  notifyJobExit:
  mailUser: user@example.com
command: myjob
```

2. Use the **bsub -yaml** command to submit a job with the YAML file that you created.

bsub -yaml file_name

Related reference

- [bsub -yaml command option](#)

JSDL files with job submission options

Use the **bsub -jsdl** command option to submit a job with a JSDL file to specify job submission options.

For more information about submitting jobs using JSDL, including a detailed mapping of JSDL elements to LSF submission options, and a complete list of supported and unsupported elements, see [Submitting jobs using JSDL](#).

Related concepts

- [Submitting jobs using JSDL](#)

Related reference

- [bsub -jsdl command option](#)
- [bsub -jsdl strict command option](#)

Job data management

LSF provides different options to manage job data.

When managing jobs in a cluster, the compute and storage resources are not always in the same physical location in a cluster. When running jobs in these cases, LSF can execute an application to a site that is close to where the data is stored, or move the data to a site that is close to the application execution site. LSF provides the different options to deal with these situations.

- [Copy a file from the local host to a remote host \(bsub -f\)](#)
Use the **bsub -f** command to copy a job file from the local (submission) host to the remote (execution) host.
- [Use LSF Data Manager for data staging](#)
LSF Data Manager is an LSF add-on that stages required data as closely as possible to the site of the application.
- [Use direct data staging \(bsub -stage\)](#)
Use the **bsub -stage** command to specify options for direct data staging (for example, IBM CAST burst buffer).
- [Configuring LSF for direct data staging](#)
Configure LSF to run direct data staging jobs (for example, IBM CAST burst buffer).

Copy a file from the local host to a remote host (bsub -f)

Use the **bsub -f** command to copy a job file from the local (submission) host to the remote (execution) host.

The **bsub -f** command, which is a standard LSF command, is the most mature method that is available for job data management. This command option provides data staging from the submission environment, to the execution environment, and back. It assumes that the required files are available on the submission host. The data copies are performed as part of a job's allocation. When the data files are large and the jobs request a large amount of resources, this means that the majority of these jobs are idle while the files are transferred. If multiple jobs or job array elements request the same file to be copied over the same slow link, LSF repeatedly copies it for each request. If the data does not exist in the submission environment but is stored in a separate site that is not part of the LSF cluster, the user is forced to implement the data copy as a part of the job's script. This method of moving data files is best suited to clusters where there are few jobs that require data staging, there are no opportunities for data reuse, or the data files to be moved are small.

When using the **bsub -f** command, LSF uses the **lsrnp** to transfer files. **lsrnp** contacts RES on the remote host to perform the file transfer. If RES is not available, LSF uses **rcp**. Ensure that the **rcp** binary file is in the user's **\$PATH** on the execution host.

Related tasks

- [Copying files from the submission host to execution host](#)

Related reference

- [lsrnp command](#)

Use LSF Data Manager for data staging

LSF Data Manager is an LSF add-on that stages required data as closely as possible to the site of the application.

When large amounts of data are required to complete computations, it is important for your applications to access the required data without being affected by the location of the data in relation to the application execution environment. LSF Data Manager can stage input data from an external source storage repository to a cache that is accessible to the cluster execution hosts. LSF Data Manager stages output data asynchronously (dependency-free) from the cache after job completion. Data transfers run separately from the job allocation, which means more jobs can request data without consuming resources while they wait for large data transfers. Remote execution cluster selection and cluster affinity are based on data availability in the LSF multicluster capability environment. LSF Data Manager transfers the required data to the cluster that the job was forwarded to. This method of moving data files is best suited to situations with large amounts of data, and there are opportunities for data reuse.

LSF Data Manager is also useful when moving data between clusters (for example, from a local cluster on premises to a cluster in the cloud). This scenario uses LSF Data Manager as a data gateway when using the LSF resource connector to make data available to virtual machines in the public cloud. This scenario uses the LSF multicluster capability and is set up as follows:

1. Install an on-cloud LSF cluster with one management host.
2. Connect the local LSF cluster to the on-cloud LSF cluster using the LSF multicluster capability.
3. Configure the LSF resource connector in the on-cloud LSF cluster only.
The on-cloud LSF cluster grows or shrinks using the LSF resource connector based on demand.
4. Install LSF Data Manager to both the local and the on-cloud LSF clusters. LSF Data Manager ensures data availability.

You can also configure LSF so that if LSF Data Manager is installed and a user runs the **bsub -f** command option, LSF Data Manager is used to transfer the files instead. For more details on how to use **bsub -f** with LSF Data Manager, refer to [Transferring data requirement files with bsub -f](#).

When using LSF with LSF Data Manager, you must enable passwordless **ssh** between the I/O nodes in the transfer queue and all file servers (source hosts for stage in, and destination hosts for stage out). Any compute node that does not directly mount the staging area must also have passwordless SSH access to the staging area's configured file servers.

Related concepts

-

Use direct data staging (bsub -stage)

Use the **bsub -stage** command to specify options for direct data staging (for example, IBM CAST burst buffer).

The **bsub -stage** command, which is a standard LSF command, is the latest method that is available for job data management. This command option allows a user to specify options for direct data staging (including the amount of storage required on execution host, and the stage in or stage out scripts to be run). Direct data staging jobs use a fast storage layer, such as IBM CAST burst buffer, between LSF servers and a parallel file system. The burst buffer allows overlapping direct data staging jobs to copy files between submission hosts and execution hosts (**bsub -f**), and to stage in and stage out data for a variety of workloads, including checkpoint/restart, scratch volume, and extended memory I/O workloads.

Using direct data staging allows LSF to consider storage that is local to an execution host. LSF provides two-level stage in and stage out operations. The `LSB_STAGE_IN_EXEC` and `LSB_STAGE_OUT_EXEC` options in the `lsf.conf` file specify scripts that are created by the cluster administrator and executed by root. The stage in and stage out scripts specified during job submission are passed into the system level scripts. These scripts can set up and clean up a job's use of local storage. The stage in and stage out scripts run separately from the job allocation. This method of moving data files allows LSF to take advantage of execution hosts with high performance local storage and allow more flexibility when creating stage in and stage out scripts that are customized for the site.

- [Submitting direct data staging jobs](#)
Submit direct data staging jobs (for example, IBM CAST burst buffer) with LSF.

Submitting direct data staging jobs

Submit direct data staging jobs (for example, IBM CAST burst buffer) with LSF.

About this task

Submit a direct data staging job and observe LSF behavior.

Procedure

1. Use the **bsub -stage** option to submit a job with direct data staging options.

```
bsub -stage "[storage= min_size [ , max_size ] ] [:in=path_to_stage_in_script ] [:out=path_to_stage_out_script ]" -q  
data_queue [-f " local_file operator [remote_file]" ...] [-data " data_requirements " [-datagr "user_group_name " ] ]  
command
```

Use the -f option to copy files between the local (submission) host and the remote (execution) host.

Use the -data and -datagr options to specify the data requirements for a job with LSF Data Manager.

For more details on specifying data requirements for your job, refer to [Specifying data requirements for your job](#).

For example,

```
bsub -stage "storage=5:in=/u/usr1/mystagein.pl:out=/home/mystagein.pl" -q bbq myjob
```

2. If you specified data requirements, use the **bjobs -data job_id** option to view the data requirements for the submitted job.

3. Manage data staging details for the job.

After you submit a job with direct data staging options, LSF automatically submits a stage in job to the transfer queue, and later submits a stage out job to the same transfer queue. Stage out jobs are always submitted.

LSF can export the following environment variables in the user scripts to the stage in/out transfer jobs:

LSF_STAGE_JOBID

The original job's base ID.

LSF_STAGE_JOBINDEX

The original job's array index.

LSF_STAGE_USER

The original job's submission user.

LSF_STAGE_HOSTS

Allocated host list for the original job.

LSF_STAGE_STORAGE_MINSIZE

Minimum SSD storage space.

LSF_STAGE_STORAGE_MAXSIZE

Target maximum SSD storage space.

LSF_STAGE_USER_STAGE_IN

User stage in script.

LSF_STAGE_USER_STAGE_OUT

User stage out script.

LSF_STAGE_IN_STATUS

Stage in job exit code.

LSF_STAGE_JOB_STATUS

The original job exit code.

LSF_STAGE_HOSTFILE

A file that contains all hosts that are reserved for the data job, including the names of the launch node.

CSM_ALLOCATION_ID

If you are using LSF with IBM Cluster Systems Manager (CSM), this environment variable is the CSM allocation ID of the original job.

Note: If the administrator stage in script exits with code 125, LSF kills the pending user job and triggers a stage out.

Then, this job does not go back to the system to wait to be rescheduled.

4. View the direct data staging details for the job.

- a. Use the **bhist -l** option to view the original direct data staging job and the stage in and stage out transfer jobs.

- b. If you are using LSF with CSM, use the CSM command **csm_allocation_query_details -a allocation_id** to view the state transition of the allocation.

Related concepts

- [with IBM Cluster Systems Manager](#)

Related tasks

- [Specifying data requirements for your job](#)

Related reference

- [bsub -stage](#)
- [bsub -f](#)
- [bsub -data](#)
- [bjobs -data](#)

Configuring LSF for direct data staging

Configure LSF to run direct data staging jobs (for example, IBM CAST burst buffer).

Procedure

1. Edit the `lsf.conf` file and define the direct data staging parameters.

```
#To support stage in/out
LSB_STAGE_IN_EXEC=bb_pre_exec.sh #bb_pre_exec.sh should be stored under $LSF_SERVERDIR
LSB_STAGE_OUT_EXEC=bb_post_exec.sh #bb_post_exec.sh should be stored under $LSF_SERVERDIR
LSB_STAGE_STORAGE=vg_available_size
LSB_STAGE_TRANSFER_RATE=0.2 #trickle rate for file transfer.
```

2. Edit the `lsb.params` file and define the direct data staging parameters that enable the allocation planner.

```
ALLOCATION_PLANNER=WINDOW[6000]
```

3. Edit the `lsb.queue`s file and configure the data transfer and data job queues.

Data transfer queue:

```
Begin Queue
QUEUE_NAME    = transfer
PRIORITY      = 99999
DATA_TRANSFER = Y
PLAN=N
HOSTS = <node_name> # Can be a launch node must have burst buffer rpm installed
End Queue
# Note: Do not submit jobs to this queue.
```

Data job queue:

```
Begin Queue
QUEUE_NAME    = bbq
PLAN = Y
JOB_CONTROLS  = SUSPEND[bmig $LSB_BATCH_JID]
RERUNNABLE    = Y
End Queue
```

4. Restart the **mbatchd** daemon to apply your changes.

```
badmin mbdrestart
```

5. Verify the LSF queue and cluster parameters are correct.

For example,

```
bqueues transfer bbq
bparams -a
```

Job scheduling and dispatch

Learn how jobs are scheduled and dispatched to hosts for execution.

Configure time windows for job dispatch and execution and use deadline constraint scheduling suspend or stop running jobs at certain time. Configure application profiles to improve the management of applications by separating scheduling policies from application-level requirements. Give jobs exclusive use of an execution host. Define job dependencies where the start of one job depends on the state of other jobs. Assign priority to jobs to control the order that jobs are dispatched. Re-queue jobs automatically when they fail. Rerun jobs automatically when an execution host becomes unavailable while a job is running.

- [Use exclusive scheduling](#)
Exclusive scheduling gives a job exclusive use of the host that it runs on. LSF dispatches the job to a host that has no other jobs running, and does not place any more jobs on the host until the exclusive job is finished.
- [Job dependency and job priority](#)
LSF provides ways to manage job dependency and job priority to provide further control the order in which to schedule jobs.
- [Job re-queue and job rerun](#)
- [Predict job start time using a simulation-based estimator](#)
LSF can predict an approximate start time for these pending jobs by using a simulation-based job start time estimator that runs on the management host and is triggered by the **mbatchd** daemon. The estimator uses a snapshot of the cluster (including the running jobs and available resources in the cluster) to simulate job scheduling behavior and determine when jobs finish and the pending jobs start. This gives users an idea of when their jobs are expected to start.

Use exclusive scheduling

Exclusive scheduling gives a job exclusive use of the host that it runs on. LSF dispatches the job to a host that has no other jobs running, and does not place any more jobs on the host until the exclusive job is finished.

Compute unit exclusive scheduling gives a job exclusive use of the compute unit that it runs on.

How exclusive scheduling works

When you submit an exclusive job (**bsub -x**) to an exclusive queue (the queue defines the **EXCLUSIVE = Y** or **EXCLUSIVE = CU** parameter in the **lsb.queues** file) and dispatched to a host, LSF locks the host (**lockU** status) until the job finishes.

LSF cannot place an exclusive job unless there is a host that has no jobs running on it.

To make sure exclusive jobs can be placed promptly, configure some hosts to run one job at a time. Otherwise, a job could wait indefinitely for a host in a busy cluster to become completely idle.

Resizable jobs

For pending allocation requests with resizable exclusive jobs, LSF does not allocate slots on a host that is occupied by the original job. For newly allocated hosts, LSF locks the LIM if the **LSB_DISABLE_LIMLOCK_EXCL=Y** parameter is not defined in the **lsf.conf** file.

If an entire host is released by a job resize release request with exclusive jobs, LSF unlocks the LIM if **LSB_DISABLE_LIMLOCK_EXCL=Y** is not defined in **lsf.conf**.

Restriction: Jobs with compute unit resource requirements cannot be auto-resizable. Resizable jobs with compute unit resource requirements cannot increase job resource allocations, but can release allocated resources.

- [Configuring an exclusive queue](#)
- [Configuring a host to run one job at a time](#)
- [Submitting an exclusive job](#)
- [Configuring a compute unit exclusive queue](#)
- [Submitting a compute unit exclusive job](#)

Configuring an exclusive queue

Procedure

To configure an exclusive queue, set **EXCLUSIVE** in the queue definition (`lsb.queues`) to **Y**.
EXCLUSIVE=CU also configures the queue to accept exclusive jobs when no compute unit resource requirement is specified.

Configuring a host to run one job at a time

Procedure

To make sure exclusive jobs can be placed promptly, configure some single-processor hosts to run one job at a time. To do so, set **SLOTS=1** and **HOSTS=all** in `lsb.resources`.

Submitting an exclusive job

Procedure

To submit an exclusive job, use the **-x** option of **bsub** and submit the job to an exclusive queue.

Configuring a compute unit exclusive queue

Procedure

To configure an exclusive queue, set **EXCLUSIVE** in the queue definition (`lsb.queues`) to **CU[*cu_type*]**.
If no compute unit type is specified, the default compute unit type defined in **COMPUTE_UNIT_TYPES** (`lsb.params`) is used.

Submitting a compute unit exclusive job

Procedure

To submit an exclusive job, use the **-R** option of **bsub** and submit the job to a compute unit exclusive queue.
`bsub -R "cu[excl]" my_job`

Job dependency and job priority

LSF provides ways to manage job dependency and job priority to provide further control the order in which to schedule jobs.

- [Job dependency scheduling](#)
Sometimes, whether a job should start depends on the result of another job. For example, a series of jobs could process input data, run a simulation, generate images based on the simulation output, and finally, record the images on a high-resolution film output device. Each step can only be performed after the previous step finishes successfully, and all subsequent steps must be aborted if any step fails.

- [Job priorities](#)
LSF provides methods of controlling job priorities.

Job dependency scheduling

Sometimes, whether a job should start depends on the result of another job. For example, a series of jobs could process input data, run a simulation, generate images based on the simulation output, and finally, record the images on a high-resolution film output device. Each step can only be performed after the previous step finishes successfully, and all subsequent steps must be aborted if any step fails.

About job dependency scheduling

Some jobs may not be considered complete until some post-job processing is performed. For example, a job may need to exit from a post-execution job script, clean up job files, or transfer job output after the job completes.

In LSF, any job can be dependent on other LSF jobs. When you submit a job, you use **bsub -w** to specify a dependency expression, usually based on the job states of preceding jobs.

LSF will not place your job unless this dependency expression evaluates to TRUE. If you specify a dependency on a job that LSF cannot find (such as a job that has not yet been submitted), your job submission fails.

Syntax

bsub -w 'dependency_expression'

The dependency expression is a logical expression that is composed of one or more dependency conditions.

- To make dependency expression of multiple conditions, use the following logical operators:
 - && (AND)
 - || (OR)
 - ! (NOT)
- Use parentheses to indicate the order of operations, if necessary.
- Enclose the dependency expression in single quotes (') to prevent the shell from interpreting special characters (space, any logic operator, or parentheses). If you use single quotes for the dependency expression, use double quotes for quoted items within it, such as job names.
- Job names specify only your own jobs, unless you are an LSF administrator.
- Use double quotes (") around job names that begin with a number.
- In Windows, enclose the dependency expression in double quotes (") when the expression contains a space. For example:
 - `bsub -w "exit(678, 0)"` requires double quotes in Windows.
 - `bsub -w 'exit(678,0)'` can use single quotes in Windows.
- In the job name, specify the wildcard character (*) at the end of a string to indicate all jobs whose name begins with the string. For example, if you use `jobA*` as the job name, it specifies jobs named `jobA`, `jobA1`, `jobA_test`, `jobA.log`, etc.

Note:

Wildcard characters can only be used at the end of job name strings within the job dependency expression.

Multiple jobs with the same name

By default, if you use the job name to specify a dependency condition, and more than one of your jobs has the same name, all of your jobs that have that name must satisfy the test.

To change this behavior, set `JOB_DEP_LAST_SUB` in `lsb.params` to 1. Then, if more than one of your jobs has the same name, the test is done on the one submitted most recently.

- [Job dependency terminology](#)
- [Dependency conditions](#)

Job dependency terminology

- Job dependency: The start of a job depends on the state of other jobs.
- Parent jobs: Jobs that other jobs depend on.
- Child jobs: Jobs that cannot start until other jobs have reached a specific state.

Example: If job2 depends on job1 (meaning that job2 cannot start until job1 reaches a specific state), then job2 is the child job and job1 is the parent job.

Dependency conditions

The following dependency conditions can be used with any job:

- `done (job_ID | "job_name")`
- `ended (job_ID | "job_name")`
- `exit (job_ID [, [op] exit_code]`
- `exit ("job_name" [, [op] exit_code]`
- `external (job_ID | "job_name", "status_text")`
- `job_ID | "job_name"`
- `post_done (job_ID | "job_name")`
- `post_err (job_ID | "job_name")`
- `started (job_ID | "job_name")`

done

Syntax

```
done (job_ID | "job_name")
```

Description

The job state is DONE.

ended

Syntax

```
ended (job_ID | "job_name")
```

Description

The job state is EXIT or DONE.

exit

Syntax

```
exit (job_ID | "job_name" [, [operator] exit_code])
```

where *operator* represents one of the following relational operators:

>

>=

<

<=

==

`!=`

Description

The job state is EXIT, and the job's exit code satisfies the comparison test.

If you specify an exit code with no operator, the test is for equality (`==` is assumed).

If you specify only the job, any exit code satisfies the test.

Examples

```
exit (myjob)
```

The job named `myjob` is in the EXIT state, and it does not matter what its exit code was.

```
exit (678,0)
```

The job with job ID 678 is in the EXIT state, and terminated with exit code 0.

```
exit ("678", !=0)
```

The job named 678 is in the EXIT state, and terminated with any non-zero exit code.

external

Syntax

```
external (job_ID | "job_name", "status_text")
```

Specify the first word of the job status or message description (no spaces). Only the first word is evaluated.

Description

The job has the specified job status, or the text of the job's status begins with the specified word.

Job ID or job name

Syntax

```
job_ID | "job_name"
```

Description

If you specify a job without a dependency condition, the test is for the DONE state (LSF assumes the "done" dependency condition by default).

post_done

Syntax

```
post_done (job_ID | "job_name")
```

Description

The job state is POST_DONE (the post-processing of specified job has completed without errors).

post_err

Syntax

```
post_err (job_ID | "job_name")
```

Description

The job state is POST_ERR (the post-processing of specified job has completed with errors).

started

Syntax

```
started (job_ID | "job_name")
```

Description

The job state is:

- USUSP, SSUSP, DONE, or EXIT
- RUN and the job has a pre-execution command that is done.

Advanced dependency conditions

If you use job arrays, you can specify additional dependency conditions that only work with job arrays.

To use other dependency conditions with array jobs, specify elements of a job array in the usual way.

Job dependency examples

```
bsub -J "JobA" -w 'done(JobB)' command
```

The simplest kind of dependency expression consists of only one dependency condition. For example, if `JobA` depends on the successful completion of `JobB`, submit the job as shown.

```
-w 'done(312) && (started(Job2) || exit("99Job"))'
```

The submitted job will not start until the job with the job ID of 312 has completed successfully, and either the job named `Job2` has started, or the job named `99Job` has terminated abnormally.

```
-w "210"
```

The submitted job will not start unless the job named 210 is finished.

Job priorities

LSF provides methods of controlling job priorities.

Note: If you enable the `RELAX_JOB_DISPATCH_ORDER` parameter in the `lsb.params` file, which allows LSF to deviate from standard job prioritization policies, LSF might break the job dispatch order as specified by the job priority.

- [User-assigned job priority](#)
User-assigned job priority enables users to order their jobs in a queue. Submitted job order is the first consideration to determine job eligibility for dispatch. After you change the priority of your job relative to other jobs in the queue, it is still subject to all scheduling policies regardless of job priority. Jobs with the same priority are ordered first come first served.
- [Automatic job priority escalation](#)
Automatic job priority escalation automatically increases job priority of jobs that have been pending for a specified period of time. User-assigned job priority must also be configured.
- [Absolute priority scheduling \(APS\)](#)
Absolute job priority scheduling (APS) provides a mechanism to control the job dispatch order to prevent job starvation. APS provides administrators with detailed yet straightforward control of the job selection process. When configured in a queue, APS sorts pending jobs for dispatch according to a job priority value calculated based on several configurable job-related factors. Each job priority weighting factor can contain sub-factors. Factors and sub-factors can be independently assigned a weight.

User-assigned job priority

User-assigned job priority enables users to order their jobs in a queue. Submitted job order is the first consideration to determine job eligibility for dispatch. After you change the priority of your job relative to other jobs in the queue, it is still subject to all scheduling policies regardless of job priority. Jobs with the same priority are ordered first come first served.

The job owner can change the priority of their own jobs relative to all other jobs in the queue. LSF and queue administrators can change the priority of all jobs in a queue.

When with the `MAX_USER_PRIORITY` parameter is configured in the `lsb.params` file, user-assigned job priority is enabled for all queues in your cluster. You can also configure automatic job priority escalation to automatically increase the priority of jobs that have been pending for a specified period of time.

Considerations

The **bt** and **bbo** commands move jobs relative to other jobs of the same priority. These commands do not change job priority.

- [Configuring job priority](#)
- [Specifying job priority](#)

Configuring job priority

Procedure

1. To configure user-assigned job priority edit `lsb.params` and define `MAX_USER_PRIORITY`. This configuration applies to all queues in your cluster.

```
MAX_USER_PRIORITY=max_priority
```

where *max_priority* specifies the maximum priority that a user can assign to a job. Valid values are positive integers. Larger values represent higher priority; 1 is the lowest.

LSF and queue administrators can assign priority beyond *max_priority* for jobs they own.

2. Use **bparams -l** to display the value of `MAX_USER_PRIORITY`.

Example

`MAX_USER_PRIORITY=100` Specifies that 100 is the maximum job priority that can be specified by a user.

Specifying job priority

Procedure

Job priority is specified at submission using **bsub** and modified after submission using **bmod**. Jobs submitted without a priority are assigned the default priority of `MAX_USER_PRIORITY/2`.

```
bsub -sp priority bmod [-sp priority | -spn]  
job_ID
```

where:

- `-sp priority` specifies the job priority. Valid values for *priority* are any integers between 1 and `MAX_USER_PRIORITY` (displayed by **bparams -l**). Incorrect job priorities are rejected.
LSF and queue administrators can specify priorities beyond `MAX_USER_PRIORITY` for jobs they own.
- `-spn` sets the job priority to the default priority of `MAX_USER_PRIORITY/2` (displayed by **bparams -l**).

Automatic job priority escalation

Automatic job priority escalation automatically increases job priority of jobs that have been pending for a specified period of time. User-assigned job priority must also be configured.

As long as a job remains pending, LSF automatically increases the job priority beyond the maximum priority specified by `MAX_USER_PRIORITY`. Job priority is not increased beyond the value of `max_int` on your system.

If `TRACK_ELIGIBLE_PENDINFO` in `lsb.params` is set to `Y` or `y`, LSF increases the job priority for pending jobs as long as it is eligible for scheduling. LSF does not increase the job priority for ineligible pending jobs.

Pending job resize allocation requests for re-sizable jobs inherit the job priority from the original job. When the priority of the allocation request gets adjusted, the priority of the original job is adjusted as well. The job priority of a running job is adjusted when there is an associated resize request for allocation growth. **bjobs** displays the updated job priority.

If necessary, a new pending resize request is regenerated after the job gets dispatched. The new job priority is used.

For re-queued and rerun jobs, the dynamic priority value is reset. For migrated jobs, the existing dynamic priority value is carried forward. The priority is recalculated based on the original value.

- [Configuring job priority escalation](#)

Configuring job priority escalation

Procedure

1. To configure job priority escalation edit `lsb.params` and define `JOB_PRIORITY_OVER_TIME`.

```
JOB_PRIORITY_OVER_TIME=increment/interval
```

where:

- **increment** specifies the value used to increase job priority every *interval* minutes. Valid values are positive integers.
- **interval** specifies the frequency, in minutes, to *increment* job priority. Valid values are positive integers.

Note: User-assigned job priority must also be configured,

2. Use **bparams -l** to display the values of `JOB_PRIORITY_OVER_TIME`.

Example

```
JOB_PRIORITY_OVER_TIME=3/20
```

Specifies that every 20 minute *interval* *increment* to job priority of pending jobs by 3.

Absolute priority scheduling (APS)

Absolute job priority scheduling (APS) provides a mechanism to control the job dispatch order to prevent job starvation. APS provides administrators with detailed yet straightforward control of the job selection process. When configured in a queue, APS sorts pending jobs for dispatch according to a job priority value calculated based on several configurable job-related factors. Each job priority weighting factor can contain sub-factors. Factors and sub-factors can be independently assigned a weight.

- APS sorts only the jobs. Job scheduling is still based on configured LSF scheduling policies. LSF attempts to schedule and dispatch jobs by their order in the APS queue, but the dispatch order is not guaranteed.
- The job priority is calculated for pending jobs across multiple queues that are based on the sum of configurable factor values. Jobs are then ordered based on the calculated APS value.
- You can adjust the following values for APS factors:
 - A weight for scaling each job-related factor and sub-factor
 - Limits for each job-related factor and sub-factor
 - A grace period for each factor and sub factor

- To configure absolute priority scheduling (APS) across multiple queues, define APS queue groups. When you submit a job to any queue in a group, the job's dispatch priority is calculated by using the formula that is defined with the `APS_PRIORITY` parameter in the group's parent queue in the `lsb.queues` file.
- Administrators can also set a static system APS value for a job. A job with a system APS priority is guaranteed to have a higher priority than any calculated value. Jobs with higher system APS settings have priority over jobs with lower system APS settings.
- Administrators can use the `ADMIN` factor to manually adjust the calculated APS value for individual jobs.

Scheduling priority factors

To calculate the job priority, APS divides job-related information into several categories. Each category becomes a factor in the calculation of the scheduling priority. You can configure the weight, limit, and grace period of each factor to get the wanted job dispatch order.

LSF uses the weight of each factor to sum the value of each factor.

Factor weight

The weight of a factor expresses the importance of the factor in the absolute scheduling priority. The factor weight is multiplied by the value of the factor to change the factor value. A positive weight increases the importance of the factor, and a negative weight decreases the importance of a factor. Undefined factors have a weight of zero, which causes the factor to be ignored in the APS calculation.

Factor limit

The limit of a factor sets the minimum and maximum absolute value of each weighted factor. Factor limits must be positive values.

Factor grace period

Each factor can be configured with a grace period. The factor is only counted as part of the APS value when the job was pending for a long time and it exceeds the grace period.

APS_PRIORITY syntax

```
APS_PRIORITY=WEIGHT [[factor, value] [sub factor, value]...]...] LIMIT [[factor, value] [sub factor, value]...]...]
GRACE_PERIOD [[factor, value] [sub factor, value]...]...
```

Factors and sub factors

Factors	Sub factors	Metric
FS (user-based fair share factor)	The existing fair share feature tunes the dynamic user priority	<p>The fair share factor automatically adjusts the APS value based on dynamic user priority.</p> <p>The FAIRSHARE parameter must be defined in the queue. The FS factor is ignored for non-fair-share queues.</p> <p>The FS factor is influenced by the following fair share parameters that are defined in the <code>lsb.queues</code> or <code>lsb.params</code> file:</p> <ul style="list-style-type: none"> • CPU_TIME_FACTOR • FWD_JOB_FACTOR • RUN_TIME_FACTOR • RUN_JOB_FACTOR • HIST_HOURS
RSRC (resource factors)	PROC	Requested tasks are the maximum of bsub -n min_task, max_task , the min of bsub -n min , or the value of the TASKLIMIT parameter in the <code>lsb.queues</code> file.
	MEM	<p>Total real memory requested (in MB or in units set in the LSF_UNIT_FOR_LIMITS parameter in the <code>lsf.conf</code> file).</p> <p>Memory requests appearing to the right of a <code> </code> symbol in a usage string are ignored in the APS calculation.</p> <p>For multi-phase memory reservation, the APS value is based on the first phase of reserved memory.</p>

Factors	Sub factors	Metric
	SWAP	Total swap space requested (in MB or in units set in the LSF_UNIT_FOR_LIMITS parameter in the lsf.conf file). As with MEM, swap space requests appearing to the right of a symbol in a usage string are ignored.
WORK (job attributes)	JPRIORITY	The job priority that is specified by: <ul style="list-style-type: none"> • Default that is specified by half of the value of the MAX_USER_PRIORITY parameter in the lsb.params file • Users with bsub -sp or bmod -sp • Automatic priority escalation with the JOB_PRIORITY_OVER_TIME parameter in the lsb.params file <p>If the TRACK_ELIGIBLE_PENDINFO parameter in the lsb.params file is set to Y or y, LSF increases the job priority for pending jobs as long as it is eligible for scheduling. LSF does not increase the job priority for ineligible pending jobs.</p>
	QPRIORITY	The priority of the submission queue.
APP		Set the priority factor at the application profile level by specifying the PRIORITY parameter in the lsb.applications file. The APP_PRIORITY factor is added to the calculated APS value to change the factor value. The APP_PRIORITY factor applies to the entire job.
USER		Set the priority factor for users by specifying the PRIORITY parameter in the User section of the lsb.users file. The USER_PRIORITY factor is added to the calculated APS value to change the factor value. The USER_PRIORITY factor applies to the entire job.
UG		Set the priority factor for user groups by specifying the PRIORITY parameter in the UserGroup section of the lsb.users file. The UG_PRIORITY factor is added to the calculated APS value to change the factor value. The UG_PRIORITY factor applies to the entire job. LSF uses the priority of the user group as specified in the bsub -G option.
ADMIN		Administrators use bmod -aps to set this sub factor value for each job. A positive value increases the APS. A negative value decreases the APS. The ADMIN factor is added to the calculated APS value to change the factor value. The ADMIN factor applies to the entire job. You cannot configure separate weight, limit, or grace period factors. The ADMIN factor takes effect as soon as it is set.

Where LSF gets the job information for each factor

Factor or sub factor	Gets job information from...
MEM	<p>The value for jobs that are submitted with -R "rusage[mem]"</p> <p>For compound resource requirements submitted with -R "n1*{rusage[mem1]} + n2*{rusage[mem2]}" the value of MEM depends on whether resources are reserved per slot.</p> <ul style="list-style-type: none"> • If RESOURCE_RESERVE_PER_TASK=N, then MEM=mem1+mem2 • If RESOURCE_RESERVE_PER_TASK=Y, then MEM=n1*mem1+n2*mem2 <p>For alternative resource requirements, use a plug-in that considers all alternatives and uses the maximum value for the resource under consideration (SWP or MEM).</p>
SWAP	<p>The value for jobs that are submitted with the -R "rusage[swp]" option</p> <p>For compound and alternative resource requirements, SWAP is determined in the same manner as MEM.</p>

Factor or sub factor	Gets job information from...
PROC	<p>The value of <i>n</i> for jobs that are submitted with the bsub -n command (min_task, max_task), or the value of the TASKLIMIT parameter in the lsb.queues file</p> <p>Task limits can be specified at the job-level (bsub -n), the application-level (TASKLIMIT), and at the queue-level (TASKLIMIT). Job-level limits (bsub -n) override application-level TASKLIMIT, which overrides queue-level TASKLIMIT. Job-level limits must fall within the maximum and minimum limits of the application profile and the queue.</p> <p>Compound resource requirements by their nature express the number of processors a job requires. The minimum number of processors that are requested by way of job-level (bsub -n), application-level (TASKLIMIT), and queue-level (TASKLIMIT) must be equal and possibly greater than the number of processors that are requested through the resource requirement. If the final term of the compound resource requirement does not specify a number of processors, then the relationship is equal to or greater than. If the final term of the compound resource requirement does specify a number of processors, then the relationship is equal to, and the maximum number of processors that are requested must be equal to the minimum requested. LSF checks only that the default value supplied in TASKLIMIT (the first value of a pair or middle value of three values) is a multiple of a block. Maximum or minimum TASKLIMIT does not need to be a multiple of the block value.</p> <p>Alternative resource requirements might not specify the number of processors a job requires.</p> <p>The minimum number of processors that are requested by way of job-level (bsub -n command), application-level (the TASKLIMIT parameter in the lsb.applications file), and queue-level (the TASKLIMIT parameter in the lsb.queues file) must be less than or equal the minimum that is implied through the resource requirement.</p> <p>The maximum number of processors that are requested by way of job-level (the bsub -n command), application-level (the TASKLIMIT parameter in the lsb.applications file), and queue-level (the TASKLIMIT parameter in the lsb.queues file) must be equal to or greater than the maximum implied through the resource requirement. Any alternative that does not specify the number of processors is assumed to request the range from minimum to maximum, or request the default number of processors.</p>
JPPRIORITY	The dynamic priority of the job, which is updated every scheduling cycle and escalated by interval that is defined in the JOB_PRIORITY_OVER_TIME parameter defined in the lsb.params file
QPPRIORITY	The priority of the job submission queue
FS	The fair share priority value of the submission user
APP	The priority of the application profile
USER	The priority of the user
UG	The priority of the user group

- [Enabling absolute priority scheduling](#)
- [Modifying the system APS value \(bmod\)](#)
- [Configuring APS across multiple queues](#)
- [Job priority behavior](#)

Enabling absolute priority scheduling

Procedure

1. Ensure that the absolute priority scheduling plug-in (**schmod_aps**) is enabled in lsb.modules.
2. Configure APS_PRIORITY in an absolute priority queue in lsb.queues:

```
APS_PRIORITY=WEIGHT[[factor, value][subfactor, value]...]...] LIMIT[[factor, value][subfactor, value]...]...]  
GRACE_PERIOD[[factor, value][subfactor, value]...]...
```

Pending jobs in the queue are ordered according to the calculated **APS** value.

If weight of a sub-factor is defined, but the weight of parent factor is not defined, the parent factor weight is set as 1.

The WEIGHT and LIMIT factors are floating-point values. Specify a *value* for GRACE_PERIOD in seconds (*values*), minutes (*valuem*), or hours (*valueh*).

The default unit for grace period is hours.

For example, the following sets a grace period of ten hours for the MEM factor, ten minutes for the JPPRIORITY factor, ten seconds for the QPPRIORITY factor, and ten hours (default) for the RSRC factor:

```
GRACE_PERIOD [ [MEM, 10h] [JPPRIORITY, 10m] [QPPRIORITY, 10s] [RSRC, 10] ]
```

Note: You cannot specify zero for the WEIGHT, LIMIT, and GRACE_PERIOD of any factor or sub-factor. APS queues cannot configure cross-queue fair share (FAIRSHARE_QUEUES) or host-partition fair share.

Modifying the system APS value (bmod)

About this task

The absolute scheduling priority for a newly submitted job is dynamic. Job priority is calculated and updated based on formula specified by APS_PRIORITY in the absolute priority queue.

You must be an administrator to modify the calculated APS value.

Procedure

1. Run **bmod job_ID** to manually override the calculated APS value.
2. Run **bmod -apns job_ID** to undo the previous **bmod -aps** setting.

Assigning a static system priority and ADMIN factor value

Procedure

Run **bmod -aps "system=value"** to assign a static job priority for a pending job.
The value cannot be zero.

In this case, job's absolute priority is not calculated. The system APS priority is guaranteed to be higher than any calculated APS priority value. Jobs with higher system APS settings have priority over jobs with lower system APS settings.

The system APS value set by **bmod -aps** is preserved after **mbatchd** reconfiguration or **mbatchd** restart.

Using the ADMIN factor to adjust the APS value

Procedure

use **bmod -aps "admin=value"** to change the calculated APS value for a pending job.
The ADMIN factor is added to the calculated APS value to change the factor value. The absolute priority of the job is recalculated. The value cannot be zero .

A **bmod -aps** command always overrides the last **bmod -aps** commands

The ADMIN APS value set by **bmod -aps** is preserved after **mbatchd** reconfiguration or **mbatchd** restart.

Example bmod output

The following commands change the APS values for jobs 313 and 314:

```
bmod -aps "system=10" 313
Parameters of job <313> are being changed
bmod -aps "admin=10.00" 314
Parameters of job <314> are being changed
```

Viewing modified APS values

Procedure

1. Run **bjobs -aps** to see the effect of the changes:

```
bjobs -aps
JOBID   USER   STAT   QUEUE   FROM_HOST EXEC_HOST   JOB_NAME   SUBMIT_TIME   APS
313     user1   PEND   owners  hostA             myjob   Feb 12 01:09   (10)
321     user1   PEND   owners  hostA             myjob   Feb 12 01:09   -
314     user1   PEND   normal  hostA             myjob   Feb 12 01:08  109.00
312     user1   PEND   normal  hostA             myjob   Feb 12 01:08   99.00
315     user1   PEND   normal  hostA             myjob   Feb 12 01:08   99.00
316     user1   PEND   normal  hostA             myjob   Feb 12 01:08   99.00
```

2. Run **bjobs -l** to show APS values modified by the administrator:

```
bjobs -l
Job <313>, User <user1>, Project <default>, Service Class <SLASamples>, Status <RUN>,
Queue <normal>, Command <myjob>, System Absolute Priority <10> ...
Job <314>, User <user1>, Project <default>, Status <PEND>, Queue <normal>,
Command <myjob>, Admin factor value <10> ...
```

3. Use **bhist -l** to see historical information about administrator changes to APS values.

For example, after running these commands:

```
a. bmod -aps "system=10" 108
b. bmod -aps "admin=20" 108
c. bmod -apsn 108
```

bhist -l shows the sequence changes to job 108:

```
bhist -l
Job <108>, User <user1>, Project <default>, Command <sleep 10000>
Tue Feb 23 15:15:26 2010: Submitted from host <HostB>, to
Queue <normal>, CWD </scratch/user1>;
Tue Feb 23 15:15:40 2010: Parameters of Job are changed:
    Absolute Priority Scheduling factor string changed to : system=10;
Tue Feb 23 15:15:48 2010: Parameters of Job are changed:
    Absolute Priority Scheduling factor string changed to : admin=20;
Tue Feb 23 15:15:58 2010: Parameters of Job are changed:
    Absolute Priority Scheduling factor string deleted;
Summary of time in seconds spent in various states by Tue Feb 23 15:16:02 2010
PENDING    PSUSP      RUN       USUSP      SSUSP      UNKWN      TOTAL
    36         0         0         0         0         0         36
...
```

Configuring APS across multiple queues

Procedure

Use **QUEUE_GROUP** in an absolute priority queue in **lsb.queues** to configure APS across multiple queues.

When APS is enabled in the queue with **APS_PRIORITY**, the **FAIRSHARE_QUEUES** parameter is ignored. The **QUEUE_GROUP** parameter replaces **FAIRSHARE_QUEUES**, which is obsolete in LSF 7.0.

For example, you want to schedule jobs from the normal queue and the short queue, factoring the job priority (weight of one) and queue priority (weight of ten) in the APS value:

```
Begin Queue
QUEUE_NAME   = normal
PRIORITY     = 30
NICE         = 20
APS_PRIORITY = WEIGHT [[JPRIORITY, 1] [QPRIORITY, 10]]
QUEUE_GROUP  = short
DESCRIPTION  = For normal low priority jobs, running only if hosts are lightly loaded.
End Queue
```

```
...
Begin Queue
QUEUE_NAME = short
PRIORITY   = 20
NICE       = 20
End Queue
```

The APS value for jobs from the normal queue and the short queue are: calculated as:

```
APS_PRIORITY = 1 * (1 * job_priority + 10 * queue_priority)
```

The first 1 is the weight of the WORK factor; the second 1 is the weight of the job priority sub-factor; the 10 is the weight of queue priority sub-factor.

If you want the job priority to increase based on the pending time, you must configure JOB_PRIORITY_OVER_TIME parameter in the lsb.params.

Example

Extending the example, you now want to add user-based fair share with a weight of 100 to the APS value in the normal queue:

```
Begin Queue
QUEUE_NAME = normal
PRIORITY   = 30
NICE       = 20
FAIRSHARE = USER SHARES [[user1, 5000] [user2, 5000] [others, 1]]
APS_PRIORITY = WEIGHT [[JPPRIORITY, 1] [QPPRIORITY, 10] [FS, 100]]
QUEUE_GROUP = short
DESCRIPTION = For normal low priority jobs, running only if hosts are lightly loaded.
End Queue
```

The APS value is now calculated as

```
APS_PRIORITY = 1 * (1 * job_priority + 10 * queue_priority) + 100 * user_priority
```

Finally, you now to add swap space to the APS value calculation. The APS configuration changes to:

```
APS_PRIORITY = WEIGHT [[JPPRIORITY, 1] [QPPRIORITY, 10] [FS, 100] [SWAP, -10]]
```

And the APS value is now calculated as

```
APS_PRIORITY = 1 * (1 * job_priority + 10 * queue_priority)
+ 100 * user_priority + 1 * (-10 * SWAP)
```

Viewing pending job order by the APS value

Procedure

Run **bjobs -aps** to see APS information for pending jobs in the order of absolute scheduling priority.

The order that the pending jobs are displayed is the order in which the jobs are considered for dispatch.

The APS value is calculated based on the current scheduling cycle, so jobs are not guaranteed to be dispatched in this order.

Pending jobs are ordered by APS value. Jobs with system APS values are listed first, from highest to lowest APS value. Jobs with calculated APS values are listed next ordered from high to low value. Finally, jobs not in an APS queue are listed. Jobs with equal APS values are listed in order of submission time.

Results

If queues are configured with the same priority, **bjobs -aps** may not show jobs in the correct expected dispatch order. Jobs may be dispatched in the order the queues are configured in lsb.queues. You should avoid configuring queues with the same priority.

Example bjobs -aps output

The following example uses this configuration;

- The APS only considers the job priority and queue priority for jobs from normal queue (priority 30) and short queue (priority 20)
 - `APS_PRIORITY = WEIGHT [[Q_PRIORITY, 10] [J_PRIORITY, 1]]`
 - `QUEUE_GROUP = short`
- Priority queue (40) and idle queue (15) do not use APS to order jobs
- `JOB_PRIORITY_OVER_TIME=5/10` in `lsb.params`
- `MAX_USER_PRIORITY=100` in `lsb.params`

bjobs -aps was run at 14:41:

```
bjobs -aps
```

JOBID	USER	STAT	QUEUE	FROM_HOST	JOB_NAME	SUBMIT_TIME	APS
15	User2	PEND	priority	HostB	myjob	Dec 21 14:30	-
22	User1	PEND	Short	HostA	myjob	Dec 21 14:30	(60)
2	User1	PEND	Short	HostA	myjob	Dec 21 11:00	360
12	User2	PEND	normal	HostB	myjob	Dec 21 14:30	355
4	User1	PEND	Short	HostA	myjob	Dec 21 14:00	270
5	User1	PEND	Idle	HostA	myjob	Dec 21 14:01	-

For job 2, $APS = 10 * 20 + 1 * (50 + 220 * 5 / 10) = 360$ For job 12, $APS = 10 * 30 + 1 * (50 + 10 * 5 / 10) = 355$ For job 4, $APS = 10 * 20 + 1 * (50 + 40 * 5 / 10) = 270$

Viewing APS configuration for a queue

Procedure

Run **bqueues -l** to see the current APS information for a queue:

```
bqueues -l normal
```

QUEUE: normal

-- No description provided. This is the default queue.

PARAMETERS/STATISTICS

PRIO	NICE	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SSUSP	USUSP	RSV
500	20	Open:Active	-	-	-	-	0	0	0	0	0	0

SCHEDULING PARAMETERS

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

SCHEDULING POLICIES: FAIRSHARE APS_PRIORITY

APS_PRIORITY:

	WEIGHT FACTORS	LIMIT FACTORS	GRACE PERIOD
FAIRSHARE	10000.00	-	-
RESOURCE	101010.00	-	1010h
PROCESSORS	-10.01	-	-
MEMORY	1000.00	20010.00	3h
SWAP	10111.00	-	-
WORK	1.00	-	-
JOB PRIORITY	-999999.00	10000.00	4131s
QUEUE PRIORITY	10000.00	10.00	-

USER_SHARES: [user1, 10]

SHARE_INFO_FOR: normal/

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME
user1	10	3.333	0	0	0.0	0

USERS: all

HOSTS: all

REQUEUE_EXIT_VALUES: 10

Job priority behavior

Fair share

The default user-based fair share can be a factor in **APS** calculation by adding the **FS** factor to **APS_PRIORITY** in the queue.

- **APS** cannot be used together with DISPATCH_ORDER=QUEUE.
- **APS** cannot be used together with cross-queue fair share (FAIRSHARE_QUEUES). The QUEUE_GROUP parameter replaces FAIRSHARE_QUEUES, which is obsolete in LSF 7.0.
- **APS** cannot be used together with queue-level fair share or host-partition fair share.

FCFS (first come first serve)

APS overrides the job sort result of FCFS.

SLA scheduling

APS cannot be used together with time-based SLAs with velocity, decline, or throughput goals.

Job re-queuing

All requeued jobs are treated as newly submitted jobs for **APS** calculation. The job priority, system, and **ADMIN APS** factors are reset on re-queue.

Rerun jobs

Rerun jobs are not treated the same as requeued jobs. A job typically reruns because the host failed, not through some user action (like job re-queue), so the job priority is not reset for rerun jobs.

Job migration

Suspended (**bstop**) jobs and migrated jobs (**bmig**) are always scheduled before pending jobs. For migrated jobs, LSF keeps the existing job priority information.

If LSB_REQUEUE_TO_BOTTOM and LSB_MIG2PEND are configured in lsf.conf, the migrated jobs keep their **APS** information. When LSB_REQUEUE_TO_BOTTOM and LSB_MIG2PEND are configured, the migrated jobs need to compete with other pending jobs based on the **APS** value. If you want to reset the **APS** value, then you should use **brequeue**, not **bmig**.

Resource reservation

The resource reservation is based on queue policies. The **APS** value does not affect current resource reservation policy.

Preemption

The preemption is based on queue policies. The **APS** value does not affect the current preemption policy.

Chunk jobs

The first chunk job to be dispatched is picked based on the **APS** priority. Other jobs in the chunk are picked based on the APS priority and the default chunk job scheduling policies.

The following job properties must be the same for all chunk jobs:

- Submitting user
- Resource requirements
- Host requirements
- Queue or application profile
- Job priority

Backfill scheduling

Not affected.

Advance reservation

Not affected.

Resizable jobs

For new resizable job allocation requests, the resizable job inherits the **APS** value from the original job. The subsequent calculations use factors as follows:

Factor or sub-factor	Behavior
FAIRSHARE	<p>Resizable jobs submitting into fair share queues or host partitions are subject to fair share scheduling policies. The dynamic priority of the user who submitted the job is the most important criterion. LSF treats pending resize allocation requests as a regular job and enforces the fair share user priority policy to schedule them.</p> <p>The dynamic priority of users depends on:</p> <ul style="list-style-type: none">• Their share assignment• The slots their jobs are currently consuming• The resources their jobs consumed in the past• The adjustment made by the fair share plug-in (libfairshareadjust.*) <p>Resizable job allocation changes affect the user priority calculation if RUN_JOB_FACTOR is greater than zero (0). Resize add requests increase number of slots in use and decrease user priority. Resize release requests decrease number of slots in use, and increase user priority. The faster a resizable job grows, the lower the user priority is, the less likely a pending allocation request can get more slots.</p>
MEM	Use the value inherited from the original job
PROC	Use the MAX value of the resize request
SWAP	Use the value inherited from the original job
JPRIORITY	<p>Use the value inherited from the original job. If the automatic job priority escalation is configured, the dynamic value is calculated.</p> <p>For a requeued and rerun resizable jobs, the JPRIORITY is reset, and the new APS value is calculated with the new JPRIORITY.</p> <p>For migrated resizable job, the JPRIORITY is carried forward, and the new APS value is calculated with the JPRIORITY continued from the original value.</p>
QPRIORITY	Use the value inherited from the original job
ADMIN	Use the value inherited from the original job

Job re-queue and job rerun

- [About job re-queuing](#)
A networked computing environment is vulnerable to any failure or temporary conditions in network services or processor resources. For example, you might get NFS stale handle errors, disk full errors, process table full errors, or network connectivity problems. Your application can also be subject to external conditions such as a software license problems, or an occasional failure due to a bug in your application.
- [Automatic job reruns](#)

About job re-queuing

A networked computing environment is vulnerable to any failure or temporary conditions in network services or processor resources. For example, you might get NFS stale handle errors, disk full errors, process table full errors, or network connectivity problems. Your application can also be subject to external conditions such as a software license problems, or an occasional failure due to a bug in your application.

Such errors are temporary and probably happen at one time but not another, or on one host but not another. You might be upset to learn all your jobs exited due to temporary errors and you did not know about it until 12 hours later.

LSF provides a way to automatically recover from temporary errors. You can configure certain exit values such that in case a job exits with one of the values, the job is automatically re-queued as if it had not yet been dispatched. This job is then be retried later. It is also possible for you to configure your queue such that a re-queued job is not scheduled to hosts on which the job had previously failed to run.

- [Automatic job re-queuing](#)
You can configure a queue to automatically re-queue a job if it exits with a specified exit value.
- [Configuring job-level automatic re-queuing](#)
- [Configuring reverse re-queuing](#)
- [Exclusive job re-queuing](#)
You can configure automatic job re-queue so that a failed job is not rerun on the same host.
- [Re-queuing a job](#)

Automatic job re-queuing

You can configure a queue to automatically re-queue a job if it exits with a specified exit value.

- The job is re-queued to the head of the queue from which it was dispatched, unless the `LSB_REQUEUE_TO_BOTTOM` parameter in `lsf.conf` is set.
- When a job is re-queued, LSF does not save the output from the failed run.
- When a job is re-queued, LSF does not notify the user by sending mail.
- A job terminated by a signal is not re-queued.

The reserved keyword `all` specifies all exit codes. Exit codes are typically between 0 and 255. Use a tilde (~) to exclude specified exit codes from the list.

For example:

```
REQUEUE_EXIT_VALUES=all ~1 ~2 EXCLUDE(9)
```

Jobs exited with all exit codes except 1 and 2 are re-queued. Jobs with exit code 9 are re-queued so that the failed job is not rerun on the same host (exclusive job re-queue).

- [Configuring automatic job re-queuing](#)

Configuring automatic job re-queuing

Procedure

To configure automatic job re-queue, set `REQUEUE_EXIT_VALUES` in the queue definition (`lsb.queues`) or in an application profile (`lsb.applications`) and specify the exit codes that cause the job to be re-queued. Application-level exit values override queue-level values. Job-level exit values (**`bsub -Q`**) override application-level and queue-level values.

```
Begin Queue
...
REQUEUE_EXIT_VALUES = 99 100
...
End Queue
```

This configuration enables jobs that exit with 99 or 100 to be re-queued.

Controlling how many times a job can be re-queued

About this task

By default, if a job fails and its exit value falls into `REQUEUE_EXIT_VALUES`, LSF re-queues the job automatically. Jobs that fail repeatedly are re-queued indefinitely by default.

Procedure

To limit the number of times a failed job is re-queued, set `MAX_JOB_REQUEUE` cluster wide (`lsb.params`), in the queue definition (`lsb.queue`s), or in an application profile (`lsb.applications`). Specify an integer greater than zero.

`MAX_JOB_REQUEUE` in `lsb.applications` overrides `lsb.queue`s, and `lsb.queue`s overrides `lsb.params` configuration.

Results

When `MAX_JOB_REQUEUE` is set, if a job fails and its exit value falls into `REQUEUE_EXIT_VALUES`, the number of times the job has been re-queued is increased by 1 and the job is re-queued. When the re-queue limit is reached, the job is suspended with `PSUSP` status. If a job fails and its exit value is not specified in `REQUEUE_EXIT_VALUES`, the job is not re-queued.

Viewing the re-queue retry limit

Procedure

1. Run **`bjobs -l`** to display the job exit code and reason if the job re-queue limit is exceeded.
2. Run **`bhist -l`** to display the exit code and reason for finished jobs if the job re-queue limit is exceeded.

Results

The job re-queue limit is recovered when LSF is restarted and reconfigured. LSF replays the job re-queue limit from the `JOB_STATUS` event and its pending reason in `lsb.events`.

Configuring job-level automatic re-queuing

Procedure

Use **`bsub -Q`** to submit a job that is automatically re-queued if it exits with the specified exit values.

Use spaces to separate multiple exit codes. The reserved keyword `all` specifies all exit codes. Exit codes are typically between 0 and 255. Use a tilde (`~`) to exclude specified exit codes from the list.

Job-level re-queue exit values override application-level and queue-level configuration of the parameter `REQUEUE_EXIT_VALUES`, if defined.

Jobs running with the specified exit code share the same application and queue with other jobs.

For example:

```
bsub -Q "all ~1 ~2 EXCLUDE(9)" myjob
```

Jobs exited with all exit codes except 1 and 2 are re-queued. Jobs with exit code 9 are re-queued so that the failed job is not rerun on the same host (exclusive job re-queue).

Enabling exclusive job re-queuing

Procedure

Define an exit code as `EXCLUDE(exit_code)` to enable exclusive job re-queue.
Exclusive job re-queue does not work for parallel jobs.

Note: If **mbatchd** is restarted, it does not remember the previous hosts from which the job exited with an exclusive re-queue exit code. In this situation, it is possible for a job to be dispatched to hosts on which the job has previously exited with an exclusive exit code.

Modifying re-queue exit values

Procedure

Use **bmod -Q** to modify or cancel job-level re-queue exit values.

bmod -Q does not affect running jobs. For re-runnable and re-queue jobs, **bmod -Q** affects the next run.

Multicuster job forwarding model

For jobs sent to a remote cluster, arguments of **bsub -Q** take effect on remote clusters.

Multicuster lease model

The arguments of **bsub -Q** apply to jobs running on remote leased hosts as if they are running on local hosts.

Configuring reverse re-queuing

About this task

By default, if you use automatic job re-queue, jobs are re-queued to the head of a queue. You can have jobs re-queued to the bottom of a queue instead. The job priority does not change.

You must already use automatic job re-queue (`REQUEUE_EXIT_VALUES` in `lsb.queues`).

To configure reverse re-queue:

Procedure

1. Set `LSB_REQUEUE_TO_BOTTOM` in `lsf.conf` to 1.
2. Reconfigure the cluster:
 - a. `lsadmin reconfig`
 - b. `badmin mbdrestart`

Exclusive job re-queuing

You can configure automatic job re-queue so that a failed job is not rerun on the same host.

Limitations

- If **mbatchd** is restarted, this feature might not work properly, since LSF forgets which hosts have been excluded. If a job ran on a host and exited with an exclusive exit code before **mbatchd** was restarted, the job could be dispatched to the same host again after **mbatchd** is restarted.
- Exclusive job re-queue does not work for multicuster jobs or parallel jobs
- A job terminated by a signal is not re-queued
- [Configuring exclusive job re-queuing](#)

Configuring exclusive job re-queuing

Procedure

Set `REQUEUE_EXIT_VALUES` in the queue definition (`lsb.queueues`) and define the exit code using parentheses and the keyword `EXCLUDE`:

```
EXCLUDE(exit_code...)
```

where *exit_code* has the following form:

```
"[all] [~number ...] | [number ...]"
```

The reserved keyword `all` specifies all exit codes. Exit codes are typically between 0 and 255. Use a tilde (`~`) to exclude specified exit codes from the list.

Jobs are re-queued to the head of the queue. The output from the failed run is not saved, and the user is not notified by LSF.

Results

When a job exits with any of the specified exit codes, it is re-queued, but it is not dispatched to the same host again.

Example

```
Begin Queue  
...  
REQUEUE_EXIT_VALUES=30 EXCLUDE(20) HOSTS=hostA hostB hostC  
...  
End Queue
```

A job in this queue can be dispatched to `hostA`, `hostB` or `hostC`.

If a job running on `hostA` exits with value 30 and is re-queued, it can be dispatched to `hostA`, `hostB`, or `hostC`. However, if a job running on `hostA` exits with value 20 and is re-queued, it can only be dispatched to `hostB` or `hostC`.

If the job runs on `hostB` and exits with a value of 20 again, it can only be dispatched on `hostC`. Finally, if the job runs on `hostC` and exits with a value of 20, it cannot be dispatched to any of the hosts, so it is pending forever.

Re-queuing a job

About this task

You can use `brequeue` to kill a job and re-queue it. When the job is re-queued, it is assigned the `PEND` status and the job's new position in the queue is after other jobs of the same priority.

Procedure

To re-queue one job, use **`brequeue`**.

- You can only use **`brequeue`** on running (`RUN`), user-suspended (`USUSP`), or system-suspended (`SSUSP`) jobs.
- Users can only re-queue their own jobs. Only root and LSF administrator can re-queue jobs that are submitted by other users.
- You cannot use **`brequeue`** on interactive batch jobs

Results

brequeue 109

LSF kills the job with job ID 109, and re-queues it in the PEND state. If job 109 has a priority of 4, it is placed after all the other jobs with the same priority.

brequeue -u User5 45 67 90

LSF kills and re-queues 3 jobs belonging to `User5`. The jobs have the job IDs 45, 67, and 90.

Automatic job reruns

Job re-queuing versus job rerunning

Automatic job re-queue occurs when a job finishes and has a specified exit code (usually indicating some type of failure).

Automatic job rerun occurs when the execution host becomes unavailable while a job is running. It does not occur if the job itself fails.

About job reruns

When a job is rerun or restarted, it is first returned to the queue from which it was dispatched with the same options as the original job. The priority of the job is set sufficiently high to ensure that the job gets dispatched before other jobs in the queue. The job uses the same job ID number. It is executed when a suitable host is available, and an email message is sent to the job owner informing the user of the restart.

Automatic job rerun can be enabled at the job level, by the user, or at the queue level, by the LSF administrator. If automatic job rerun is enabled, the following conditions cause LSF to rerun the job:

- The execution host becomes unavailable while a job is running
- The system fails while a job is running

When LSF reruns a job, it returns the job to the submission queue, with the same job ID. LSF dispatches the job as if it was a new submission, even if the job has been check-pointed.

Once job is rerun, LSF schedules re-sizable jobs based on their initial allocation request.

Execution host failures

If the execution host fails, LSF dispatches the job to another host. You receive a mail message informing you of the host failure and the re-queuing of the job.

LSF system failures

If the LSF system fails, LSF re-queues the job when the system restarts.

- [Configuring queue-level job reruns](#)
- [Submitting a re-runnable job](#)
- [Disabling a job from re-running](#)
- [Disabling post-execution for re-runnable jobs](#)

Configuring queue-level job reruns

Procedure

To enable automatic job rerun at the queue level, set `RERUNNABLE` in `lsb.queues` to `yes`.

Submitting a re-runnable job

Procedure

To enable automatic job rerun at the job level, use **bsub -r**.
Interactive batch jobs (**bsub -I**) cannot be re-runnable.

Disabling a job from re-running

Procedure

To disable automatic job rerun at the job level, use **bsub -rn**.

Disabling post-execution for re-runnable jobs

About this task

Running of post-execution commands upon restart of a re-runnable job may not always be desirable; for example, if the post-exec removes certain files, or does other cleanup that should only happen if the job finishes successfully.

Procedure

Use `LSB_DISABLE_RERUN_POST_EXEC=Y` in `lsf.conf` to prevent the post-exec from running when a job is rerun.

Predict job start time using a simulation-based estimator

LSF can predict an approximate start time for these pending jobs by using a simulation-based job start time estimator that runs on the management host and is triggered by the **mbatchd** daemon. The estimator uses a snapshot of the cluster (including the running jobs and available resources in the cluster) to simulate job scheduling behavior and determine when jobs finish and the pending jobs start. This gives users an idea of when their jobs are expected to start.

In clusters with long running parallel jobs, there may be a small number of long running jobs (that is, between 100 to 1000 jobs) pending in the queue for several days and these jobs may run for several days or weeks.

To use simulation-based estimation to predict start times, jobs must be submitted with either a run time limit (by using the **bsub -W** option or by submitting to a queue or application profile with a defined `RUNLIMIT` value) or an estimated run time (by using the **bsub -We** option or by submitting to an application profile with a defined `RUNTIME` value). LSF considers jobs without a run time limit or an estimated run time as never finished once they are dispatched to the simulation-based estimator. If both a run time limit and an estimated run time are specified for a job, the smaller value is used as the job's run time in the simulation-based estimator.

While simulating the job, the estimator assumes that the job will run and finish normally as defined by the run time limit or estimated run time, and that all resources required by the job are completely consumed.

To enable the simulation-based estimator, define `LSB_ENABLE_ESTIMATION=Y` in `lsf.conf`. When enabled, the estimator starts up five minutes after `mbatchd` starts or restarts. By default, the estimator provides predictions for the first 1000 jobs or for predicted start times up to one week in the future, whichever comes first. Estimation also ends when all pending jobs have prediction job start times.

Optionally, you can control the default values for when **mbatchd** stops the current round of estimation to balance the accuracy of the job start predictions against the computation effort on the management host. **mbatchd** stops the current round of estimation when the estimator reaches any one of the following estimation thresholds specified in `lsb.params`:

- **ESTIMATOR_MAX_JOBS_PREDICTION**: Specifies the number of pending jobs that the estimator predicts, which is 1000 by default.
- **ESTIMATOR_MAX_TIME_PREDICTION**: Specifies the amount of time into the future, in minutes, that a job is predicted to start before the estimator stops the current round of estimation. By default, the estimator stops once a job is predicted to start in one week (10080 minutes).
- **ESTIMATOR_MAX_RUNTIME_PREDICTION**: Specifies the amount of time that the estimator runs, up to the value of the **ESTIMATOR_SIM_START_INTERVAL** parameter. By default, the estimator stops once it has run for 30 minutes or the amount of time as specified by the **ESTIMATOR_SIM_START_INTERVAL** parameter, whichever is smaller.

You can also define the following optional parameters to further specify how the job start estimator runs or groups the job start times together:

- **ESTIMATOR_CPUS**: Defines a space-delimited list of management host CPU numbers where the job start time estimator processes can run. Use this parameter to bind the estimator processes to the specified CPUs, which reduces the impact of the estimator on the performance of **mbatchd** and **mbsched**.
- **ESTIMATOR_SIM_START_INTERVAL**: Specifies the amount of time, in minutes, since the start of the last round of the job start time estimation that **mbatchd** waits before triggering another round of estimation.

Viewing detailed estimation results

Run the **bjobs -l** command to view the results of the job start time estimation. The results are shown in the **ESTIMATION** section of the output.

If the job start time estimator successfully estimated a job start time and host allocation, the following details are shown in the **bjobs -l** command output:

```
ESTIMATION:
Thu Aug 27 17:33:53: Started simulation-based estimation;
Thu Aug 27 17:34:39: Simulated job start time <Thu Aug 27 18:00:02> on host(s)
                    <1* host1> <15* host2>;
```

The estimated start time of the job is 18:00:02 on August 27 and the jobs will be run on `host1` and `host2`.

In the IBM® Spectrum LSF multicluster capability job forwarding mode, the **bjobs -l** command shows the estimated start time on the execution cluster, and the host name includes the execution cluster name.

If the estimator could not estimate a job start time, **mbatchd** determines whether a bound estimate can be shown for the job. The bound estimate is the maximum simulation time in the future compared to the last completed round of simulation. For the jobs submitted or queued before this round of simulation, **bjobs -l** shows the lower bound of the estimated job start time (that is, the earliest possible job start time).

For example, the **bjobs -l** command output for the lower-bound estimated job start time is as follows:

```
ESTIMATION:
Thu Aug 27 17:33:53: Started simulation-based estimation;
Thu Aug 27 17:34:39: Simulated job cannot start before <Fri Aug 28 18:00:29>;
```

In this example, the last round of the completed simulation started at 17:33:53 on August 27 and the job to be estimated was submitted at 17:00:29 on the same day, but the simulation ended before it could predict the job start time of this particular job. When the simulation ended, the last successful prediction was for a job start time of 18:00:29 on August 28. Subsequent jobs that did not get an estimation cannot start before this time, so this time is the lower bound estimate.

Jobs that did not have a job start estimation or a lower bound estimate do not show an **ESTIMATION** section in the **bjobs -l** output.

View estimation results in a custom bjobs format

The **bjobs -o** command for customized output can display the results of the job start time estimation. Use the `estimated_sim_start_time` field name (or the alias `esstart_time`) to display the estimated start time of the job:

```
bjobs -o "jobid:7 stat:5 user:7 submit_time:20 estimated_sim_start_time:24"
JOBID   STAT   USER   SUBMIT_TIME   ESTIMATED_SIM_START_TIME
5       RUN    usera   Aug 26 11:20   Aug 26 13:30
6       PEND   usera   Aug 26 11:55   Aug 26 15:22
7       PEND   usera   Aug 26 12:25   Aug 27 18:00
```

Job affinity scheduling with host attributes

Create attributes for hosts and use these attributes for job affinity scheduling. Submit jobs and define host preferences based on which hosts have specific attributes.

Host attributes give you a flexible way of specifying host affinity preferences to give you more control over host selection for job scheduling. Use the **battr create** command to create attributes that are assigned to hosts. Use the **bsub -jobaff** command option when submitting jobs to define host preferences based on host attributes. If you specified a preferred attribute with **bsub -jobaff** and this attribute does not exist, LSF automatically creates the attribute on the execution hosts after dispatching the job to the hosts.

For example, you can specify that jobs can only run (or prefer to run) on hosts with a specific attribute or on hosts that do not possess a specific attribute. You can also specify that a job can only run (or prefer to run) on hosts or compute units that are already running a specific job, or on hosts or compute units that are not running a specific job.

- [Configuring host attributes for job affinity](#)
Configure job affinity by defining host attribute parameters in the lsb.params file.
- [Creating and managing host attributes for job affinity](#)
Manage host attributes for job affinity with the **battr** command.
- [Submitting jobs with host attributes for job affinity](#)
Use the **bsub -jobaff** command option to specify host attribute affinity preferences for job affinity scheduling.

Configuring host attributes for job affinity

Configure job affinity by defining host attribute parameters in the lsb.params file.

Procedure

Edit the lsb.params file to define the parameters for host attributes.

- Define the ATTR_CREATE_USERS parameter to specify the users that can create host attributes.
ATTR_CREATE_USERS=none | all | "user_name ..."

Specify a space-separated list of users that have permission to create attributes, or specify the **all** keyword to indicate that all LSF users can create attributes. If you specify the **none** keyword, the host attribute affinity feature is disabled and no users have permission to create job attributes. Users that are not specified in this parameter cannot create host attributes. By default, this parameter is set to **none**.

- Define the ATTR_MAX_NUM parameter to specify a maximum number of host attributes that can exist in the cluster. Cluster performance might be affected if there are too many host attributes that co-exist in the cluster. Specify a maximum number of host attributes to limit the decrease in cluster performance.

ATTR_MAX_NUM=integer

If the number of host attributes in the cluster reaches this value, LSF rejects any requests to create new attributes. By default, this parameter is set to 100.

- Define the ATTR_TTL parameter to specify a time to live (TTL) for newly-created host attributes.
ATTR_TTL=time_hours

ATTR_TTL=time_minutesm | M

When LSF creates a new host attribute, the time-to-live (TTL) of the attribute is set to this parameter value. When the attribute is used by a new job, the attribute's TTL is reset to the value of the ATTR_TTL parameter. Use the m or M keyword to indicate that the parameter value is in minutes, otherwise the parameter value is in hours. When the TTL reaches zero, the **mbatchd** daemon removes this attribute. The default TTL for host attributes is one hour.

- d. Optional: Enable the SAME_JOB_AFFINITY parameter to allow users to specify affinity preferences for jobs to run on the same host or compute unit as another job.

SAME_JOB_AFFINITY=Y | y

If enabled, users can use the samehost and samecu keywords with the **bsub -jobaff** command option when specifying affinity preferences for jobs to run on the same host or compute unit as another job. By default, this is disabled.

Creating and managing host attributes for job affinity

Manage host attributes for job affinity with the **battr** command.

Procedure

1. Use the **battr create** command to create host attributes.

```
battr create -m "host_name ..." [-d "description"] attr_name ...
```

This command creates the specified attributes on the specified hosts. Use a space to separate multiple attributes. Use the -m option to specify one or more hosts in which to create the attributes, and use a space to separate multiple hosts. You can specify the names in condensed host formats, but you cannot specify host groups, remote (lease-in) hosts, or client hosts. You can use the optional -d option to specify a description for the attributes.

```
battr create -m "hostA hostB" -d "Hosts in room 103" room103
```

2. Use the **battr delete** command to delete host attributes.

```
battr delete -m "host_name ..." | all attr_name ...
```

This command deletes the specified attributes from the specified hosts. Use a space to separate multiple attributes. Use the -m option to specify one or more hosts in which to create the attributes, and use a space to separate multiple hosts.

```
battr delete -m "hostA hostB" room103
```

3. Use the **battr show** command to show information on host attributes in the cluster.

```
battr show [-w] [-m "host_name ..."] [-u user_name] [attr_name ...]
```

You can show information on all host attributes or specify one or more attributes to display. Use a space to separate multiple attributes. Use the -w option to display attribute information in a wide format without truncating fields. Use the -m option to specify one or more hosts from which to show the attributes, otherwise this command shows attributes from all hosts in the cluster. Use the -u option to show attributes that are created by the specified user.

```
battr show -m "hostA hostB"
```

4. Use the **bhosts -attr** or **bhosts -l** command options to show information on attributes on the specified hosts.

Submitting jobs with host attributes for job affinity

Use the **bsub -jobaff** command option to specify host attribute affinity preferences for job affinity scheduling.

Procedure

1. Use no keyword (or the attribute keyword) with the **bsub -jobaff** command option to specify the host attribute affinity preferences for job scheduling.

```
bsub -jobaff "[! | # | ~]attribute_name ..."
```

```
bsub -jobaff "attribute([! | # | ~]attribute_name ...)"
```

This specifies preferences for hosts for job scheduling based on host attributes. Use a space to separate multiple attributes. Specifying the following special characters before the attribute name indicates the following preferences:

- Default (no special character): It is preferred for the selected host to have this attribute.
- `!`: It is preferred for the selected host to not have this attribute.
- `#`: It is mandatory for the selected host to have this attribute.
- `~`: It is mandatory for the selected host to not have this attribute.

The following command specifies that the job must be scheduled on hosts that have the room103 attribute.

```
bsub -jobaff "attribute(#room103)"
```

2. Use the `samehost` keyword with the **bsub -jobaff** command option to specify the preference for the job to run on the same host on which another job with the specified job ID runs.

```
bsub -jobaff "[! | # | ~]samehost(job_id)"
```

The `SAME_JOB_AFFINITY` parameter must be set to `Y` or `y` in the `lsb.params` file to use the `samehost` keyword. The job ID can be a simple job or array job element, but you cannot specify multiple job IDs or an array job ID. Specifying the following special characters before the `samehost` keyword indicates the following preferences:

- Default (no special character): It is preferred for the job to run on the same host on which the specified job runs.
- `!`: It is preferred for the job to not run on the same host on which the specified job runs.
- `#`: It is mandatory for the job to run on the same host on which the specified job runs.
- `~`: It is mandatory for the job to not run on the same host on which the specified job runs.

The following command specifies that the job must be scheduled on hosts that are also running the job with job ID 234.

```
bsub -jobaff "#samehost(234)"
```

3. Use the `samecu` keyword with the **bsub -jobaff** command option to specify the preference for the job to run on the same compute unit on which another job with the specified job ID runs.

```
bsub -jobaff "[! | # | ~]samecu(job_id)"
```

The `SAME_JOB_AFFINITY` parameter must be set to `Y` or `y` in the `lsb.params` file to use the `samecu` keyword. The job ID can be a simple job or array job element, but you cannot specify multiple job IDs or an array job ID. Specifying the following special characters before the `samecu` keyword indicates the following preferences:

- Default (no special character): It is preferred for the job to run on the same compute unit on which the specified job runs.
- `!`: It is preferred for the job to not run on the same compute unit on which the specified job runs.
- `#`: It is mandatory for the job to run on the same compute unit on which the specified job runs.
- `~`: It is mandatory for the job to not run on the same compute unit on which the specified job runs.

The following command specifies that the job must not be scheduled on a host in the same compute unit that is also running the job with job ID 234.

```
bsub -jobaff "~samecu(234)"
```

Control job execution

Use resource usage limits to control how much resource can be consumed by running jobs. Automatically suspend jobs based on the load conditions on the execution hosts. Use pre- and post-execution processing to run commands on an execution host before and after completion of a job. Use job starters to set up the runtime environment for a job. Job submission and execution controls use external, site-specific executable files to validate, modify, and reject jobs, transfer data, and modify the job execution environment.

- [Pre-execution and post-execution processing](#)

The pre- and post-execution processing feature provides a way to run commands on an execution host prior to and after completion of LSF jobs. Use pre-execution commands to set up an execution host with the required directories, files, environment, and user permissions. Use post-execution commands to define post-job processing such as cleaning up job files or transferring job output.

- [Job starters](#)

- [Job control actions](#)
Learn how to configure job control actions to override or augment the default job control actions.
- [Submit jobs as other users](#)
Use the **bsubmit** command with the `lsf.usermapping` configuration file to submit jobs as other users.
- [External job submission and execution controls](#)
The job submission and execution controls use external, site-specific executable files to validate, modify, and reject jobs; and to transfer data and modify the job execution environment.

Pre-execution and post-execution processing

The pre- and post-execution processing feature provides a way to run commands on an execution host prior to and after completion of LSF jobs. Use pre-execution commands to set up an execution host with the required directories, files, environment, and user permissions. Use post-execution commands to define post-job processing such as cleaning up job files or transferring job output.

- [About pre- and post-execution processing](#)
The pre- and post-execution processing feature consists of two types:
- [Configuration to enable pre- and post-execution processing](#)
The pre- and post-execution processing feature is enabled by defining at least one of the parameters in the list below at the application or queue level, or by using the `-E` option of the **bsub** command to specify a pre-execution command. In some situations, specifying a queue-level or application-level pre-execution command can have advantages over requiring users to use **bsub -E**. For example, license checking can be set up at the queue or application level so that users do not have to enter a pre-execution command every time they submit a job.
- [Pre- and post-execution processing behavior](#)
Job-based pre- and post-execution processing applies to both UNIX and Windows hosts. Host-based pre- and post-execution processing only applies to UNIX host.
- [Configuration to modify pre- and post-execution processing](#)
Configuration parameters modify various aspects of pre- and post-execution processing behavior by:
- [Pre- and post-execution processing commands](#)

About pre- and post-execution processing

The pre- and post-execution processing feature consists of two types:

- Job-based pre- and post-execution processing, which is intended for sequential jobs and runs only on the first execution host.
- Host-based pre- and post-execution processing, which is intended for parallel jobs and runs on all execution hosts.

You can use pre- and post-execution processing to run commands before a batch job starts or after it finishes. Typical uses of this feature include the following:

- Reserving resources such as tape drives and other devices not directly configurable in LSF
- Making job-starting decisions in addition to those directly supported by LSF
- Creating and deleting scratch directories for a job
- Customizing scheduling based on the exit code of a pre-execution command
- Checking availability of software licenses
- Assigning jobs to run on specific processors on SMP machines
- Transferring data files needed for job execution
- Modifying system configuration files before and after job execution
- Using a post-execution command to clean up a state left by the pre-execution command or the job

Any executable command line can serve as a pre-execution or post-execution command. By default, the commands run under the same user account, environment, home directory, and working directory as the job.

When `JOB_INCLUDE_POSTPROC` is defined in an application profile or `lsb.params`, a job is considered in RUN state while the job is in post exec stage (which is DONE state for regular jobs).

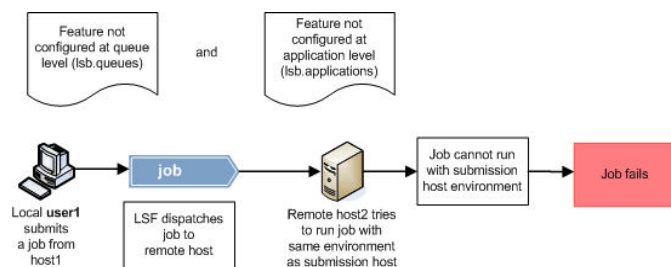
Job-based pre- and post-execution processing

Job-based pre-execution and post-execution commands can be defined at the queue, application, and job levels.

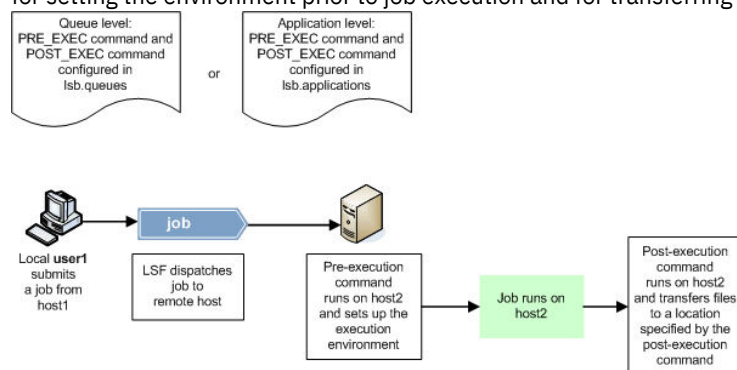
The command path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory, file name, and expanded values for %J (*job_ID*) and %I (*index_ID*).

When the job is resizable, job grow requests are ignored. However, job shrink requests can be processed. For either case, LSF does not invoke the job resized notification command.

The following illustration shows the default behavior (feature not enabled) of job-based pre- and post-execution processing:



The following example illustrates how job-based pre- and post-execution processing works at the queue or application level for setting the environment prior to job execution and for transferring resulting files after the job runs.



The table below provides the scope of job-based pre- and post-execution processing:

Applicability	Details
Operating system	<ul style="list-style-type: none">• UNIX• Windows• A mix of UNIX and Windows hosts
Dependencies	<ul style="list-style-type: none">• UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs.• On a Windows Server 2003, x64 Edition platform, users must have read and execute privileges for cmd.exe.
Limitations	<ul style="list-style-type: none">• Applies to batch jobs only (jobs submitted using the bsub command)

Host-based pre- and post-execution processing

Host-based pre- and post-execution processing is different from job-based pre- and post-execution processing in that it is intended for parallel jobs (you can also use this feature for sequential jobs) and is executed on all execution hosts, as opposed to only the first execution host. The purpose of this is to set up the execution hosts before all job-based pre-execution and other pre-processing which depend on host-based preparation, and clean up execution hosts after job-based post execution and other post-processing.

This feature can be used in a number of ways. For example:

- HPC sites can have multiple ways to check for system health before actually launching jobs, such as checking for host or node status, key file systems are mounted, infiniband is working, required directories, files, environment, and correct user permissions are set, etc.)
- Administrators can configure site specific policy to run host-based pre- and post-execution processing to set up ssh access to computer nodes. By default, ssh is disabled. However, with host-based pre- and post-execution processing, ssh access to the nodes allocated for the job can be enabled for the duration of job life cycle. This is required for debugging a parallel job on a non-first execution host and will not impact the overall cluster security policy.
- Administrators can configure host-based pre- and post-execution processing to create and later remove temporary working directories on each host.

You can define the host-based pre- and post-execution processing at the application level and the queue level. Failure handling is also supported.

There are two ways to enable host-based pre- and post-execution processing for a job:

- Configure HOST_PRE_EXEC and HOST_POST_EXEC in lsb.queues.
- Configure HOST_PRE_EXEC and HOST_POST_EXEC in lsb.applications.

When configuring host-based pre- and post-execution processing, note the following:

- Host-based pre- and post-execution processing is only supported on UNIX.
- Host-based pre- and post-execution processing does not support the return of some environment variables in output and the setting of those environment variables for the job.
- If a job is in the host-based pre-execution processing stage, **sbatchd** rejects any signals that are not termination signals and requests that the signal be sent again. If the job is in the host-based post-execution processing stage, job signals are rejected or ignored no matter how JOB_INCLUDE_POSTPROC is defined.
- You cannot use the default value for JOB_PREPROC_TIMEOUT or JOB_POSTPROC_TIMEOUT for host-based pre- and post-execution processing. Configure a value based on how long it would take for host-based pre- and post-execution processing to run.
- Checkpointing can not be performed until host-based pre-execution processing is finished. During that time, **sbatchd** returns a retry error.
- Starting with LSF release 9.1.2, host-based pre- and post-execution processing will not be executed on allocated hosts to which the jobs were expanded by auto-resize.
- Host-based pre- and post-execution processing treats lease-in host the same as the local host.
- If a job with host-based pre- or post-execution processing is dispatched to Windows hosts, the job will fail, then display a pending reason.
- Since host-based pre- and post-execution processing is not defined at the job level, MultiCluster forwarded and XL jobs do not take local queue and application host-based pre- and post-execution processing information, but instead follow the remote queue and application configuration.
- The host-based pre- and post-execution processing feature is only supported by LSF 9.1.2 and future versions.

Configuration to enable pre- and post-execution processing

The pre- and post-execution processing feature is enabled by defining at least one of the parameters in the list below at the application or queue level, or by using the -E option of the **bsub** command to specify a pre-execution command. In some situations, specifying a queue-level or application-level pre-execution command can have advantages over requiring users to use **bsub -E**. For example, license checking can be set up at the queue or application level so that users do not have to enter a pre-execution command every time they submit a job.

Parameters for enabling the pre- and post-execution processing feature:

- PRE_EXEC=*command* (in lsb.queues):
 - Enables job-based pre-execution processing at the queue level.
 - The job-based pre-execution command runs on the execution host before the job starts.
 - If the PRE_EXEC command exits with a non-zero exit code, LSF re-queues the job to the front of the queue.
 - The PRE_EXEC command uses the same environment variable values as the job.
 - The PRE_EXEC command can only be used for job-based pre- and post-execution processing.
- POST_EXEC=*command* (in lsb.queues):
 - Enables job-based post-execution processing at the queue level.

- The POST_EXEC command uses the same environment variable values as the job.
- The post-execution command for the queue remains associated with the job. The original post-execution command runs even if the job is re-queued or if the post-execution command for the queue is changed after job submission.
- Before the post-execution command runs, LSB_JOBEXIT_STAT is set to the exit status of the job. The success or failure of the post-execution command has no effect on LSB_JOBEXIT_STAT.
- The post-execution command runs after the job finishes, even if the job fails.
- Specify the environment variable \$USER_POSTEXEC to allow UNIX users to define their own post-execution commands.
- The POST_EXEC command can only be used for job-based pre- and post-execution processing.
- PRE_EXEC=*command* (in lsb.applications):
 - Enables job-based pre-execution processing at the application level.
 - The pre-execution command runs on the execution host before the job starts.
 - If the PRE_EXEC command exits with a non-zero exit code, LSF re-queues the job to the front of the queue.
 - The PRE_EXEC command uses the same environment variable values as the job.
 - The PRE_EXEC command can only be used for job-based pre- and post-execution processing.
- POST_EXEC=*command* (in lsb.applications):
 - Enables job-based post-execution processing at the application level.
 - The POST_EXEC command uses the same environment variable values as the job.
 - The post-execution command for the application profile remains associated with the job. The original post-execution command runs even if the job is moved to a different application profile or is re-queued, or if the post-execution command for the original application profile is changed after job submission.
 - Before the post-execution command runs, LSB_JOBEXIT_STAT is set to the exit status of the job. The success or failure of the post-execution command has no effect on LSB_JOBEXIT_STAT.
 - The post-execution command runs after the job finishes, even if the job fails.
 - Specify the environment variable \$USER_POSTEXEC to allow UNIX users to define their own post-execution commands.
 - The POST_EXEC command can only be used for job-based pre- and post-execution processing.
- HOST_PRE_EXEC=*command* (in lsb.queues):
 - Enables host-based pre-execution processing at the queue level.
 - The pre-execution command runs on all execution hosts before the job starts.
 - If the HOST_PRE_EXEC command exits with a non-zero exit code, LSF re-queues the job to the front of the queue.
 - The HOST_PRE_EXEC command uses the same environment variable values as the job.
 - The HOST_PRE_EXEC command can only be used for host-based pre- and post-execution processing.
- HOST_POST_EXEC=*command* (in lsb.queues):
 - Enables host-based post-execution processing at the queue level.
 - The HOST_POST_EXEC command uses the same environment variable values as the job.
 - The post-execution command for the queue remains associated with the job. The original post-execution command runs even if the job is re-queued or if the post-execution command for the queue is changed after job submission.
 - Before the post-execution command runs, LSB_JOBEXIT_STAT is set to the exit status of the job. The success or failure of the post-execution command has no effect on LSB_JOBEXIT_STAT.
 - The post-execution command runs after the job finishes, even if the job fails.
 - Specify the environment variable \$USER_POSTEXEC to allow UNIX users to define their own post-execution commands.
 - The HOST_POST_EXEC command can only be used for host-based pre- and post-execution processing.
- HOST_PRE_EXEC=*command* (in lsb.applications):
 - Enables host-based pre-execution processing at the application level.
 - The pre-execution command runs on all execution hosts before the job starts.
 - If the HOST_PRE_EXEC command exits with a non-zero exit code, LSF re-queues the job to the front of the queue.
 - The HOST_PRE_EXEC command uses the same environment variable values as the job.
 - The HOST_PRE_EXEC command can only be used for host-based pre- and post-execution processing.
- HOST_POST_EXEC=*command* (in lsb.applications):
 - Enables host-based post-execution processing at the application level.
 - The HOST_POST_EXEC command uses the same environment variable values as the job.
 - The post-execution command for the application profile remains associated with the job. The original post-execution command runs even if the job is moved to a different application profile or is re-queued, or if the post-

- execution command for the original application profile is changed after job submission.
- Before the post-execution command runs, LSB_JOBEXIT_STAT is set to the exit status of the job. The success or failure of the post-execution command has no effect on LSB_JOBEXIT_STAT.
- The post-execution command runs after the job finishes, even if the job fails.
- Specify the environment variable \$USER_POSTEXEC to allow UNIX users to define their own post-execution commands.
- The HOST_POST_EXEC command can only be used for host-based pre- and post-execution processing.

Examples

The following queue specifies the job-based pre-execution command `/usr/share/lsf/pri_prexec` and the job-based post-execution command `/usr/share/lsf/pri_postexec`.

```
Begin Queue
QUEUE_NAME      = priority
PRIORITY        = 43
NICE            = 10
PRE_EXEC        = /usr/share/lsf/pri_prexec
POST_EXEC       = /usr/share/lsf/pri_postexec
End Queue
```

The following application specifies the job-based pre-execution `/usr/share/lsf/catia_prexec` and the job-based post-execution command `/usr/share/lsf/catia_postexec`.

```
Begin Application
NAME            = catia
DESCRIPTION     = CATIA V5
CPULIMIT       = 24:0/hostA      # 24 hours of host hostA
FILELIMIT      = 20000
DATALIMIT      = 20000          # jobs data segment limit
CORELIMIT      = 20000
TASKLIMIT      = 5              # job task limit
PRE_EXEC       = /usr/share/lsf/catia_prexec
POST_EXEC      = /usr/share/lsf/catia_postexec
REQUEUE_EXIT_VALUES = 55 34 78
End Application
```

The following example specifies the host-based pre-execution command `/usr/share/lsf/catia_host_prexec` and the host-based post-execution command `/usr/share/lsf/catia_host_postexec`.

```
Begin Application
NAME = catia
DESCRIPTION = CATIA host_based pre/post
HOST_PRE_EXEC = /usr/share/lsf/catia_host_prexec
HOST_POST_EXEC = /usr/share/lsf/catia_host_postexec
End Application
```

Pre- and post-execution processing behavior

Job-based pre- and post-execution processing applies to both UNIX and Windows hosts. Host-based pre- and post-execution processing only applies to UNIX host.

Host type	Environment
UNIX	<ul style="list-style-type: none"> The pre- and post-execution commands run in the <code>/tmp</code> directory under <code>/bin/sh -c</code>, which allows the use of shell features in the commands. The following example shows valid configuration lines: <pre>PRE_EXEC= /usr/share/lsf/misc/testq_pre >> /tmp/pre.out POST_EXEC= /usr/share/lsf/misc/testq_post grep -v "Testing..."</pre> LSF sets the PATH environment variable to <code>PATH='/bin /usr/bin /sbin /usr/sbin'</code> The stdin, stdout, and stderr are set to <code>/dev/null</code>

Host type	Environment
Windows	<ul style="list-style-type: none"> • The pre- and post-execution commands run under cmd.exe /c • The standard input, standard output, and standard error are set to NULL • The PATH is determined by the setup of the LSF Service

Note: If the pre-execution or post-execution command is not in your usual execution path, you must specify the full path name of the command.

Command execution order for pre- and post-execution processing

Pre-execution processing flow/stages are:

1. Host-based queue level pre-processing
2. Host-based application level pre-processing
3. Job-based queue level pre-processing
4. Job-based job level pre-processing or job-based application level pre-processing

Post-execution processing flow/stages are:

1. Job-based job level post-processing or job-based application level post-processing
2. Job-based queue level post-processing
3. Host-based application level post-processing
4. Host-based queue level post-processing

If queue level host-based pre-execution processing fails, then application level host-based pre-execution processing will not be executed. If host-based pre-execution processing fails, then any other job-based pre-execution processing will not be executed. If host-based pre-execution processing fails, or the job fails, host-based post-execution processing is still executed to perform any cleanup activities. The execution result will be reported as a post processing result to the management host and shown by **bhist**. If application level host-based post-execution processing fails, queue level host-based post-execution processing is still executed.

Command behavior for job-based pre-execution processing

A pre-execution command returns information to LSF by means of the exit status. LSF holds the job in the queue until the specified pre-execution command returns an exit code of zero (0). If the pre-execution command exits with a non-zero value, the job pends until LSF tries again to dispatch it. While the job remains in the PEND state, LSF dispatches other jobs to the execution host.

If the pre-execution command exits with a value of 99, the job exits without pending. This allows you to cancel the job if the pre-execution command fails.

You must ensure that the pre-execution command runs without side effects; that is, you should define a pre-execution command that does not interfere with the job itself. For example, if you use the pre-execution command to reserve a resource, you cannot also reserve the same resource as part of the job submission.

LSF users can specify a pre-execution command at job submission. LSF first finds a suitable host on which to run the job and then runs the pre-execution command on that host. If the pre-execution command runs successfully and returns an exit code of zero, LSF runs the job.

Command behavior for job-based post-execution processing

A post-execution command runs after the job finishes, regardless of the exit state of the job. Once a post-execution command is associated with a job, that command runs even if the job fails. You cannot configure the post-execution command to run only under certain conditions.

The resource usage of post-execution processing is not included in the job resource usage calculation, and post-execution command exit codes are not reported to LSF.

If `POST_EXEC=$USER_POSTEXEC` in either `lsb.applications` or `lsb.queues`, UNIX users can define their own post-execution commands:

```
setenv USER_POSTEXEC /path_name
```

where the path name for the post-execution command is an absolute path.

If POST_EXEC=\$USER_POSTEXEC and ...	Then ...
The user defines the USER_POSTEXEC environment variable	<ul style="list-style-type: none"> LSF runs the post-execution command defined by the environment variable USER_POSTEXEC After the user-defined command runs, LSF reports successful completion of post-execution processing If the user-defined command fails, LSF reports a failure of post-execution processing
The user does not define the USER_POSTEXEC environment variable	<ul style="list-style-type: none"> LSF reports successful post-execution processing without actually running a post-execution command

Important:

Do not allow users to specify a post-execution command when the pre- and post-execution commands are set to run under the root account.

Command execution for host-based pre- and post-execution processing

All environment variables set for job execution are passed to and set for all execution hosts before host-based pre- and post-execution processing begins.

By default, host-based pre- and post-execution processing runs under the account of the user who submits the job. To run host-based pre and post execution commands under a different user account at the queue level (such as root for privileged operations), configure the parameter LSB_PRE_POST_EXEC_USER in lsf.sudoers. Also, the /etc/lsf.sudoers file must be deployed on all nodes in order to run host-based pre- and post-execution processing.

The execution is successful only if all of the following conditions are met:

- All execution hosts received the pre/post command.
- All execution hosts executed the command with exit code 0.
- All execution hosts executed the command within the specified timeout.

The execution result is aggregated to the first execution host and then reports to the management host.

If there is any assigned CPU affinity range, queue or application level host-based pre-execution processing is limited to run within that range. Host-based post-execution processing is not constrained to run within the CPU affinity range.

The **rusage** of host-based pre-execution on the first execution host will be collected and counted as job **rusage**. On a non-first execution host, the **rusage** of the host-based pre-execution will be ignored. During host-based post-execution, there is no **rusage** collection.

If **sbatchd** quits and a job finishes before **sbatchd** restarts, then host-based post-execution processing will be executed.

The following example shows host-based pre- and post-execution processing for normal low priority jobs, running only if hosts are lightly loaded:

```
bqueues -l normal
QUEUE: normal
-- Default queue.
```

PARAMETERS/STATISTICS

```
PRIO NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SSUSP  USUSP  RSV
30   20  Open:Active        -   -   -   -   0     0     0     0     0     0
Interval for a host to accept two jobs is 0 seconds
```

SCHEDULING PARAMETERS

```
          r15s  r1m  r15m  ut      pg      io      ls      it      tmp      swp      mem
loadSched  -    -    -    -      -      -      -      -      -      -      -
loadStop   -    -    -    -      -      -      -      -      -      -      -
```

```

SCHEDULING POLICIES: NO_INTERACTIVE

USERS: all
HOSTS: all
ADMINISTRATORS: Admin1
PRE_EXEC: echo "queue-level pre-exec" >> /tmp/pre.$LSB_JOBID.$LSB_JOBINDEX
POST_EXEC: echo "queue-level post-exec" >> /tmp/post.$LSB_JOBID.$LSB_JOBINDEX

HOST_PRE_EXEC: echo "queue-level host-based pre-exec" >> /tmp/pre.$LSB_JOBID.$LSB_JOBINDEX
HOST_POST_EXEC: echo "queue-level host-based post-exec" >> /tmp/post.$LSB_JOBID.$LSB_JOBINDEX

bapp -l app
APPLICATION NAME: app

STATISTICS:
      NJOBS      PEND      RUN      SSUSP      USUSP      RSV
        0         0         0         0         0         0

PARAMETERS:
PRE_EXEC: echo "app-level pre-exec" >> /tmp/pre.$LSB_JOBID.$LSB_JOBINDEX
POST_EXEC: echo "app-level post-exec" >> /tmp/post.$LSB_JOBID.$LSB_JOBINDEX
RESIZABLE_JOBS: Auto
HOST_PRE_EXEC: echo "app-level host-based pre-exec" >> /tmp/pre.$LSB_JOBID.$LSB_JOBINDEX
HOST_POST_EXEC: echo "app-level host-based post-exec" >> /tmp/post.$LSB_JOBID.$LSB_JOBINDEX

```

- [Checking job history for a pre-execution script failure](#)

Checking job history for a pre-execution script failure

About this task

Each time your job tries to run on a host and the pre-execution script fails to run successfully, your job pends until it is dispatched again.

Procedure

Run `bhist -l job_number`.

The history of the job displays, including any pending and dispatching on hosts due to pre-execution scripts exiting with an incorrect exit code.

Configuration to modify pre- and post-execution processing

Configuration parameters modify various aspects of pre- and post-execution processing behavior by:

- Preventing a new job from starting until post-execution processing has finished
- Controlling the length of time post-execution processing can run
- Specifying a user account under which the pre- and post-execution commands run
- Controlling how many times pre-execution retries
- Determining if email providing details of the post execution output should be sent to the user who submitted the job. For more details, see [LSB_POSTEXEC SEND MAIL](#).

Some configuration parameters only apply to job-based pre- and post-execution processing and some apply to both job- and host-based pre- and post-execution processing:

Job- and host-based	Job-based only
---------------------	----------------

Job- and host-based	Job-based only
JOB_INCLUDE_POSTPROC in lsb.applications and lsb.params	PREEXEC_EXCLUDE_HOST_EXIT_VALUES in lsb.params
MAX_PREEXEC_RETRY in lsb.applications and lsb.params	
LOCAL_MAX_PREEXEC_RETRY in lsb.applications and lsb.params	
LOCAL_MAX_PREEXEC_RETRY_ACTION in lsb.applications, lsb.queues, and lsb.params	
REMOTE_MAX_PREEXEC_RETRY in lsb.applications and lsb.params	
LSB_DISABLE_RERUN_POST_EXEC in lsf.conf	
JOB_PREPROC_TIMEOUT in lsb.applications and lsb.params	
JOB_POSTPROC_TIMEOUT in lsb.applications and lsb.params	
LSB_PRE_POST_EXEC_USER in lsf.sudoers	
LSB_POSTEXEC_SEND_MAIL in lsf.conf	

For details on each parameter, see [configuration reference](#).

JOB_PREPROC_TIMEOUT is designed to protect the system from hanging during pre-execution processing. When LSF detects pre-execution processing is running longer than the JOB_PREPROC_TIMEOUT value (the default value is infinite), LSF will terminate the execution. Therefore, the LSF Administrator should ensure JOB_PREPROC_TIMEOUT is set to a value longer than any pre-execution processing is required. JOB_POSTPROC_TIMEOUT should also be set to a value that gives host-based post execution processing enough time to run.

Configuration to modify when new jobs can start

When a job finishes, sbatchd reports a job finish status of DONE or EXIT to mbatchd. This causes LSF to release resources associated with the job, allowing new jobs to start on the execution host before post-execution processing from a previous job has finished.

In some cases, you might want to prevent the overlap of a new job with post-execution processing. Preventing a new job from starting prior to completion of post-execution processing can be configured at the application level or at the job level.

At the job level, the **bsub -w** option allows you to specify job dependencies; the keywords **post_done** and **post_err** cause LSF to wait for completion of post-execution processing before starting another job.

At the application level:

File	Parameter and syntax	Description
lsb.applications lsb.params	JOB_INCLUDE_POSTPROC=Y	<ul style="list-style-type: none"> Enables completion of post-execution processing before LSF reports a job finish status of DONE or EXIT Prevents a new job from starting on a host until post-execution processing is finished on that host

- sbatchd sends both job finish status (DONE or EXIT) and post-execution processing status (POST_DONE or POST_ERR) to mbatchd at the same time
- The job remains in the RUN state and holds its job slot until post-execution processing has finished
- Job requeue happens (if required) after completion of post-execution processing, not when the job itself finishes
- For job history and job accounting, the job CPU and run times include the post-execution processing CPU and run times
- The job control commands **bstop**, **bkill**, and **bresume** have no effect during post-execution processing
- If a host becomes unavailable during post-execution processing for a rerunnable job, mbatchd sees the job as still in the RUN state and reruns the job
- LSF does not preempt jobs during post-execution processing

Configuration to modify the post-execution processing time

Controlling the length of time post-execution processing can run is configured at the application level.

File	Parameter and syntax	Description
lsb.applications lsb.params	JOB_POSTPROC_TIMEOUT= <i>minutes</i>	<ul style="list-style-type: none">Specifies the length of time, in minutes, that post-execution processing can run.The specified value must be greater than zero.If post-execution processing takes longer than the specified value, sbatchd reports post-execution failure—a status of POST_ERR. On UNIX and Linux, it kills the entire process group of the job's pre-execution processes. On Windows, only the parent process of the pre-execution command is killed when the timeout expires, the child processes of the pre-execution command are not killed.If JOB_INCLUDE_POSTPROC=Y and sbatchd kills the post-execution process group, post-execution processing CPU time is set to zero, and the job's CPU time does not include post-execution CPU time.

Configuration to modify the pre- and post-execution processing user account

Specifying a user account under which the pre- and post-execution commands run is configured at the system level. By default, both the pre- and post-execution commands run under the account of the user who submits the job.

File	Parameter and syntax	Description
lsf.sudoers	LSB_PRE_POST_EXEC_USER= <i>user_name</i>	<ul style="list-style-type: none">Specifies the user account under which pre- and post-execution commands run (UNIX only)This parameter applies only to pre- and post-execution commands configured at the queue level; pre-execution commands defined at the application or job level run under the account of the user who submits the jobIf the pre-execution or post-execution commands perform privileged operations that require root permissions on UNIX hosts, specify a value of rootYou must edit the lsf.sudoers file on all UNIX hosts within the cluster and specify the same user account

Configuration to control how many times pre-execution retries

By default, if job pre-execution fails, LSF retries the job automatically. The job remains in the queue and pre-execution is retried 5 times by default, to minimize any impact to performance and throughput.

Limiting the number of times LSF retries job pre-execution is configured cluster-wide (lsb.params), at the queue level (lsb.queue), and at the application level (lsb.applications). Pre-execution retry in lsb.applications overrides lsb.queue, and lsb.queue overrides lsb.params configuration.

Configuration file	Parameter and syntax	Behavior
--------------------	----------------------	----------

Configuration file	Parameter and syntax	Behavior
lsb.params	LOCAL_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the local cluster. Specify an integer greater than 0 <p>By default, the number of retries is unlimited.</p>
	MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>
	REMOTE_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. <p>Equivalent to MAX_PREEEXEC_RETRY</p> <ul style="list-style-type: none"> Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>
lsb.queues	LOCAL_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the local cluster. Specify an integer greater than 0 <p>By default, the number of retries is unlimited.</p>
	MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>
	REMOTE_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. <p>Equivalent to MAX_PREEEXEC_RETRY</p> <ul style="list-style-type: none"> Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>
lsb.applications	LOCAL_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the local cluster. Specify an integer greater than 0 <p>By default, the number of retries is unlimited.</p>
	MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>

Configuration file	Parameter and syntax	Behavior
	REMOTE_MAX_PREEEXEC_RETRY= <i>integer</i>	<ul style="list-style-type: none"> Controls the maximum number of times to attempt the pre-execution command of a job on the remote cluster. <p>Equivalent to MAX_PREEEXEC_RETRY</p> <ul style="list-style-type: none"> Specify an integer greater than 0 <p>By default, the number of retries is 5.</p>

When pre-execution retry is configured, if a job pre-execution fails and exits with non-zero value, the number of pre-exec retries is set to 1. When the pre-exec retry limit is reached, the job is suspended with PSUSP status.

The number of times that pre-execution is retried includes queue-level, application-level, and job-level pre-execution command specifications. When pre-execution retry is configured, a job will be suspended when the sum of its queue-level pre-exec retry times + application-level pre-exec retry times is greater than the value of the pre-execution retry parameter or if the sum of its queue-level pre-exec retry times + job-level pre-exec retry times is greater than the value of the pre-execution retry parameter.

The pre-execution retry limit is recovered when LSF is restarted and reconfigured. LSF replays the pre-execution retry limit in the PRE_EXEC_START or JOB_STATUS events in lsb.events.

Configuration to define default behavior of a job after it reaches the pre-execution retry limit

By default, if LSF retries the pre-execution command of a job on the local cluster and reaches the pre-execution retry threshold (LOCAL_MAX_PREEEXEC_RETRY in lsb.params, lsb.queues, or lsb.applications), LSF suspends the job.

This default behavior of a job that has reached the pre-execution retry limit is configured cluster-wide (lsb.params), at the queue level (lsb.queues), and at the application level (lsb.applications). The behavior specified in lsb.applications overrides lsb.queues, and lsb.queues overrides the lsb.params configuration.

Configuration file	Parameter and syntax	Behavior
lsb.params	LOCAL_MAX_PREEEXEC_RETRY_ACTION = SUSPEND EXIT	<ul style="list-style-type: none"> Specifies the default behavior of a job (on the local cluster) that has reached the maximum pre-execution retry limit. If set to SUSPEND, the job is suspended and its status is set to PSUSP. <p>If set to EXIT, the job status is set to EXIT and the exit code is the same as the last pre-execution fail exit code.</p> <p>By default, the job is suspended.</p>
lsb.queues	LOCAL_MAX_PREEEXEC_RETRY_ACTION = SUSPEND EXIT	<ul style="list-style-type: none"> Specifies the default behavior of a job (on the local cluster) that has reached the maximum pre-execution retry limit. If set to SUSPEND, the job is suspended and its status is set to PSUSP. <p>If set to EXIT, the job status is set to EXIT and the exit code is the same as the last pre-execution fail exit code.</p> <p>By default, this is not defined.</p>

Configuration file	Parameter and syntax	Behavior
lsb.applications	LOCAL_MAX_PREEEXEC_RETRY_ACTION = SUSPEND EXIT	<ul style="list-style-type: none"> Specifies the default behavior of a job (on the local cluster) that has reached the maximum pre-execution retry limit. If set to SUSPEND, the job is suspended and its status is set to PSUSP. <p>If set to EXIT, the job status is set to EXIT and the exit code is the same as the last pre-execution fail exit code.</p> <p>By default, this is not defined.</p>

- [Set host exclusion based on job-based pre-execution scripts](#)

Set host exclusion based on job-based pre-execution scripts

Before you begin

You must know the exit values your pre-execution script exits with that indicate failure.

About this task

Any non-zero exit code in a pre-execution script indicates a failure. For those jobs that are designated as re-runnable on failure, LSF filters on the pre-execution script failure to determine whether the job that failed in the pre-execution script should exclude the host where the pre-execution script failed. That host is no longer a candidate to run the job.

Procedure

1. Create a pre-execution script that exits with a specific value if it is unsuccessful.

Example:

```
#!/bin/sh

# Usually, when pre_exec failed due to host reason like
# /tmp is full, we should exit directly to let LSF
# re-dispatch the job to a different host.
# For example:
#   define PREEEXEC_RETRY_EXIT_VALUES = 10 in lsb.params
#   exit 10 when pre_exec detect that /tmp is full.
# LSF will re-dispatch this job to a different host under
# such condition.
DISC=/tmp
PARTITION=`df -Ph | grep -w $DISC | awk '{print $6}'`
FREE=`df -Ph | grep -w $DISC | awk '{print $5}' | awk -F% '{print $1}'`

echo "$FREE"
if [ "${FREE}" != "" ]
then
    if [ "${FREE}" -le "2" ] # When there's only 2% available space for
                          # /tmp on this host, we can let LSF
                          # re-dispatch the job to a different host
    then
        exit 10
    fi
fi

# Sometimes, when pre_exec failed due to nfs server being busy,
# it can succeed if we retry it several times in this script to
# affect LSF performance less.
RETRY=10
```

```

while [ $RETRY -gt 0 ]
do
    #mount host_name:/export/home/bill /home/bill
    EXIT=`echo $?`
    if [ $EXIT -eq 0 ]
    then
        RETRY=0
    else
        RETRY=`expr $RETRY - 1`
        if [ $RETRY -eq 0 ]
        then
            exit 99 # We have tried for 9 times.
                    # Something is wrong with nfs server, we need
                    # to fail the job and fix the nfs problem first.
                    # We need to submit the job again after nfs problem
                    # is resolved.
        fi
    fi
done

```

2. In `lsb.params`, use `PREEXEC_EXCLUDE_HOST_EXIT_VALUES` to set the exit values that indicate the pre-execution script failed to run.
Values from 1-255 are allowed, excepting 99 (reserved value). Separate values with a space.

For the example script above, set `PREEXEC_EXCLUDE_HOST_EXIT_VALUES=10`.

3. (Optional) Define `MAX_PREEXEC_RETRY` to limit the total number of times LSF retries the pre-execution script on hosts.
4. Run `badadmin reconfig`.

Results

If a pre-execution script exits with value 10 (according to the example above), LSF adds this host to an exclusion list and attempts to reschedule the job on another host.

Hosts remain in a job's exclusion list for a period of time specified in the `LSB_EXCLUDE_HOST_PERIOD` parameter in `lsf.conf`, or until **mbatchd** restarts.

In the multicluster job lease model, `LSB_EXCLUDE_HOST_PERIOD` does not apply, so jobs remain in a job's exclusion list until **mbatchd** restarts.

What to do next

To view a list of hosts on a job's host exclusion list, run **bjobs -lp**.

Pre- and post-execution processing commands

Commands for submission

The **bsub** -E option specifies a pre-execution command, and the **bsub** -Ep option specifies a post-execution command.

The **bsub** -w option allows you to specify job dependencies that cause LSF to wait for completion of post-execution processing before starting another job.

Command	Description
bsub -E <i>command</i>	<ul style="list-style-type: none"> Defines the pre-execution command at the job level.
bsub -Ep <i>command</i>	<ul style="list-style-type: none"> Defines the post-execution command at the job level.

Command	Description
bsub -w 'post_done(job_id "job_name")'	<ul style="list-style-type: none"> Specifies the job dependency condition required to prevent a new job from starting until post-execution processing has finished without errors.
bsub -w 'post_err(job_id "job_name")'	<ul style="list-style-type: none"> Specifies the job dependency condition required to prevent a new job from starting until post-execution processing has exited with errors.

Commands to monitor

Command	Description
bhist -l bhist	<ul style="list-style-type: none"> Displays the POST_DONE and POST_ERR states which can be referenced by a job submitted with bsub -w. The resource usage of post-processing is not included in the job resource usage. The CPU and run times shown do not include resource usage for post-execution processing unless the parameter JOB_INCLUDE_POSTPROC is defined in lsb.applications or lsb.params. Displays the job exit code and reason if the pre-exec retry limit is exceeded.
bjobs -l	<ul style="list-style-type: none"> Displays information about pending, running, and suspended jobs. During post-execution processing, the job status will be RUN if the parameter JOB_INCLUDE_POSTPROC is defined in lsb.applications or lsb.params. The resource usage shown does not include resource usage for post-execution processing. Displays the job exit code and reason if the pre-exec retry limit is exceeded.
bacct	<ul style="list-style-type: none"> Displays accounting statistics for finished jobs. The CPU and run times shown do not include resource usage for post-execution processing, unless the parameter JOB_INCLUDE_POSTPROC is defined in lsb.applications or lsb.params.

Commands to control

Command	Description
bmod -E command	<ul style="list-style-type: none"> Changes the pre-execution command at the job level.
bmod -Ep command	<ul style="list-style-type: none"> Changes the post-execution command at the job level.
bmod -w 'post_done(job_id "job_name")'	<ul style="list-style-type: none"> Specifies the job dependency condition required to prevent a new job from starting until post-execution processing has finished without errors.
bmod -w 'post_err(job_id "job_name")'	<ul style="list-style-type: none"> Specifies the job dependency condition required to prevent a new job from starting until post-execution processing has exited with errors.

Commands to display configuration

Command	Description
bapp -l	<ul style="list-style-type: none"> Displays information about application profiles configured in lsb.applications, including the values defined for PRE_EXEC, POST_EXEC, HOST_PRE_EXEC, HOST_POST_EXEC, JOB_INCLUDE_POSTPROC, JOB_POSTPROC_TIMEOUT, LOCAL_MAX_PREEEXEC_RETRY, MAX_PREEEXEC_RETRY, and REMOTE_MAX_PREEEXEC_RETRY.

Command	Description
bparams	<ul style="list-style-type: none"> Displays the value of parameters defined in <code>lsb.params</code>, including the values defined for <code>LOCAL_MAX_PREEEXEC_RETRY</code>, <code>MAX_PREEEXEC_RETRY</code>, and <code>REMOTE_MAX_PREEEXEC_RETRY</code>.
bqueues -l	<ul style="list-style-type: none"> Displays information about queues configured in <code>lsb.queues</code>, including the values defined for <code>PRE_EXEC</code>, <code>POST_EXEC</code>, <code>HOST_PRE_EXEC</code>, <code>HOST_POST_EXEC</code>, <code>LOCAL_MAX_PREEEXEC_RETRY</code>, <code>MAX_PREEEXEC_RETRY</code>, and <code>REMOTE_MAX_PREEEXEC_RETRY</code>.

Use a text editor to view the `lsf.sudoers` configuration file.

Job starters

- [About job starters](#)

A *job starter* is a specified shell script or executable program that sets up the environment for a job and then runs the job. The job starter and the job share the same environment. This chapter discusses two ways of running job starters in LSF and how to set up and use them.

- [Command-level job starters](#)

A command-level job starter allows you to specify an executable file that does any necessary setup for the job and runs the job when the setup is complete. You can select an existing command to be a job starter, or you can create a script containing a desired set of commands to serve as a job starter.

- [Queue-level job starters](#)

LSF administrators can define a job starter for an individual queue to create a specific environment for jobs to run in. A queue-level job starter specifies an executable that performs any necessary setup, and then runs the job when the setup is complete. The `JOB_STARTER` parameter in `lsb.queues` specifies the command or script that is the job starter for the queue.

- [Control the execution environment with job starters](#)

In some cases, using **bsub -L** does not result in correct environment settings on the execution host.

About job starters

A *job starter* is a specified shell script or executable program that sets up the environment for a job and then runs the job. The job starter and the job share the same environment. This chapter discusses two ways of running job starters in LSF and how to set up and use them.

Some jobs have to run in a particular environment, or require some type of setup to be performed before they run. In a shell environment, job setup is often written into a wrapper shell script file that itself contains a call to start the desired job.

A job starter is a specified wrapper script or executable program that typically performs environment setup for the job, then calls the job itself, which inherits the execution environment created by the job starter. LSF controls the job starter process, rather than the job. One typical use of a job starter is to customize LSF for use with specific application environments, such as Alias Renderer or IBM Rational ClearCase.

Two ways to run job starters

You run job starters two ways in LSF. You can accomplish similar things with either job starter, but their functional details are slightly different.

Command-level

Are user-defined. They run interactive jobs submitted using **lsgrun**, or **ch**. Command-level job starters have no effect on batch jobs, including interactive batch jobs run with **bsub -I**.

Use the `LSF_JOB_STARTER` environment variable to specify a job starter for interactive jobs.

Queue-level

Defined by the LSF administrator, and run batch jobs submitted to a queue defined with the JOB_STARTER parameter set. Use **bsub** to submit jobs to queues with job-level job starters.

A queue-level job starter is configured in the queue definition in `lsb.queues`.

Pre-execution commands are not job starters

A job starter differs from a pre-execution command. A pre-execution command must run successfully and exit before the LSF job starts. It can signal LSF to dispatch the job, but because the pre-execution command is an unrelated process, it does not control the job or affect the execution environment of the job. A job starter, however, is the process that LSF controls. It is responsible for invoking LSF and controls the execution environment of the job.

Examples

The following are some examples of job starters:

- In UNIX, a job starter defined as **/bin/ksh -c** causes jobs to be run under a Korn shell environment.
- In Windows, a job starter defined as **C:\cmd.exe /C** causes jobs to be run under a DOS shell environment.
Note:
For job starters that execute on a Windows Server 2003, x64 Edition platform, users must have “Read” and “Execute” privileges for `cmd.exe`.
- Setting the JOB_STARTER parameter in `lsb.queues` to `$USER_STARTER` enables users to define their own job starters by defining the environment variable `USER_STARTER`.
Restriction: `USER_STARTER` can only be used in UNIX clusters. Mixed or Windows-only clusters are not supported.
- Setting a job starter to **make clean** causes the command **make clean** to be run before the user job.

Command-level job starters

A command-level job starter allows you to specify an executable file that does any necessary setup for the job and runs the job when the setup is complete. You can select an existing command to be a job starter, or you can create a script containing a desired set of commands to serve as a job starter.

This section describes how to set up and use a command-level job starter to run interactive jobs.

Command-level job starters have no effect on batch jobs, including interactive batch jobs.

A job starter can also be defined at the queue level using the JOB_STARTER parameter. Only the LSF administrator can configure queue-level job starters.

LSF_JOB_STARTER environment variable

Use the LSF_JOB_STARTER environment variable to specify a command or script that is the job starter for the interactive job. When the environment variable LSF_JOB_STARTER is defined, RES invokes the job starter rather than running the job itself, and passes the job to the job starter as a command-line argument.

Using command-level job starters

- UNIX: The job starter is invoked from within a Bourne shell, making the command-line equivalent:

```
/bin/sh -c "$LSF_JOB_STARTER command [argument ...]"
```

where *command* and *argument* are the command-line arguments you specify in **lsrun**, **lsgrun**, or **ch**.

- Windows: RES runs the job starter, passing it your commands as arguments:

```
LSF_JOB_STARTER command [argument ...]
```

Examples

UNIX

If you define the LSF_JOB_STARTER environment variable using the following C-shell command:

```
% setenv LSF_JOB_STARTER "/bin/sh -c"
```

Then you run a simple C-shell job:

```
% lsrunc "'a.out; hostname'"
```

The command that actually runs is

```
/bin/sh -c "/bin/sh -c 'a.out; hostname'"
```

The job starter can be a shell script. In the following example, the LSF_JOB_STARTER environment variable is set to the Bourne shell script named job_starter:

```
$ LSF_JOB_STARTER=/usr/local/job_starter
```

The job_starter script contains the following:

```
#!/bin/sh
set term = xterm eval "$@"
```

Windows

If you define the LSF_JOB_STARTER environment variable as follows:

```
set LSF_JOB_STARTER=C:\cmd.exe /C
```

Then you run a simple DOS shell job:

```
C:\> lsrunc dir /p
```

The command that actually runs is:

```
C:\cmd.exe /C dir /p
```

Queue-level job starters

LSF administrators can define a job starter for an individual queue to create a specific environment for jobs to run in. A queue-level job starter specifies an executable that performs any necessary setup, and then runs the job when the setup is complete. The JOB_STARTER parameter in lsb.queues specifies the command or script that is the job starter for the queue.

This section describes how to set up and use a queue-level job starter.

Queue-level job starters have no effect on interactive jobs, unless the interactive job is submitted to a queue as an interactive batch job.

LSF users can also select an existing command or script to be a job starter for their interactive jobs using the LSF_JOB_STARTER environment variable.

- [Configuring a queue-level job starter](#)
- [JOB_STARTER parameter \(lsb.queues\)](#)

The JOB_STARTER parameter in the queue definition (lsb.queues) has the following format:

Configuring a queue-level job starter

Procedure

Use the `JOB_STARTER` parameter in `lsb.queues` to specify a queue-level job starter in the queue definition. All jobs submitted to this queue are run using the job starter. The jobs are called by the specified job starter process rather than initiated by the batch daemon process.

For example:

```
Begin Queue
.
JOB_STARTER = xterm -e
.
End Queue
```

All jobs submitted to this queue are run under an **xterm** terminal emulator.

JOB_STARTER parameter (lsb.queues)

The `JOB_STARTER` parameter in the queue definition (`lsb.queues`) has the following format:

```
JOB_STARTER=starter [starter] ["%USRCMD"] [starter]
```

The string *starter* is the command or script that is used to start the job. It can be any executable that can accept a job as an input argument. Optionally, additional strings can be specified.

When starting a job, LSF runs the `JOB_STARTER` command, and passes the shell script containing the job commands as the argument to the job starter. The job starter is expected to do some processing and then run the shell script containing the job commands. The command is run under **/bin/sh -c** and can contain any valid Bourne shell syntax.

%USRCMD string

The special string `%USRCMD` indicates the position of the job starter command in the job command line. By default, the user commands run after the job starter, so the `%USRCMD` string is not usually required. For example, these two job starters both give the same results:

```
JOB_STARTER = /bin/csh -c
JOB_STARTER = /bin/csh -c "%USRCMD"
```

You must enclose the `%USRCMD` string in quotes. The `%USRCMD` string can be followed by additional commands. For example:

```
JOB_STARTER = /bin/csh -c "%USRCMD;sleep 10"
```

If a user submits the following job to the queue with this job starter:

```
bsub myjob arguments
```

the command that actually runs is:

```
/bin/csh -c "myjob arguments; sleep 10"
```

Control the execution environment with job starters

In some cases, using **bsub -L** does not result in correct environment settings on the execution host.

LSF provides the following two job starters:

- **preservestarter** - preserves the default environment of the execution host. It does not include any submission host settings.
- **augmentstarter** - augments the default user environment of the execution host by adding settings from the submission host that are not already defined on the execution host

bsub -L cannot be used for a Windows execution host.

Where the job starter executables are located

By default, the job starter executables are installed in LSF_BINDIR. If you prefer to store them elsewhere, make sure they are in a directory that is included in the default PATH on the execution host.

For example:

- On Windows, put the job starter under %WINDIR%.
- On UNIX, put the job starter under \$HOME/bin.

Source code for the job starters

The source code for the job starters is installed in LSF_MISC/examples.

Add to the initial login environment

By default, the **preservestarter** job starter preserves the environment that RES establishes on the execution host, and establishes an initial login environment for the user with the following variables from the user's login environment on the execution host:

- HOME
- USER
- SHELL
- LOGNAME

Any additional environment variables that exist in the user's login environment on the submission host must be added to the job starter source code.

Example

A user's .login script on the submission host contains the following setting:

```
if ($TERM != "xterm") then
    set TERM=`tset - -Q -m 'switch:?vt100' ....
else
    stty -tabs
endif
```

The TERM environment variable must also be included in the environment on the execution host for login to succeed. If it is missing in the job starter, the login fails, the job starter may fail as well. If the job starter can continue with only the initial environment settings, the job may execute correctly, but this is not likely.

Job control actions

Learn how to configure job control actions to override or augment the default job control actions.

After a job is started, it can be killed, suspended, or resumed by the system, an LSF user, or LSF administrator. LSF job control actions cause the status of a job to change.

Default job control actions

After a job is started, it can be killed, suspended, or resumed by the system, an LSF user, or LSF administrator. LSF job control actions cause the status of a job to change. LSF supports the following default actions for job controls:

- SUSPEND
- RESUME
- TERMINATE

On successful completion of the job control action, the LSF job control commands cause the status of a job to change.

The environment variable `LS_EXEC_T` is set to the value `JOB_CONTROLS` for a job when a job control action is initiated.

SUSPEND action

Change a running job from RUN state to one of the following states:

- USUSP or PSUSP in response to **bstop**
- SSUSP state when the LSF system suspends the job

The default action is to send the following signals to the job:

- SIGTSTP for parallel or interactive jobs. SIGTSTP is caught by the parent process and passed to all the child processes running on other hosts.
- SIGSTOP for sequential jobs. SIGSTOP cannot be caught by user programs. The SIGSTOP signal can be configured with the `LSB_SIGSTOP` parameter in `lsf.conf`.

LSF invokes the SUSPEND action when:

- The user or LSF administrator issues a **bstop** or **bkill** command to the job
- Load conditions on the execution host satisfy *any* of:
 - The suspend conditions of the queue, as specified by the `STOP_COND` parameter in `lsb.queues`
 - The scheduling thresholds of the queue or the execution host
- The run window of the queue closes
- The job is preempted by a higher priority job

RESUME action

Change a suspended job from SSUSP, USUSP, or PSUSP state to the RUN state. The default action is to send the signal SIGCONT.

LSF invokes the RESUME action when:

- The user or LSF administrator issues a **brresume** command to the job
- Load conditions on the execution host satisfy *all* of:
 - The resume conditions of the queue, as specified by the `RESUME_COND` parameter in `lsb.queues`
 - The scheduling thresholds of the queue and the execution host
- A closed run window of the queue opens again
- A preempted job finishes

TERMINATE action

Terminate a job. This usually causes the job change to EXIT status. The default action is to send SIGINT first, then send SIGTERM 10 seconds after SIGINT, then send SIGKILL 10 seconds after SIGTERM. The delay between signals allows user programs to catch the signals and clean up before the job terminates.

To override the 10 second interval, use the parameter `JOB_TERMINATE_INTERVAL` in the `lsb.params` file.

LSF invokes the TERMINATE action when:

- The user or LSF administrator issues a **bkill** or **brequeue** command to the job
- The `TERMINATE_WHEN` parameter in the queue definition (`lsb.queues`) causes a SUSPEND action to be redirected to TERMINATE
- The job reaches its CPULIMIT, MEMLIMIT, RUNLIMIT or PROCESSLIMIT
- The administrator defines a cluster wide termination grace period for killing orphan jobs, or the user issues a **bsub -w -ti** command sub-option to enforce immediate automatic orphan job termination on a per-job basis.

If the execution of an action is in progress, no further actions are initiated unless it is the TERMINATE action. A TERMINATE action is issued for all job states except PEND.

Windows job control actions

On Windows, actions equivalent to the UNIX signals have been implemented to do the default job control actions. Job control messages replace the SIGINT and SIGTERM signals, but only customized applications will be able to process them.

Termination is implemented by the `TerminateProcess()` system call.

Configure job control actions

Several situations may require overriding or augmenting the default actions for job control. For example:

- Notifying users when their jobs are suspended, resumed, or terminated
- An application holds resources that are not freed by suspending the job. The administrator can set up an action to be performed that causes the resource to be released before the job is suspended and re-acquired when the job is resumed.
- The administrator wants the job check-pointed before being:
 - Suspended when a run window closes
 - Killed when the RUNLIMIT is reached
- A distributed parallel application must receive a catch-able signal when the job is suspended, resumed or terminated to propagate the signal to remote processes.

To override the default actions for the SUSPEND, RESUME, and TERMINATE job controls, specify the `JOB_CONTROLS` parameter in the queue definition in `lsb.queues`.

JOB_CONTROLS parameter (lsb.queues)

The `JOB_CONTROLS` parameter has the following format:

```
Begin Queue
...
JOB_CONTROLS = SUSPEND[signal | CHPNT | command] \
                RESUME[signal | command] \
                TERMINATE[signal | CHPNT | command]
...
End Queue
```

When LSF needs to suspend, resume, or terminate a job, it invokes one of the following actions as specified by `SUSPEND`, `RESUME`, and `TERMINATE`.

signal

A UNIX signal name (for example, `SIGTSTP` or `SIGTERM`). The specified signal is sent to the job.

The same set of signals is not supported on all UNIX systems. To display a list of the symbolic names of the signals (without the `SIG` prefix) supported on your system, use the **kill -l** command.

CHKPNT

Checkpoint the job. Only valid for `SUSPEND` and `TERMINATE` actions.

- If the `SUSPEND` action is `CHKPNT`, the job is check-pointed and then stopped by sending the `SIGSTOP` signal to the job automatically.
- If the `TERMINATE` action is `CHKPNT`, then the job is check-pointed and killed automatically.

Command

A `/bin/sh` command line.

- Do not quote the command line inside an action definition.
- Do not specify a signal followed by an action that triggers the same signal (for example, do not specify `JOB_CONTROLS=TERMINATE[bkill]` or `JOB_CONTROLS=TERMINATE[brequeue]`). This will cause a deadlock between the signal and the action.

Use a command as a job control action

- The command line for the action is run with `/bin/sh -c` so you can use shell features in the command.
- The command is run as the user of the job.
- All environment variables set for the job are also set for the command action. The following additional environment variables are set:

- LSB_JOBPGIDS: A list of current process group IDs of the job
 - LSB_JOBPIIDS: A list of current process IDs of the job
- For the SUSPEND action command, the environment variables LSB_SUSP_REASONS and LSB_SUSP_SUBREASONS are also set. Use them together in your custom job control to determine the exact load threshold that caused a job to be suspended.
 - LSB_SUSP_REASONS: An integer representing a bitmap of suspending reasons as defined in lsb.h. The suspending reason can allow the command to take different actions based on the reason for suspending the job.
 - LSB_SUSP_SUBREASONS: An integer representing the load index that caused the job to be suspended. When the suspending reason SUSP_LOAD_REASON (suspended by load) is set in LSB_SUSP_REASONS, LSB_SUSP_SUBREASONS is set to one of the load index values defined in lsf.h.
- The standard input, output, and error of the command are redirected to the NULL device, so you cannot tell directly whether the command runs correctly. The default null device on UNIX is /dev/null.
- You should make sure the command line is correct. If you want to see the output from the command line for testing purposes, redirect the output to a file inside the command line.

TERMINATE job actions

Use caution when configuring TERMINATE job actions that do more than just kill a job. For example, resource usage limits that terminate jobs change the job state to SSUSP while LSF waits for the job to end. If the job is not killed by the TERMINATE action, it remains suspended indefinitely.

TERMINATE_WHEN parameter (lsb.queues)

In certain situations you may want to terminate the job instead of calling the default SUSPEND action. For example, you may want to kill jobs if the run window of the queue is closed. Use the TERMINATE_WHEN parameter to configure the queue to invoke the TERMINATE action instead of SUSPEND.

Syntax

```
TERMINATE_WHEN = [LOAD] [PREEMPT] [WINDOW]
```

Example

The following defines a night queue that will kill jobs if the run window closes.

```
Begin Queue
NAME           = night
RUN_WINDOW     = 20:00-08:00
TERMINATE_WHEN = WINDOW
JOB_CONTROLS   = TERMINATE[ kill -KILL $LSB_JOBPIIDS; \
    echo "job $LSB_JOBID killed by queue run window" | \
    mail $USER ]
End Queue
```

LSB_SIGSTOP parameter (lsf.conf)

Use LSB_SIGSTOP to configure the SIGSTOP signal sent by the default SUSPEND action.

If LSB_SIGSTOP is set to anything other than SIGSTOP, the SIGTSTP signal that is normally sent by the SUSPEND action is not sent. For example, if LSB_SIGSTOP=SIGKILL, the three default signals sent by the TERMINATE action (SIGINT, SIGTERM, and SIGKILL) are sent 10 seconds apart.

Avoid signal and action deadlock

Do not configure a job control to contain the signal or command that is the same as the action associated with that job control. This will cause a deadlock between the signal and the action.

For example, the **kill** command uses the TERMINATE action, so a deadlock results when the TERMINATE action itself contains the **kill** command.

Any of the following job control specifications will cause a deadlock:

- JOB_CONTROLS=TERMINATE[kill]
- JOB_CONTROLS=TERMINATE[brequeue]
- JOB_CONTROLS=RESUME[bresume]

- `JOB_CONTROLS=SUSPEND[bstop]`

Customize cross-platform signal conversion

LSF supports signal conversion between UNIX and Windows for remote interactive execution through RES.

On Windows, the CTRL+C and CTRL+BREAK key combinations are treated as signals for console applications (these signals are also called console control actions).

LSF supports these two Windows console signals for remote interactive execution. LSF regenerates these signals for user tasks on the execution host.

Default signal conversion

In a mixed Windows and UNIX environment, LSF has the following default conversion between the Windows console signals and the UNIX signals:

Windows	UNIX
CTRL+C	SIGINT
CTRL+BREAK	SIGQUIT

For example, if you issue the **lsrun** or **bsub -I** commands from a Windows console but the task is running on an UNIX host, pressing the CTRL+C keys will generate a UNIX `SIGINT` signal to your task on the UNIX host. The opposite is also true.

Custom signal conversion

For **lsrun** (but not **bsub -I**), LSF allows you to define your own signal conversion using the following environment variables:

- `LSF_NT2UNIX_CTRLC`
- `LSF_NT2UNIX_CTRLB`

For example:

- `LSF_NT2UNIX_CTRLC=SIGXXXX`
- `LSF_NT2UNIX_CTRLB=SIGYYYY`

Here, `SIGXXXX/SIGYYYY` are UNIX signal names such as `SIGQUIT`, `SIGTINT`, etc. The conversions will then be: `CTRL+C=SIGXXXX` and `CTRL+BREAK=SIGYYYY`.

If both `LSF_NT2UNIX_CTRLC` and `LSF_NT2UNIX_CTRLB` are set to the same value (`LSF_NT2UNIX_CTRLC=SIGXXXX` and `LSF_NT2UNIX_CTRLB=SIGXXXX`), `CTRL+C` will be generated on the Windows execution host.

For **bsub -I**, there is no conversion other than the default conversion.

Process tracking through cgroups

This feature depends on the Control Groups (cgroups) functions provided by the LINUX kernel. The cgroups functions are supported on x86_64 and PowerPC LINUX with kernel version 2.6.24 or later.

Process tracking through cgroups can capture job processes that are not in the existing job's process tree and have process group IDs that are different from the existing ones, or job processes that run very quickly, before LSF has a chance to find them in the regular or on-demand process table scan issued by PIM.

Note: LSF only detects the cgroup service status while LSF is starting or restarting. After LSF starts or restarts successfully, it will no longer check the cgroup service status. In addition, you cannot perform actions on the cgroup service (such as starting or stopping the service) when LSF is running, otherwise the job status is not correct.

To work around this issue and be able to perform actions on the cgroup service after LSF is running, run the **badadmin hclose** command to close the host, perform the actions on the cgroup service, then run the **badadmin hopen** command to open the host.

Process tracking is controlled by two parameters in `lsf.conf`:

- **LSF_PROCESS_TRACKING**: Tracks job processes and executes job control functions such as termination, suspension, resume and other signaling, on Linux systems which support cgroup's freezer subsystem.
- **LSF_LINUX_CGROUP_ACCT**: Tracks processes based on CPU and memory accounting for Linux systems that support cgroup's memory and cpuacct subsystems.

Different LSF hosts in the cluster can use different versions of **cgroup** as long as each individual LSF host is only running one version of **cgroup**. If you have both versions of cgroup enabled in a host, you must disable one of the versions. For example, hostA can use cgroup **v1** and hostB can use cgroup **v2** as long as each host is only running one version of cgroup.

If you plan to use the process tracking and cgroup accounting, you must set up freezer, cpuacct and memory subsystems on each machine in the cluster which support cgroups.

For example, to configure the cgroup's subsystems to support both LSF cgroup features:

- For Linux kernel versions earlier than 3.0 (for example, Red Hat 6.2, 6.3 and 6.4, and SUSE 11 Patch 1), add the following lines to `/etc/fstab`:

CAUTION:

Confirm that the appropriate functionality is correctly installed on the system before making updates to `/etc/fstab`.

```
cgroup /cgroup/freezer cgroup freezer,ns 0 0
cgroup /cgroup/cpuset cgroup cpuset 0 0
cgroup /cgroup/cpuacct cgroup cpuacct 0 0
cgroup /cgroup/memory cgroup memory 0 0
```

- For Linux kernel versions above 3.0 (for example, SUSE 11 Patch 2), add the following lines to `/etc/fstab`:

```
cgroup /cgroup/freezer cgroup freezer 0 0
cgroup /cgroup/cpuset cgroup cpuset 0 0
cgroup /cgroup/cpuacct cgroup cpuacct 0 0
cgroup /cgroup/memory cgroup memory 0 0
```

Then, run the following command: **mount -a -t cgroup**

Make sure these directories (`/cgroup/freezer`, `/cgroup/cpuset`, `/cgroup/cpuacct`, `/cgroup/memory`) exist in the `/cgroup` directory before the **mount** command is issued.

If you only want to enable one LSF cgroup feature (for example, **LSF_LINUX_CGROUP_ACCT**), add the following lines to `/etc/fstab`:

```
cgroup /cgroup/cpuacct cgroup cpuacct 0 0
cgroup /cgroup/memory cgroup memory 0 0
```

Or, if you use **cgconfig** to manage cgroups, you can instead configure the cgroup's subsystems to support both LSF cgroup features by adding the following to `/etc/cgconfig.conf`:

```
mount {
  freezer = /cgroup/freezer;
  cpuset = /cgroup/cpuset;
  cpuacct = /cgroup/cpuacct;
  memory = /cgroup/memory;
}
```

To start or restart the **cgconfig** service, use `/etc/init.d/cgconfig start|restart`. Normally, the **cgconfig** is not installed by default. To install it, use the corresponding rpm package **libcgroup** for Red Hat and **libcgroup1** for SUSE.

For one successful cgroup mount operation, you can use the file `/proc/mounts` to check, it should contains the lines similar as:

```
cgroup /cgroup/freezer cgroup rw,relatime,freezer 0 0
cgroup /cgroup/cpuset cgroup rw,relatime,cpuset 0 0
cgroup /cgroup/cpuacct cgroup rw,relatime,cpuacct 0 0
cgroup /cgroup/memory cgroup rw,relatime,memory 0 0
```

If you no longer need the cgroup subsystem mounted, you can use the command **umount -a -t cgroup** to dismount all cgroup type mounting points listed in `/etc/fstab`.

You can also dismount them individually, such as:

```
umount /cgroup/freezer
umount /cgroup/cpuset
```

```
umount /cgroup/cpuacct
umount /cgroup/memory
```

Submit jobs as other users

Use the **bsubmit** command with the `lsf.usermapping` configuration file to submit jobs as other users.

A job submission user can submit jobs as another job execution user. This is useful if you want the job submission user to be a particular user, but to map the job execution user to other users.

To enable this feature, you must download and deploy the **bsubmit** executable file, which is a wrapper for the **bsub** command. For more details, refer to the following link: <https://github.com/IBMSpectrumComputing/lsf-utils/tree/master/bsubmit>

To define the execution users to which you can map submission users, create a configuration file named `lsf.usermapping` in the `$LSF_ENVDIR` directory to define the user mapping policy for the **bsubmit** command. The `lsf.usermapping` file allows you to map several job execution users and user groups to a single submission user or user group. This file must be owned by the LSF administrator, with file permissions set to read/write for the owner and read-only for all other users.

For example,

#Submission user or group	# Execution users or groups
<code>userA</code>	<code>userB, userC, userD</code>
<code>groupA</code>	<code>groupB</code>

This `lsf.usermapping` configuration file means that the `userA` user can submit jobs as `userB`, `userC`, or `userD`. Users in the `groupA` group can submit jobs as any user in the `groupB` user group.

To submit jobs as other users, use the new `bsubmit` command. For example, run the following command if the job submission user `userA` is submitting a job as job execution user `userC`:

```
bsubmit --user userC myjob
```

External job submission and execution controls

The job submission and execution controls use external, site-specific executable files to validate, modify, and reject jobs; and to transfer data and modify the job execution environment.

By writing external submission (**esub**), external post-submission (**epsub**), and external execution (**eexec**) binary files or scripts, you can, for example, prevent the overuse of resources, specify execution hosts, or set required environment variables that are based on the job submission options. In addition, you can use external post-submission (**epsub**) binary files or scripts to communicate with external components using job submission information such as job ID or queue name.

- [About job submission and execution controls](#)
The job submission and execution controls feature uses the executable files **esub** and **eexec** to control job options and the job execution environment.
- [Configuration to enable job submission and execution controls](#)
Enable job submission and execution controls with at least one **esub**, **epsub**, or **eexec** executable file in the directory specified by the parameter `LSF_SERVERDIR` in the `lsf.conf` file. LSF does not include a default **esub**, **epsub**, or **eexec**; write your own executable files to meet the job requirements of your site.
- [Job submission and execution controls behavior](#)
The following examples illustrate how customized **esub**, **epsub**, and **eexec** executable files can control job submission and execution.
- [Configuration to modify job submission and execution controls](#)
There are configuration parameters that modify various aspects of job submission and execution controls behavior by:
- [Job submission and execution controls commands](#)
- [Command arguments for job submission and execution controls](#)
esub arguments provide flexibility for filtering and modifying job submissions by letting you specify options for **esub** executables. As of LSF release 9.1.1.1, **bsub -a** supports arguments for a given **esub** executable. Users can customize

their **esub** applications, put them under LSF_SERVERDIR, and then submit jobs as **bsub -a "application_name" user_job**.

About job submission and execution controls

The job submission and execution controls feature uses the executable files **esub** and **eexec** to control job options and the job execution environment.

In addition, the **esub** executable files can communicate with external components using job submission information such as job ID and queue name and perform additional logic after job submission.

External submission (esub)

An **esub** is an executable file that you write to meet the job requirements at your site. The following are some of the things that you can use an **esub** to do:

- Validate job options
- Change the job options that are specified by a user
- Change user environment variables on the submission host (at job submission only)
- Reject jobs (at job submission only)
- Pass data to stdin of **eexec**
- Automate job resource requirements
- Enable data provenance to trace job files

When a user submits a job by using **bsub** or modifies a job by using **bmod**, LSF runs the **esub** executable files on the submission host before the job is accepted. If the user submitted the job with options such as -R to specify required resources or -q to specify a queue, an **esub** can change the values of those options to conform to resource usage policies at your site.

Note: When compound resource requirements are used at any level, an **esub** can create job-level resource requirements, which overwrite most application-level and queue-level resource requirements.

An **esub** can also change the user environment on the submission host before job submission so that when LSF copies the submission host environment to the execution host, the job runs on the execution host with the values specified by the **esub**. For example, an **esub** can add user environment variables to those environment variables already associated with the job.

LSF runs the default executable file named "esub" if it exists in the LSF_SERVERDIR directory, followed by any mandatory **esub** executable files that are defined by LSB_ESUB_METHOD, followed by any application-specific **esub** executable files (with *.application_name* in the file name).

External post-submission (esub)

An **esub** is an executable file that you write to meet the post-submission job requirements at your site with information that is not available before job submission. The following are some of the things that you can use an **esub** to do with the newly-available job information:

- Pass job information to an external entity
- Post job information to a local log file
- Perform general logic after a job is submitted to LSF

When a user submits a job by using **bsub**, modifies a job by using **bmod**, or restarts a job by using **brestart**, LSF runs the **esub** executable files on the submission host immediately after the job is accepted, and the job may or may not have started running while **esub** is running.

When submitting interactive jobs, **bsub** or **bmod** runs **esub**, then resumes regular interactive job behavior (that is, **bsub** or **bmod** runs **esub**, then runs the interactive job).

esub does not pass information to **eexec**, nor does it get information from **eexec**. **esub** can only read information from the temporary file that contains job submission options (as indicated by the LSB_SUB_PARM_FILE environment variable) and from the environment variables. The information that is available to the **esub** after job submission includes the following:

- A temporary file that contains job submission options, which is available through the LSB_SUB_PARM_FILE environment variable. The file that this environment variable specifies is a different file from the one that is initially created by **esub** before the job submission.
- The LSF job ID, which is available through the LSB_SUB_JOB_ID environment variable. For job arrays, the job ID includes the job array index.
- The name of the final queue to which the job is submitted (including any queue modifications made by **esub**), which is available through the LSB_SUB_JOB_QUEUE environment variable.
- The LSF job error number if the job submission failed, which is available through the LSB_SUB_JOB_ERR environment variable.

Since **esub** is run after job submission, the **esub** executable files cannot modify job submission parameters or job environment variables. That is, LSB_SUB_MODIFY_FILE and LSB_SUB_MODIFY_ENVFILE are not available to **esub**.

If the **esub** rejects a job, the corresponding **esub** file does not run.

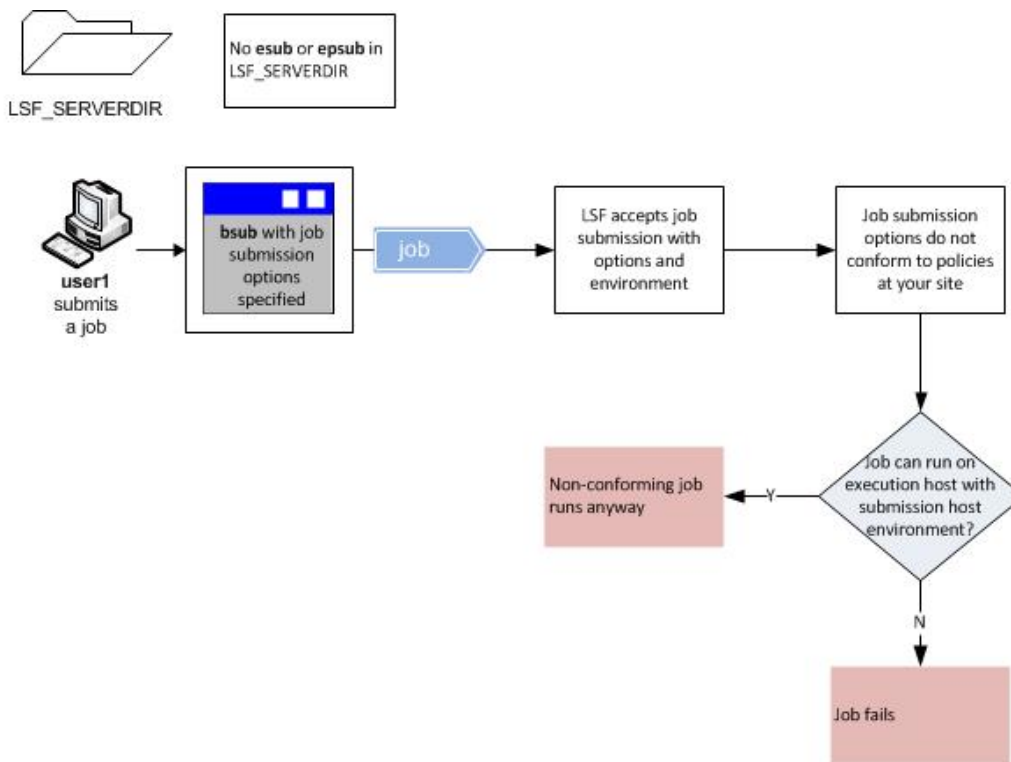
After job submission, **bsub** or **bmod** waits for the **esub** scripts to finish before returning. If the **bsub** or **bmod** return time is crucial, do not use **esub** to perform time-consuming activities. In addition, if **esub** hangs, **bsub** or **bmod** waits indefinitely for the **esub** script to finish. This is similar to the **esub** behavior, because **bsub** or **bmod** hangs if an **esub** script hangs.

LSF runs the default executable file named "esub" if it exists in the LSF_SERVERDIR directory, followed by any mandatory **esub** executable files that are defined by LSB_ESUB_METHOD, followed by any application-specific **esub** executable files (with *application_name* in the file name).

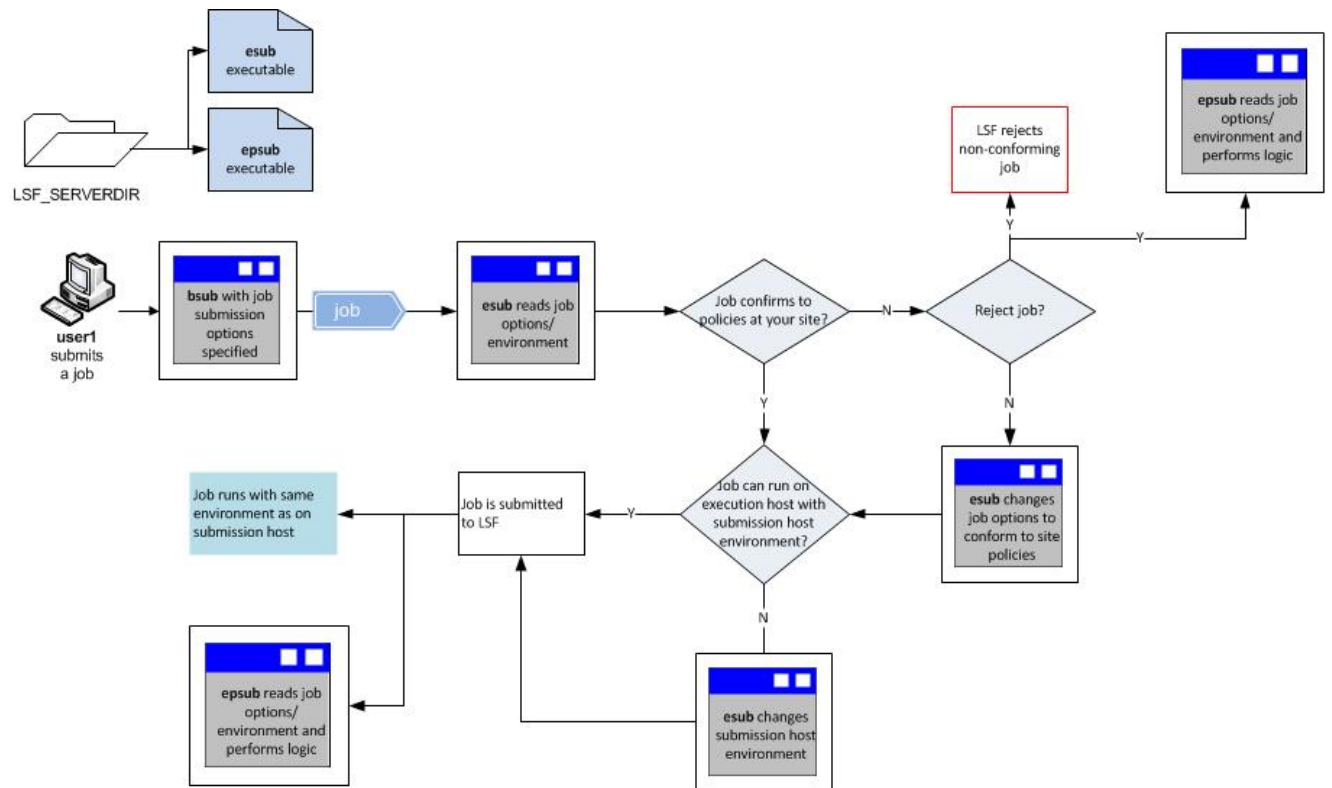
If a mandatory program specified using the LSB_ESUB_METHOD parameter does not have a corresponding **esub** executable file (*esub.application_name*), but has a corresponding **esub** executable file (*esub.application_name*), the job is submitted normally using the normal external job submission and post-submission mechanisms.

Except for these differences, **esub** uses the same framework as **esub**.

Use of esub or epsub not enabled



With esub or epsub enabled



An **esub** executable file is typically used to enforce site-specific job submission policies and command line syntax by validating or pre-parsing the command line. The file indicated by the environment variable `LSB_SUB_PARM_FILE` stores the values that are submitted by the user. An **esub** reads the `LSB_SUB_PARM_FILE` and then accepts or changes the option values or rejects the job. Because an **esub** runs before job submission, using an **esub** to reject incorrect job submissions improves overall system performance by reducing the load on the management batch daemon (**mbatchd**).

An **esub** can be used for the following purposes:

- Reject any job that requests more than a specified number of CPUs
- Change the submission queue for specific user accounts to a higher priority queue
- Check whether the job specifies an application and, if so, submit the job to the correct application profile

Note: If an **esub** executable file fails, the job is still submitted to LSF.

Multiple esub executable files

LSF provides a parent external submission executable file (`LSF_SERVERDIR/mesub`) that supports the use of application-specific **esub** executable files. Users can specify one or more **esub** executable files by using the `-a` option of **bsub** or **bmod**. When a user submits or modifies a job or when a user restarts a job that was submitted or modified with the `-a` option included, **mesub** runs the specified **esub** executable files.

An LSF administrator can specify one or more mandatory **esub** executable files by defining the parameter `LSB_ESUB_METHOD` in `lsf.conf`. If a mandatory **esub** is defined, **mesub** runs the mandatory **esub** for all jobs that are submitted to LSF in addition to any **esub** executable files specified with the `-a` option.

The naming convention is `esub.application`. LSF always runs the executable file that is named "**esub**" (without `.application`) if it exists in `LSF_SERVERDIR`.

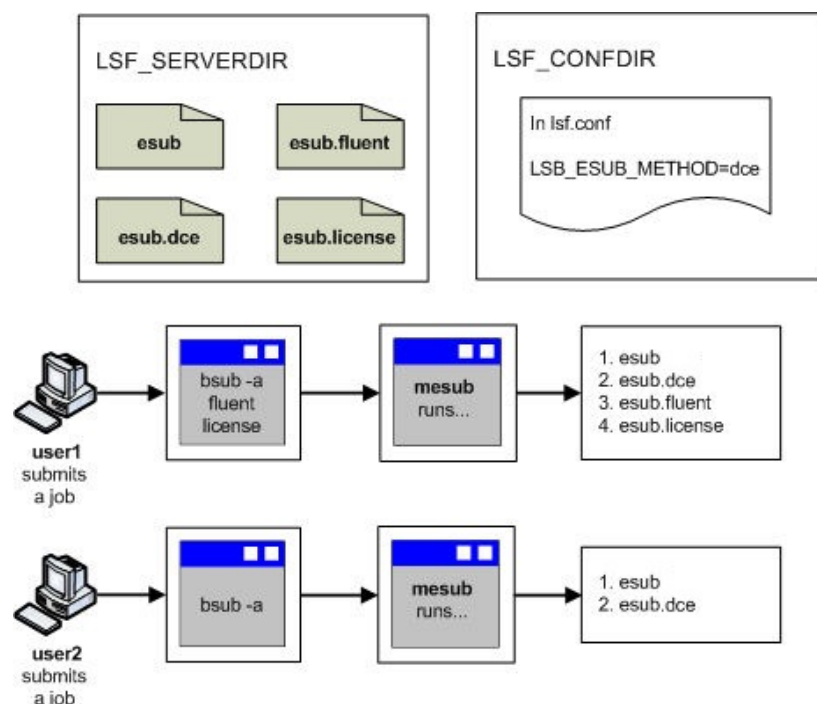
Note: All **esub** executable files must be stored in the `LSF_SERVERDIR` directory that is defined in `lsf.conf`.

The **mesub** runs multiple **esub** executable files in the following order:

1. Any executable file with the name "**esub**" in `LSF_SERVERDIR`
2. The mandatory **esub** or **esubs** specified by `LSB_ESUB_METHOD` in `lsf.conf`
3. One or more **esubs** in the order that is specified by **bsub -a**

Example of multiple esub execution

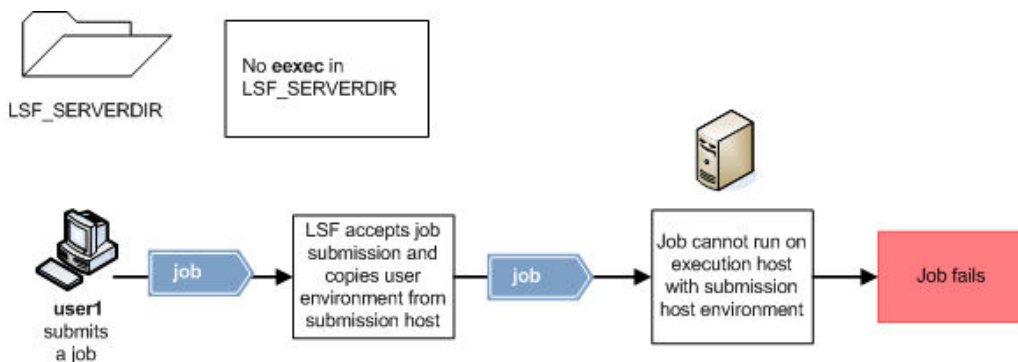
An **esub** runs only once, even if it is specified by both the **bsub** -a option and the parameter LSB_ESUB_METHOD.



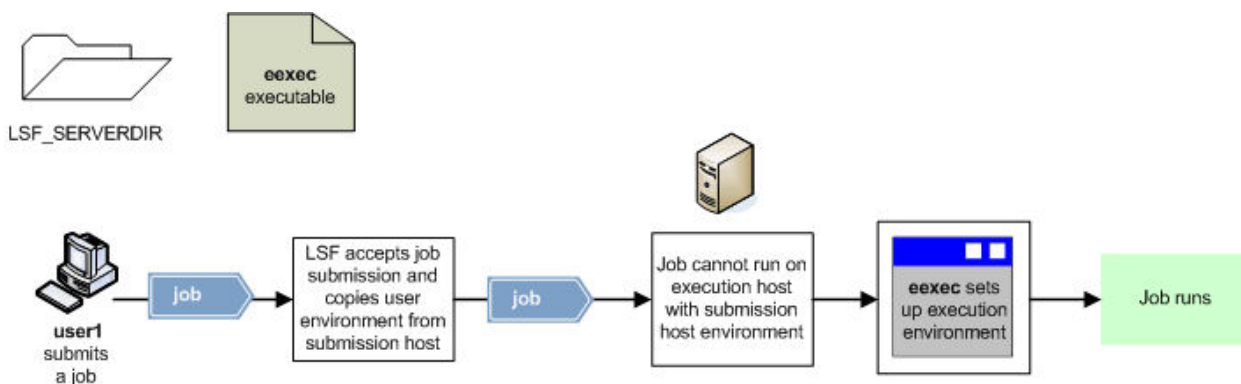
External execution (eexec)

An **eexec** is an executable file that you write to control the job environment on the execution host.

Use of eexec not enabled



With eexec enabled



The following are some of the things that you can use an **eexec** to do:

- Monitor job state or resource usage
- Receive data from stdout of **esub**
- Run a shell script to create and populate environment variables that are needed by jobs
- Monitor the number of tasks that are running on a host and raise a flag when this number exceeds a pre-determined limit
- Pass DCE credentials and AFS tokens by using a combination of **esub** and **eexec** executable files; LSF functions as a pipe for passing data from the stdout of **esub** to the stdin of **eexec**

For example, if you have a mixed UNIX and Windows cluster, the submission and execution hosts might use different operating systems. In this case, the submission host environment might not meet the job requirements when the job runs on the execution host. You can use an **eexec** to set the correct user environment between the two operating systems.

Typically, an **eexec** executable file is a shell script that creates and populates the environment variables that are required by the job. An **eexec** can also monitor job execution and enforce site-specific resource usage policies.

If an **eexec** executable file exists in the directory that is specified by LSF_SERVERDIR, LSF starts that **eexec** for all jobs that are submitted to the cluster. By default, LSF runs **eexec** on the execution host before the job starts. The job process that starts **eexec** waits for **eexec** to finish before the job continues with job execution.

Unlike a pre-execution command that is defined at the job, queue, or application levels, an **eexec**:

- Runs at job start, finish, or checkpoint
- Allows the job to run without pending if **eexec** fails; **eexec** has no effect on the job state
- Runs for all jobs, regardless of queue or application profile

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX and Linux • Windows
Security	<ul style="list-style-type: none"> • Data passing between esub on the submission host and eexec on the execution host is not encrypted.
Job types	<ul style="list-style-type: none"> • Batch jobs that are submitted with the bsub command or modified by the bmod command. • Batch jobs that are restarted with the brestart command. • Interactive tasks that are executed remotely by the following commands: <ul style="list-style-type: none"> ◦ lsrun ◦ lsgrun ◦ lsmake
Dependencies	<ul style="list-style-type: none"> • UNIX and Windows user accounts must be valid on all hosts in the cluster, or the correct type of account mapping must be enabled. <ul style="list-style-type: none"> ◦ For a mixed UNIX and Windows cluster, UNIX and Windows user account mapping must be enabled. ◦ For a cluster with a non-uniform user name space, between-host account mapping must be enabled. ◦ For a mulicuster environment with a non-uniform user name space, cross-cluster user account mapping must be enabled. • User accounts must have the correct permissions to successfully run jobs. • An eexec that requires root privileges to run on UNIX, must be configured to run as the root user.

Applicability	Details
Limitations	<ul style="list-style-type: none"> Only an esub started by bsub can change the job environment on the submission host. An esub started by bmod or brestart cannot change the environment. Any esub messages that are provided to the user must be directed to standard error, not to standard output. Standard output from any esub is automatically passed to eexec. An eexec can handle only one standard output stream from an esub as standard input to eexec. You must make sure that your eexec handles standard output from correctly if any esub writes to standard output. The esub and eexec combination cannot handle daemon authentication. To configure daemon authentication, you must enable external authentication, which uses the eauth executable file.

Configuration to enable job submission and execution controls

Enable job submission and execution controls with at least one **esub**, **epsub**, or **eexec** executable file in the directory specified by the parameter LSF_SERVERDIR in the lsf.conf file. LSF does not include a default **esub**, **epsub**, or **eexec**; write your own executable files to meet the job requirements of your site.

Executable file	UNIX naming convention	Windows naming convention
esub	LSF_SERVERDIR/esub.application	LSF_SERVERDIR\esub.application.exe LSF_SERVERDIR\esub.application.bat
epsub	LSF_SERVERDIR/epsub.application	LSF_SERVERDIR\epsub.application.exe LSF_SERVERDIR\epsub.application.bat
eexec	LSF_SERVERDIR/eexec	LSF_SERVERDIR\eexec.exe LSF_SERVERDIR\eexec.bat

The name of your **esub/epsub** indicates the application with which it runs. For example: esub.fluent or epsub.fluent.

Restriction: The names esub.user and epsub.user are reserved. Do not use esub.user and epsub.user for application-specific **esub** and **epsub** executable files.

Valid file names contain only alphanumeric characters, underscores (_), and hyphens (-).

Once the LSF_SERVERDIR contains one or more **esub/epsub** executable files, users can specify the **esub/epsub** executable files that are associated with each job they submit. If an **eexec** exists in LSF_SERVERDIR, LSF invokes that **eexec** for all jobs that are submitted to the cluster.

The following **esub** executable files are provided as separate packages, available from IBM upon request:

- **esub.afs** or **esub.dce**: for installing LSF onto an AFS or DCE filesystem
- **esub.bproc**: Beowulf Distributed Process Space (BProc) job submission
- **esub.checkcmd**: Check **bsub** option arguments.
- **esub.dprov**: Data provenance options for job submission
- **esub.fluent**: FLUENT job submission
- **esub.intelmpi**: Intel® MPI job submission
- **esub.lammpi**: LAM/MPI job submission
- **esub.ls_dyna**: LS-Dyna job submission
- **esub.mpich_gm**: MPICH-GM job submission
- **esub.mpich2**: MPICH2 job submission
- **esub.mpichp4**: MPICH-P4 job submission
- **esub.mvapich**: MVAPICH job submission
- **esub.openmpi**: OpenMPI job submission
- **esub.p8aff**: POWER8 affinity job submission
- **esub.poe**: POE job submission
- **esub.pvm**: PVM job submission
- **esub.tv**, **esub.tvlammpi**, **esub.tvmpich_gm**, **esub.tvpoe**: TotalView® debugging for various MPI applications.

Environment variables used by esub

When you write an **esub**, you can use the following environment variables that are provided by LSF for the **esub** execution environment:

LSB_SUB_PARM_FILE

Points to a temporary file that LSF uses to store the **bsub** options that are entered in the command line. An **esub** reads this file at job submission and either accepts the values, changes the values, or rejects the job. Job submission options are stored as name-value pairs on separate lines with the format option_name=value.

For example, if a user submits the following job:

```
bsub -q normal -x -P myproject -R "rlm rusage[mem=100]" -n 90 myjob
```

The LSB_SUB_PARM_FILE contains the following lines:

```
LSB_SUB_QUEUE="normal"
LSB_SUB_EXCLUSIVE=Y
LSB_SUB_RES_REQ="rlm usage[mem=100]"
LSB_SUB_PROJECT_NAME="myproject"
LSB_SUB_COMMAND_LINE="myjob"
LSB_SUB_NUM_PROCESSORS=90
LSB_SUB_MAX_NUM_PROCESSORS=90
LSB_SUB_MEM_USAGE=100
```

An **esub** can change any or all of the job options by writing to the file specified by the environment variable LSB_SUB_MODIFY_FILE.

The temporary file pointed to by LSB_SUB_PARM_FILE stores the following information:

Option	bsub or bmod option	Description
LSB_SUB_ADDITIONAL	-a	String that contains the application name or names of the esub executable files that are requested by the user. Restriction: The -a option is the only option that an esub cannot change or add at job submission.
LSB_SUB_BEGIN_TIME	-b	Begin time, in seconds since 00:00:00 GMT, 1 January 1970.
LSB_SUB_CHECKPOINT_DIR	-k	Checkpoint directory The file path of the checkpoint directory can contain up to 4000 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.
LSB_SUB_COMMAND_LINE	bsub job command argument	The LSB_SUB_COMMANDNAME parameter must be set in the lsf.conf parameter to enable esub to use this variable.
LSB_SUB_CHECKPOINT_PERIOD	-k	Checkpoint period in seconds
LSB_SUB3_CWD	-cwd	Current working directory
LSB_SUB_DEPENDENCY_CONDITION	-w	Dependency condition
LSB_SUB_ERROR_FILE	-e, -eo	Standard error file name
LSB_SUB_EXCLUSIVE	-x	Exclusive execution, which is specified by Y.
LSB_SUB_HOLD	-H	Hold job.
LSB_SUB_HOST_SPEC	-c or -w	Host specifier, limits the CPU time or RUN time.
LSB_SUB_HOSTS	-m	List of requested execution host names
LSB_SUB_INPUT_FILE	-i, -io	Standard input file name
LSB_SUB_INTERACTIVE	-I	Interactive job, which is specified by Y.

Option	bsub or bmod option	Description
LSB_SUB6_JOB_AFF	-jobaff	Job's affinity preferences
LSB_SUB_JOB_DESCRIPTION	-Jd	Job description
LSB_SUB_JOB_NAME	-J	Job name
LSB_SUB_LOGIN_SHELL	-L	Login shell
LSB_SUB_MAIL_USER	-u	Email address to which LSF sends job-related messages.
LSB_SUB_MEM_USAGE	-R "rusage[mem=value]"	Specifies the mem value in the <code>rusage[]</code> string.
LSB_SUB_MAX_NUM_PROCESSORS	-n	Maximum number of processors requested
LSB_SUB_SWP_USAGE	-R "rusage[swp=value]"	Specifies the swp value in the <code>rusage[]</code> string.
LSB_MC_SUB_CLUSTERS	-clusters	Cluster names
LSB_SUB_MODIFY	bmod	Indicates that bmod invoked esub , specified by <code>Y</code> .
LSB_SUB_MODIFY_ONCE	bmod	Indicates that the job options that are specified at job submission are already modified by bmod , and that bmod is invoking esub again. This is specified by <code>Y</code> .
LSB_SUB4_NETWORK	-network	Defines network requirements before job submission
LSB_SUB4_ORPHAN_TERM_NO_WAIT	-ti	Tells LSF to terminate an orphaned job immediately (ignores the grace period).
LSB_SUB4_ELIGIBLE_PEND_TIME_LIMIT	-eptl	The eligible pending time limit for the job. LSB_SUB4_ELIGIBLE_PEND_TIME_LIMIT= [hour:]minute if bsub -eptl or bmod -eptl is specified. LSB_SUB4_ELIGIBLE_PEND_TIME_LIMIT= SUB_RESET if bmod -eptln is specified.
LSB_SUB4_PEND_TIME_LIMIT	-ptl	The pending time limit for the job. LSB_SUB4_PEND_TIME_LIMIT= [hour:]minute if bsub -ptl or bmod -ptl is specified. LSB_SUB4_PEND_TIME_LIMIT= SUB_RESET if bmod -ptln is specified.
LSB_SUB_NOTIFY_BEGIN	-B	LSF sends an email notification when the job begins, specified by <code>Y</code> .
LSB_SUB_NOTIFY_END	-N	LSF sends an email notification when the job ends, which are specified by <code>Y</code> .
LSB_SUB_NUM_PROCESSORS	-n	Minimum number of processors requested.
LSB_SUB_OTHER_FILES	bmod -f	Indicates the number of files to be transferred. The value is SUB_RESET if bmod is being used to reset the number of files to be transferred. The file path of the directory can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the director and file name.

Option	bsub or bmod option	Description
LSB_SUB_OTHER_FILES _number	bsub -f	The <i>number</i> indicates the particular file transfer value in the specified file transfer expression. For example, for bsub -f "a > b" -f "c < d" , the following parameters are defined: LSB_SUB_OTHER_FILES=2 LSB_SUB_OTHER_FILES_0="a > b" LSB_SUB_OTHER_FILES_1="c < d"
LSB_SUB4_OUTDIR	-outdir	Output directory
LSB_SUB_OUTFILE	-o, -oo	Standard output file name.
LSB_SUB_PREEXEC	-E	Pre-execution command. The file path of the directory can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.
LSB_SUB_PROJECT_NAME	-P	Project name.
LSB_SUB_PTY	-Ip	An interactive job with PTY support, which is specified by "Y"
LSB_SUB_PTY_SHELL	-Is	An interactive job with PTY shell support, which is specified by "Y"
LSB_SUB_QUEUE	-q	Submission queue name
LSB_SUB_RERUNNABLE	-r	Y specifies a rerunnable job. N specifies a non-rerunnable job (specified with bsub -rn). The job is not rerunnable even it was submitted to a rerunnable queue or application profile. For bmod -rn , the value is SUB_RESET.
LSB_SUB_RESOURCE_REQ	-R	Resource requirement string— <i>does not</i> support multiple resource requirement strings.
LSB_SUB_RESTART	brestart	Y indicates to esub that the job options are associated with a restarted job.
LSB_SUB_RESTART_FORCE	brestart -f	Y indicates to esub that the job options are associated with a forced restarted job.
LSB_SUB_RLIMIT_CORE	-C	Core file size limit
LSB_SUB_RLIMIT_CPU	-c	CPU limit
LSB_SUB_RLIMIT_DATA	-D	Data size limit For AIX, if the XPG_SUS_ENV=ON environment variable is set in the user's environment before the process is executed and a process attempts to set the limit lower than current usage, the operation fails with errno set to EINVAL. If the XPG_SUS_ENV environment variable is not set, the operation fails with errno set to EFAULT.
LSB_SUB_RLIMIT_FSIZE	-F	File size limit
LSB_SUB_RLIMIT_PROCESS	-p	Process limit
LSB_SUB_RLIMIT_RSS	-M	Resident size limit
LSB_SUB_RLIMIT_RUN	-W	Wall-clock run limit in seconds. (Note this value is not in minutes, unlike the run limit specified by bsub -W).
LSB_SUB_RLIMIT_STACK	-S	Stack size limit

Option	bsub or bmod option	Description
LSB_SUB_RLIMIT_SWAP	-v	Process virtual memory limit
LSB_SUB_RLIMIT_THREAD	-T	Thread limit
LSB_SUB_TERM_TIME	-t	Termination time, in seconds, since 00:00:00 GMT, Jan. 1, 1970
LSB_SUB_TIME_EVENT	-wt	Time event expression
LSB_SUB_USER_GROUP	-G	User group name
LSB_SUB_JOB_WARNING_ACTION	-wa	Job warning action
LSB_SUB_JOB_ACTION_WARNING_TIME	-wt	Job warning time period
LSB_SUB_WINDOW_SIG	-s	Window signal number
LSB_SUB2_JOB_GROUP	-g	Submits a job to a job group
LSB_SUB2_LICENSE_PROJECT	-lp	License Scheduler project name
LSB_SUB2_IN_FILE_SPOOL	-is	Spooled input file name
LSB_SUB2_JOB_CMD_SPOOL	-Zs	Spooled job command file name
LSB_SUB2_JOB_PRIORITY	-sp	Job priority For bmod -spn , the value is SUB_RESET.
LSB_SUB2_SLA	-sla	SLA scheduling options
LSB_SUB2_USE_RSV	-U	Advance reservation ID
LSB_SUB3_ABSOLUTE_PRIORITY	bmod -aps bmod -apsn	For bmod -aps , the value equal to the APS string given. For bmod -apsn , the value is SUB_RESET.
LSB_SUB3_AUTO_RESIZABLE	-ar	Job autoresizable attribute. LSB_SUB3_AUTO_RESIZABLE=Y if bsub -ar -app or bmod -ar is specified. LSB_SUB3_AUTO_RESIZABLE= SUB_RESET if bmod -arn is used.
LSB_SUB3_APP	-app	Application profile name For bmod -appn , the value is SUB_RESET.
LSB_SUB3_CWD	-cwd	Current working directory
LSB_SUB3_INIT_CHKPNT_PERIOD	-k init	Initial checkpoint period

Option	bsub or bmod option	Description
LSB_SUB_INTERACTIVE LSB_SUB3_INTERACTIVE_SSH	bsub -IS	The session of the interactive job is encrypted with SSH.
LSB_SUB_INTERACTIVE LSB_SUB_PTY LSB_SUB3_INTERACTIVE_SSH	bsub -ISp	If LSB_SUB_INTERACTIVE is specified by "Y", LSB_SUB_PTY is specified by "Y", and LSB_SUB3_INTERACTIVE_SSH is specified by "Y", the session of interactive job with PTY support is encrypted by SSH.
LSB_SUB_INTERACTIVE LSB_SUB_PTY LSB_SUB_PTY_SHELL LSB_SUB3_INTERACTIVE_SSH	bsub -ISs	If LSB_SUB_INTERACTIVE is specified by "Y", LSB_SUB_PTY is specified by "Y", LSB_SUB_PTY_SHELL is specified by "Y", and LSB_SUB3_INTERACTIVE_SSH is specified by "Y", the session of interactive job with PTY shell support is encrypted by SSH.
LSB_SUB3_JOB_REQUEUE	-Q	String format parameter that contains the job requeue exit values For bmod -Qn , the value is SUB_RESET.
LSB_SUB3_MIG	-mig -mign	Migration threshold
LSB_SUB3_POST_EXEC	-Ep	Run the specified post-execution command on the execution host after the job finishes (you must specify the first execution host). The file path of the directory can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.
LSB_SUB6_RC_ACCOUNT	-rcacct	LSF resource connector account name that is assigned to a job, which is then tagged to the resource connector host that runs the job.
LSB_SUB3_RESIZE_NOTIFY_CMD	-rnc	Job resize notification command. LSB_SUB3_RESIZE_NOTIFY_CMD=<cmd> if bsub -rnc or bmod -rnc is specified. LSB_SUB3_RESIZE_NOTIFY_CMD =SUB_RESET if bmod -rnc is used.
LSB_SUB3_RUNTIME_ESTIMATION	-We	Runtime estimate in seconds. (Note this runtime is not in minutes, unlike the runtime estimate specified by bsub -We).
LSB_SUB3_RUNTIME_ESTIMATION_ACC	-We+	Runtime estimate that is the accumulated run time plus the runtime estimate.
LSB_SUB3_RUNTIME_ESTIMATION_PERC	-Wep	Runtime estimate in percentage of completion
LSB_SUB3_USER_SHELL_LIMITS	-ul	Pass user shell limits to execution host.
LSB_SUB_INTERACTIVESUB3_XJOB_SSH	bsub -IX	If both are set to "Y", the session between the X-client and X-server as well as the session between the execution host and submission host are encrypted with SSH.

Option	bsub or bmod option	Description
LSF_SUB4_SUB_ENV_VARS	-env	Controls the propagation of job submission environment variables to the execution hosts. If any environment variables in LSF_SUB4_SUB_ENV_VARS conflict with the contents of the LSB_SUB_MODIFY_ENVFILE file, the conflicting environment variables in LSB_SUB_MODIFY_ENVFILE take effect.

LSB_SUB_MODIFY_FILE

Points to the file that **esub** uses to modify the **bsub** job option values that are stored in the LSB_SUB_PARM_FILE. You can change the job options by having your **esub** write the new values to the LSB_SUB_MODIFY_FILE in any order by using the same format shown for the LSB_SUB_PARM_FILE. The value SUB_RESET, integers, and boolean values do not require quotes. String parameters must be entered with quotes around each string, or space-separated series of strings. When your **esub** runs at job submission, LSF checks the LSB_SUB_MODIFY_FILE and applies changes so that the job runs with the revised option values.

Restriction:

LSB_SUB_ADDITIONAL is the only option that an **esub** cannot change or add at job submission.

LSB_SUB_MODIFY_ENVFILE

Points to the file that **esub** uses to modify the user environment variables with which the job is submitted (not specified by **bsub** options). You can change these environment variables by having your **esub** write the values to the LSB_SUB_MODIFY_ENVFILE in any order by using the format variable_name=value, or variable_name="string". LSF uses the LSB_SUB_MODIFY_ENVFILE to change the environment variables on the submission host. When your **esub** runs at job submission, LSF checks the LSB_SUB_MODIFY_ENVFILE and applies changes so that the job is submitted with the new environment variable values. LSF associates the new user environment with the job so that the job runs on the execution host with the new user environment.

LSB_SUB_ABORT_VALUE

Indicates to LSF that a job is rejected. For example, if you want LSF to reject a job, make sure that your **esub** contains the following line:

```
exit $LSB_SUB_ABORT_VALUE
```

Restriction: When an **esub** exits with the LSB_SUB_ABORT_VALUE, **esub** must not write to LSB_SUB_MODIFY_FILE or to LSB_SUB_MODIFY_ENVFILE.

If multiple **esubs** are specified and one of the **esubs** exits with a value of LSB_SUB_ABORT_VALUE, LSF rejects the job without running the remaining **esubs** and returns a value of LSB_SUB_ABORT_VALUE.

LSF_INVOKE_CMD

Specifies the name of the LSF command that most recently invoked an external executable.

The length of environment variables that are used by **esub** must be less than 4096.

Environment variables used by epsub

When you write an **epsub**, you can use the following environment variables that are provided by LSF for the **epsub** execution environment:

LSB_SUB_JOB_ERR

Indicates the error number for an externally submitted job that is defined by **mbatchd** if the job submission failed. This variable is available to the external post-submission scripts (**epsub**) to determine the reason for the job submission failure.

If the job submission is successful, this value is LSB_NO_ERROR (or 0).

LSB_SUB_JOB_ID

Indicates the ID of a submitted job that is assigned by LSF, as shown by the **bjobs** command. A value of -1 indicates that **mbatchd** rejected the job submission.

LSB_SUB_JOB_QUEUE

Indicates the name of the final queue from which the job is dispatched, which includes any queue modifications that are made by **esub**.

LSB_SUB_PARM_FILE

Points to a temporary file that LSF uses to store the **bsub** options that are entered in the command line. Job submission options are stored as name-value pairs on separate lines in the format option_name=value. The file that this environment variable specifies is a different file from the one that is initially created by **esub** before the job submission.

In addition to the environment variables available to **epsub**, you can also use the environment variables that are provided by LSF for the **esub** execution environment, except for LSB_SUB_MODIFY_FILE and LSB_SUB_MODIFY_ENVFILE.

Environment variables used by eexec

When you write an **eexec**, you can use the following environment variables in addition to all user-environment or application-specific variables.

LS_EXEC_T

Indicates the stage or type of job execution. LSF sets LS_EXEC_T to:

- START at the beginning of job execution
- END at job completion
- CHPNT at job checkpoint start

LS_JOBPID

Stores the process ID of the LSF process that invoked **eexec**. If **eexec** is intended to monitor job execution, **eexec** must spawn a child and then have the parent **eexec** process exit. The **eexec** child can periodically test that the job process is still alive by using the LS_JOBPID variable.

Job submission and execution controls behavior

The following examples illustrate how customized **esub**, **epsub**, and **eexec** executable files can control job submission and execution.

Validating job submission parameters by using esub

When a user submits a job by using the **bsub -P** command option, LSF accepts any project name that is entered by the user and associates that project name with the job. This example shows an **esub** that supports project-based accounting by enforcing the use of valid project names for jobs that are submitted by users who are eligible to charge to those projects. If a user submits a job to any project other than proj1 or proj2, or if the user name is not user1 or user2, LSF rejects the job based on the exit value of LSB_SUB_ABORT_VALUE.

```
#!/bin/sh

. $LSB_SUB_PARM_FILE

# Redirect stderr to stdout so echo can be used for error messages exec 1>&2
# Check valid projects
if [ $LSB_SUB_PROJECT_NAME != "proj1" -o $LSB_SUB_PROJECT_NAME != "proj2" ]; then
    echo "Incorrect project name specified"
    exit $LSB_SUB_ABORT_VALUE
fi

USER=`whoami`
if [ $LSB_SUB_PROJECT_NAME="proj1" ]; then
    # Only user1 and user2 can charge to proj1
    if [ $USER != "user1" -a $USER != "user2" ]; then
        echo "You are not allowed to charge to this project"
        exit $LSB_SUB_ABORT_VALUE
    fi
fi
```

Changing job submission parameters by using esub

The following example shows an **esub** that modifies job submission options and environment variables based on the user name that submits a job. This **esub** writes the changes to `LSB_SUB_MODIFY_FILE` for userA and to `LSB_SUB_MODIFY_ENVFILE` for userB. LSF rejects all jobs that are submitted by userC without writing to either file:

```
#!/bin/sh
. $LSB_SUB_PARM_FILE

# Redirect stderr to stdout so echo can be used for error messages exec 1>&2
USER=`whoami`
# Make sure userA is using the right queue queueA
if [ $USER="userA" -a $LSB_SUB_QUEUE != "queueA" ]; then
    echo "userA has submitted a job to an incorrect queue"
    echo "...submitting to queueA"
    echo 'LSB_SUB_QUEUE="queueA"' > $LSB_SUB_MODIFY_FILE
fi

# Make sure userB is using the right shell (/bin/sh)
if [ $USER="userB" -a $SHELL != "/bin/sh" ]; then
    echo "userB has submitted a job using $SHELL"
    echo "...using /bin/sh instead"
    echo 'SHELL="/bin/sh"' > $LSB_SUB_MODIFY_ENVFILE
fi

# Deny userC the ability to submit a job
if [ $USER="userC" ]; then
    echo "You are not permitted to submit a job."
    exit $LSB_SUB_ABORT_VALUE
fi
```

Monitoring the execution environment by using eexec

This example shows how you can use an **eexec** to monitor job execution:

```
#!/bin/sh
# eexec
# Example script to monitor the number of jobs executing through RES.
# This script works in cooperation with an elim that counts the
# number of files in the TASKDIR directory. Each RES process on a host
# will have a file in the TASKDIR directory.
# Don't want to monitor lsbatch jobs.
if [ "$LSB_JOBID" != "" ] ; then
    exit 0
fi

TASKDIR="/tmp/RES_dir"
# directory containing all the task files
# for the host.
# you can change this to whatever
# directory you wish, just make sure anyone
# has read/write permissions.

# if TASKDIR does not exist create it

if [ "test -d $TASKDIR" != "0" ] ; then
    mkdir $TASKDIR > /dev/null 2>&1
fi

# Need to make sure LS_JOBPID, and USER are defined
# exit normally
if [ "test -z $LS_JOBPID" = "0" ] ; then
    exit 0
elif [ "test -z $USER" = "0" ] ; then
    exit 0
fi

taskFile="$TASKDIR/$LS_JOBPID.$USER"

# Fork grandchild to stay around for the duration of the task

touch $taskFile >/dev/null 2>&1
```



```
(
    (while : ;
    do
        kill -0 $LS_JOBPID >/dev/null 2>&1
        if [ $? -eq 0 ] ; then
            sleep 10 # this is the poll interval
                    # increase it if you want but
                    # see the elim for its
                    # corresponding update interval
        else
            rm $taskFile >/dev/null 2>&1
            exit 0
        fi
    done) &
wait
```

Monitoring job submission information by using ebsub

This example shows how you can use an **esub** to monitor job submission:

```
#!/bin/sh
# ebsub
# Example script to monitor job submissions to mbatchd.
# This script outputs the final job submission parameters after the
# job is submitted.
exec 1>&2
. $LSB_SUB_PARM_FILE
echo I am ebsub app >>/home/user1/esub.out

echo $LSB_SUB_JOB_QUEUE t
echo $LSB_SUB_JOB_ID >> /home/user1/esub.$LSB_SUB_JOB_ID
echo $LSB_SUB_JOB_ERR
```

Passing data between esub and eexec

A combination of **esub** and **eexec** executable files can be used to pass AFS/DCE tokens from the submission host to the execution host. LSF passes data from the standard output of **esub** to the standard input of **eexec**. A daemon wrapper script can be used to renew the tokens.

Configuration to modify job submission and execution controls

There are configuration parameters that modify various aspects of job submission and execution controls behavior by:

- Defining a mandatory **esub/esub** that applies to all jobs in the cluster.
- Specifying the **eexec** user account (UNIX only).

Configuration to define a mandatory esub/esub

Configuration file	Parameter and syntax	Behavior
lsf.conf	LSB_ESUB_METHOD=" <i>application_name</i> [<i>application_name</i>] ..."	<ul style="list-style-type: none"> • The specified esub/esub or esubs/esubs run for all jobs submitted to the cluster, in addition to any esub/esub specified by the user in the command line • For example, to specify a mandatory esub/esub named esub.fluent/esub.fluent, define <code>LSB_ESUB_METHOD=fluent</code>

Configuration to specify the eexec user account

The **eexec** executable runs under the submission user account. You can modify this behavior for UNIX hosts by specifying a different user account.

Configuration file	Parameter and syntax	Behavior
lsf.sudoers	LSF_EEXEC_USER= <i>user_name</i>	<ul style="list-style-type: none">Changes the user account under which eexec runs

Job submission and execution controls commands

Commands for submission

Command	Description
bsub -a <i>application_name</i> [<i>application_name</i>] ...	<ul style="list-style-type: none">Specifies one or more esub/epsub executable files to run at job submissionFor example, to specify the esub/epsub named esub.fluent/epsub.fluent, use bsub -a fluentLSF runs the executable file named "esub" if it exists in the LSF_SERVERDIR directory, followed by any esub executable files that are defined by LSB_ESUB_METHOD, followed by the esub executable files that are specified by the -a optionLSF runs eexec if an executable file with that name exists in LSF_SERVERDIRAfter the job is submitted, LSF runs the executable file named "epsub" if it exists in the LSF_SERVERDIR directory, followed by any epsub executable files that are defined by LSB_ESUB_METHOD, followed by the epsub executable files that are specified by the -a option
brestart	<ul style="list-style-type: none">Restarts a checkpointed job and runs the esub/epsub executable files specified when the job was submittedLSF runs the executable file named "esub" if it exists in the LSF_SERVERDIR directory, followed by any esub executable files that are defined by LSB_ESUB_METHOD, followed by the esub executable files that are specified by the -a optionLSF runs eexec if an executable file with that name exists in LSF_SERVERDIRAfter the job submission, LSF runs the executable file named "epsub" if it exists in the LSF_SERVERDIR directory, followed by any epsub executable files that are defined by LSB_ESUB_METHOD, followed by the epsub executable files that are specified by the -a option
lrun	<ul style="list-style-type: none">Submits an interactive task; LSF runs eexec if an eexec executable exists in LSF_SERVERDIRLSF runs eexec only at task startup (LS_EXEC_T=START)
lsgrun	<ul style="list-style-type: none">Submits an interactive task to run on a set of hosts; LSF runs eexec if an eexec executable exists in LSF_SERVERDIRLSF runs eexec only at task startup (LS_EXEC_T=START)

Commands to monitor

Not applicable: There are no commands to monitor the behavior of this feature.

Commands to control

Command	Description
---------	-------------

Command	Description
bmod -a <i>application_name</i> [<i>application_name</i>] ...	<ul style="list-style-type: none"> Resubmits a job and changes the esub/epsub previously associated with the job LSF runs the executable file named "esub" if it exists in the LSF_SERVERDIR directory, followed by any esub executable files that are defined by LSB_ESUB_METHOD, followed by the esub executable files that are specified by the -a option LSF runs eexec if an executable file with that name exists in LSF_SERVERDIR After the job submission, LSF runs the executable file named "epsub" if it exists in the LSF_SERVERDIR directory, followed by any epsub executable files that are defined by LSB_ESUB_METHOD, followed by the epsub executable files that are specified by the -a option
bmod -an	<ul style="list-style-type: none"> Dissociates from a job all esub/epsub executable files that were previously associated with the job LSF runs the executable file named "esub" if it exists in the LSF_SERVERDIR directory, followed by any esub executable files that are defined by LSB_ESUB_METHOD, followed by the esub executable files that are specified by the -a option LSF runs eexec if an executable file with that name exists in LSF_SERVERDIR After the job submission, LSF runs the executable file named "epsub" if it exists in the LSF_SERVERDIR directory, followed by any epsub executable files that are defined by LSB_ESUB_METHOD.

Commands to display configuration

Command	Description
badmin showconf	<ul style="list-style-type: none"> Displays all configured parameters and their values set in lsf.conf or ego.conf that affect mbatchd and sbatchd. Use a text editor to view other parameters in the lsf.conf or ego.conf configuration files. When using the LSF multicluster capability, displays the parameters of daemons on the local cluster.

Use a text editor to view the lsf.sudoers configuration file.

Command arguments for job submission and execution controls

esub arguments provide flexibility for filtering and modifying job submissions by letting you specify options for **esub** executables. As of LSF release 9.1.1.1, **bsub -a** supports arguments for a given **esub** executable. Users can customize their **esub** applications, put them under LSF_SERVERDIR, and then submit jobs as **bsub -a "application_name" user_job**.

Specifying **esub** arguments means it is unnecessary to write scripts for different permutations of input. For example, to check if the resource requirements exceed some bound, an argument for specifying the bound can be passed to the **esub** executable. It is not necessary to write a separate script for every bound.

As another example, in the case of Energy Aware Scheduling, a user may want to specify a certain energy or performance goal. Instead of providing and maintaining a separate **esub** for each possible choice (for example, **bsub -a energy_hi energy_low energy_max_performance** etc.), one **esub** can handle all the related options (for example, "**-a eas=a,b,c**").

You can:

- Specify arguments for **esub** executables with command **bsub -a**
- Modify arguments for **esub** executables for a submitted job with command **bmod -a**
- Specify arguments for **esub** executables when restarting a job with command **brestart -a**

The following are some examples of how to specify arguments for **esub** executables:

- To specify a single argument for a single **esub** executable, use:
bsub -a "application_name(var1)" user_job

- To specify multiple arguments for a single **esub** executable, use:
bsub -a “application_name(var1,var2,...,varN)” user_job
- To specify multiple arguments including a string argument for a single **esub** executable, use:
bsub -a “application_name(var1,var2 is a string,...,varN)” user_job
- To specify arguments for multiple **esub**, use:
bsub -a “application_name1(var1,var2) esubname2(var1,var2)” user_job
- To specify no argument to an **esub**, use:
bsub -a “application_name1” user_job

The variables you define in the **esub** arguments can include environment variables and command output substitution.

Valid **esub** arguments can contain alphanumeric characters, spaces, special characters (`"\$!) and other characters (~@#%^&*()-=_+[]|{};':./<>?). Special patterns like variables (e.g., \$PATH) and program output (e.g., `ls`) in an **esub** argument will also be processed.

For example, if you use **bsub -a “esub1 (\$PATH, `ls`)” user_job**, the first argument passed to **esub1** would be the value of variable *PATH*, and the second argument passed to **esub1** would be the output of command *ls*.

You can include a special character in an **esub** argument with an escape character or a pair of apostrophes ('). The usage may vary among different shells. You can specify an **esub** argument containing separators ('(', ')', ',') and space characters (' ').

You can also use an escape character '\ ' to specify arguments containing special characters, separators and space characters. For example:

bsub -a “esubname1(var1,var2 contains \(\)\,)” user_job

For fault tolerance, extra space characters are allowed between entities including **esub**, separators and arguments. For example, the following is valid input:

bsub -a “ esub1 (var1 , var2) ” user_job

The maximum length allowed for an **esub** argument is 1024 characters. The maximum number of arguments allowed for an **esub** is 128.

Note: The same arguments that are passed to **esub** are also passed to **esub**. You cannot pass different arguments to an **esub** file and an **esub** file with the same application name.

Interactive jobs and remote tasks

Run interactive jobs with the **bsub -I**, **bsub -Is**, and **bsub -Ip** commands to take advantage of batch scheduling policies and host selection features for resource-intensive jobs. Run tasks interactively and remotely with non-batch utilities such as **lrun** and **lsgun**.

- [Interactive jobs with bsub](#)
 - [Interactive and remote tasks](#)
- You can run tasks interactively and remotely with non-batch utilities such as **lrun** and **lsgun**.

Interactive jobs with bsub

- [About interactive jobs](#)
It is sometimes desirable from a system management point of view to control all workload through a single centralized scheduler.
- [Submit interactive jobs](#)
- [Performance tuning for interactive batch jobs](#)
LSF is often used on systems that support both interactive and batch users. On one hand, users are often concerned

that load sharing will overload their workstations and slow down their interactive tasks. On the other hand, some users want to dedicate some machines for critical batch jobs so that they have guaranteed resources. Even if all your workload is batch jobs, you still want to reduce resource contentions and operating system overhead to maximize the use of your resources.

- [Interactive batch job messaging](#)
- [Run X applications with bsub](#)

You can start an X session on the least loaded host by submitting it as a batch job:

- [Configuring SSH X11 forwarding for jobs](#)
- [Write job scripts](#)

You can build a job file one line at a time, or create it from another file, by running **bsub** without specifying a job to submit. When you do this, you start an interactive session in which **bsub** reads command lines from the standard input and submits them as a single batch job. You are prompted with **bsub>** for each line.

- [Register utmp file entries for interactive batch jobs](#)

LSF administrators can configure the cluster to track user and account information for interactive batch jobs submitted with **bsub -Ip** or **bsub -Is**. User and account information is registered as entries in the UNIX utmp file, which holds information for commands such as **who**. Registering user information for interactive batch jobs in utmp allows more accurate job accounting.

About interactive jobs

It is sometimes desirable from a system management point of view to control all workload through a single centralized scheduler.

Running an interactive job through the LSF batch system allows you to take advantage of batch scheduling policies and host selection features for resource-intensive jobs. You can submit a job and the least loaded host is selected to run the job.

Since all interactive batch jobs are subject to LSF policies, you will have more control over your system. For example, you may dedicate two servers as interactive servers, and disable interactive access to all other servers by defining an interactive queue that only uses the two interactive servers.

Scheduling policies

Running an interactive batch job allows you to take advantage of batch scheduling policies and host selection features for resource-intensive jobs.

An interactive batch job is scheduled using the same policy as all other jobs in a queue. This means an interactive job can wait for a long time before it gets dispatched. If fast response time is required, interactive jobs should be submitted to high-priority queues with loose scheduling constraints.

Interactive queues

You can configure a queue to be interactive-only, batch-only, or both interactive and batch with the parameter **INTERACTIVE** in **lsb.queues**.

Interactive jobs with non-batch utilities

Non-batch utilities such as **lsrun**, **lsgrun**, etc., use LIM simple placement advice for host selection when running interactive tasks.

Submit interactive jobs

Use the **bsub -I** option to submit batch interactive jobs, and the **bsub -Is** and **-Ip** options to submit batch interactive jobs in pseudo-terminals.

Pseudo-terminals are not supported for Windows.

For more details, see the `bsub` command.

Attention: For interactive jobs to work, the submission and execution host must be connected. That is, the `nios` daemon on the submission host must have a TCP connection with the `res` daemon on the execution host.

Find out which queues accept interactive jobs

Before you submit an interactive job, you need to find out which queues accept interactive jobs with the **bqueues -l** command.

If the output of this command contains the following, this is a batch-only queue. This queue does not accept interactive jobs:

```
SCHEDULING POLICIES: NO_INTERACTIVE
```

If the output contains the following, this is an interactive-only queue:

```
SCHEDULING POLICIES: ONLY_INTERACTIVE
```

If none of the above are defined or if `SCHEDULING POLICIES` is not in the output of **bqueues -l**, both interactive and batch jobs are accepted by the queue.

You configure interactive queues in the `lsb.queues` file.

- [Submitting an interactive job](#)
- [Submitting an interactive job by using a pseudo-terminal](#)
- [Submitting an interactive job and redirect streams to files](#)
- [Submitting an interactive job, redirect streams to files, and display streams](#)

Submitting an interactive job

Procedure

Use the **bsub -I** option to submit an interactive batch job.

For example:

```
bsub -I ls
```

Submits a batch interactive job which displays the output of **ls** at the user's terminal.

```
% bsub -I -q interactive -n 4,10 lsmake
```

```
<<Waiting for dispatch ...>>
```

This example starts Make on 4 to 10 processors and displays the output on the terminal.

A new job cannot be submitted until the interactive job is completed or terminated.

When an interactive job is submitted, a message is displayed while the job is awaiting scheduling. The **bsub** command stops display of output from the shell until the job completes, and no mail is sent to the user by default. A user can issue a `ctrl-c` at any time to terminate the job.

Interactive jobs cannot be check-pointed.

Interactive batch jobs cannot be re-runnable (**bsub -r**)

You can submit interactive batch jobs to re-runnable queues (`RERUNNABLE=y` in `lsb.queues`) or re-runnable application profiles (`RERUNNABLE=y` in `lsb.applications`).

Submitting an interactive job by using a pseudo-terminal

About this task

Submission of interaction jobs using pseudo-terminal is not supported for Windows for either **lsrun** or **bsub** LSF commands.

Some applications such as `vi` require a pseudo-terminal in order to run correctly.

You can also submit an interactive job using a pseudo-terminal with shell mode support. This option should be specified for submitting interactive shells, or applications which redefine the CTRL-C and CTRL-Z keys (for example, `jove`).

Procedure

1. Submit a batch interactive job using a pseudo-terminal.

```
bsub -Ip vi myfile
```

Submits a batch interactive job to edit `myfile`.

When you specify the **-Ip** option, **bsub** submits a batch interactive job and creates a pseudo-terminal when the job starts.

2. Submit a batch interactive job and create a pseudo-terminal with shell mode support.

```
bsub -Is csh
```

Submits a batch interactive job that starts up `csh` as an interactive shell.

When you specify the **-Is** option, **bsub** submits a batch interactive job and creates a pseudo-terminal with shell mode support when the job starts.

Submitting an interactive job and redirect streams to files

bsub -i, -o, -e

About this task

You can use the `-I` option together with the `-i`, `-o`, and `-e` options of **bsub** to selectively redirect streams to files. For more details, see the `bsub(1)` man page.

Procedure

To save the standard error stream in the `job.err` file, while standard input and standard output come from the terminal:

```
% bsub -I -q interactive -e job.err lsmake
```

Splitting stdout and stderr

About this task

If in your environment there is a wrapper around **bsub** and LSF commands so that end-users are unaware of LSF and LSF-specific options, you can redirect standard output and standard error of batch interactive jobs to a file with the `>` operator.

By default, both standard error messages and output messages for batch interactive jobs are written to stdout on the submission host.

Procedure

1. To write both `stderr` and `stdout` to `mystdout`:

```
bsub -I myjob 2>mystderr 1>mystdout
```

2. To redirect both `stdout` and `stderr` to different files, set `LSF_INTERACTIVE_STDERR=y` in `lsf.conf` or as an environment variable.

For example, with `LSF_INTERACTIVE_STDERR` set:

```
bsub -I myjob 2>mystderr 1>mystdout
```

`stderr` is redirected to `mystderr`, and `stdout` to `mystdout`.

Submitting an interactive job, redirect streams to files, and display streams

About this task

When using any of the interactive **bsub** options (for example: `-I`, `-Is`, `-ISs`) as well as the `-o` or `-e` options, you can also have your output displayed on the console by using the `-tty` option.

Procedure

To run an interactive job, redirect the error stream to file, and display the stream to the console:

```
% bsub -I -q interactive -e job.err -tty lsmake
```

Performance tuning for interactive batch jobs

LSF is often used on systems that support both interactive and batch users. On one hand, users are often concerned that load sharing will overload their workstations and slow down their interactive tasks. On the other hand, some users want to dedicate some machines for critical batch jobs so that they have guaranteed resources. Even if all your workload is batch jobs, you still want to reduce resource contentions and operating system overhead to maximize the use of your resources.

Numerous parameters can be used to control your resource allocation and to avoid undesirable contention.

Types of load conditions

Since interferences are often reflected from the load indices, LSF responds to load changes to avoid or reduce contentions. LSF can take actions on jobs to reduce interference before or after jobs are started. These actions are triggered by different load conditions. Most of the conditions can be configured at both the queue level and at the host level. Conditions defined at the queue level apply to all hosts used by the queue, while conditions defined at the host level apply to all queues using the host.

Scheduling conditions

These conditions, if met, trigger the start of more jobs. The scheduling conditions are defined in terms of load thresholds or resource requirements.

At the queue level, scheduling conditions are configured as either resource requirements or scheduling load thresholds, as described in `lsb.queues`. At the host level, the scheduling conditions are defined as scheduling load thresholds, as described in `lsb.hosts`.

Suspending conditions

These conditions affect running jobs. When these conditions are met, a `SUSPEND` action is performed to a running job.

At the queue level, suspending conditions are defined as `STOP_COND` as described in `lsb.queues` or as suspending load threshold. At the host level, suspending conditions are defined as stop load threshold as described in `lsb.hosts`.

Resuming conditions

These conditions determine when a suspended job can be resumed. When these conditions are met, a RESUME action is performed on a suspended job.

At the queue level, resume conditions are defined as by RESUME_COND in `lsb.queues`, or by the `loadSched` thresholds for the queue if RESUME_COND is not defined.

Types of load indices

To effectively reduce interference between jobs, correct load indices should be used properly. Below are examples of a few frequently used parameters.

Paging rate (pg)

The paging rate (pg) load index relates strongly to the perceived interactive performance. If a host is paging applications to disk, the user interface feels very slow.

The paging rate is also a reflection of a shortage of physical memory. When an application is being paged in and out frequently, the system is spending a lot of time performing overhead, resulting in reduced performance.

The paging rate load index can be used as a threshold to either stop sending more jobs to the host, or to suspend an already running batch job to give priority to interactive users.

This parameter can be used in different configuration files to achieve different purposes. By defining paging rate threshold in `lsf.cluster.cluster_name`, the host will become busy from LIM's point of view; therefore, no more jobs will be advised by LIM to run on this host.

By including paging rate in queue or host scheduling conditions, jobs can be prevented from starting on machines with a heavy paging rate, or can be suspended or even killed if they are interfering with the interactive user on the console.

A job suspended due to pg threshold will not be resumed even if the resume conditions are met unless the machine is interactively idle for more than PG_SUSP_IT seconds.

Interactive idle time (it)

Strict control can be achieved using the idle time (it) index. This index measures the number of minutes since any interactive terminal activity. Interactive terminals include hard wired ttys and **rlogin** sessions, and X shell windows such as **xterm**. On some hosts, LIM also detects mouse and keyboard activity.

This index is typically used to prevent batch jobs from interfering with interactive activities. By defining the suspending condition in the queue as `it<1 && pg>50`, a job from this queue will be suspended if the machine is not interactively idle and the paging rate is higher than 50 pages per second. Furthermore, by defining the resuming condition as `it>5 && pg<10` in the queue, a suspended job from the queue will not resume unless it has been idle for at least five minutes and the paging rate is less than ten pages per second.

The it index is only non-zero if no interactive users are active. Setting the it threshold to five minutes allows a reasonable amount of think time for interactive users, while making the machine available for load sharing, if the users are logged in but absent.

For lower priority batch queues, it is appropriate to set an it suspending threshold of two minutes and scheduling threshold of ten minutes in the `lsb.queues` file. Jobs in these queues are suspended while the execution host is in use, and resume after the host has been idle for a longer period. For hosts where all batch jobs, no matter how important, should be suspended, set a per-host suspending threshold in the `lsb.hosts` file.

CPU run queue length (r15s, r1m, r15m)

Running more than one CPU-bound process on a machine (or more than one process per CPU for multiprocessors) can reduce the total throughput because of operating system overhead, as well as interfering with interactive users. Some tasks such as compiling can create more than one CPU-intensive task.

Queues should normally set CPU run queue scheduling thresholds below 1.0, so that hosts already running compute-bound jobs are left alone. LSF scales the run queue thresholds for multiprocessor hosts by using the effective run queue lengths, so

multiprocessors automatically run one job per processor in this case.

For short to medium-length jobs, the `r1m` index should be used. For longer jobs, you might want to add an `r15m` threshold. An exception to this are high priority queues, where turnaround time is more important than total throughput. For high priority queues, an `r1m` scheduling threshold of 2.0 is appropriate.

CPU utilization (ut)

The `ut` parameter measures the amount of CPU time being used. When all the CPU time on a host is in use, there is little to gain from sending another job to that host unless the host is much more powerful than others on the network. A `ut` threshold of 90% prevents jobs from going to a host where the CPU does not have spare processing cycles.

If a host has very high `pg` but low `ut`, then it may be desirable to suspend some jobs to reduce the contention.

Some commands report `ut` percentage as a number from 0-100, some report it as a decimal number between 0-1. The configuration parameter in the `lsf.cluster.cluster_name` file, the configuration files, and the **bsub -R** resource requirement string take a fraction in the range from 0 to 1.

The command **bhist** shows the execution history of batch jobs, including the time spent waiting in queues or suspended because of system load.

The command **bjobs -p** shows why a job is pending.

Scheduling conditions and resource thresholds

Three parameters, `RES_REQ`, `STOP_COND` and `RESUME_COND`, can be specified in the definition of a queue. Scheduling conditions are a more general way for specifying job dispatching conditions at the queue level. These parameters take resource requirement strings as values which allows you to specify conditions in a more flexible manner than using the `loadSched` or `loadStop` thresholds.

Interactive batch job messaging

LSF can display messages to `stderr` or the Windows console when the following changes occur with interactive batch jobs:

- Job state
- Pending reason
- Suspend reason

Other job status changes, like switching the job's queue, are not displayed.

Limitations

Interactive batch job messaging is not supported in a multicluster environment.

Windows

Interactive batch job messaging is not fully supported on Windows. Only changes in the job state that occur before the job starts running are displayed. No messages are displayed after the job starts.

- [Configuring interactive batch job messaging](#)
- [Example messages](#)

Configuring interactive batch job messaging

About this task

Messaging for interactive batch jobs can be specified cluster-wide or in the user environment.

Procedure

1. Enable interactive batch job messaging for all users in the cluster.

In `lsf.conf`:

- `LSB_INTERACT_MSG_ENH=Y`
- (Optional) `LSB_INTERACT_MSG_INTVAL`

`LSB_INTERACT_MSG_INTVAL` specifies the time interval, in seconds, in which LSF updates messages about any changes to the pending status of the job. The default interval is 60 seconds. `LSB_INTERACT_MSG_INTVAL` is ignored if `LSB_INTERACT_MSG_ENH` is not set.

2. Enable messaging for interactive batch jobs.

Define `LSB_INTERACT_MSG_ENH` and `LSB_INTERACT_MSG_INTVAL` as environment variables.

The user-level definition of `LSB_INTERACT_MSG_ENH` overrides the definition in `lsf.conf`.

Example messages

Job in pending state

The following example shows messages displayed when a job is in pending state:

```
bsub -Is -R "ls < 2" csh
Job <2812> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>

<< Job's resource requirements not satisfied: 2 hosts; >>
<< Load information unavailable: 1 host; >>

<< Just started a job recently: 1 host; >>
<< Load information unavailable: 1 host; >>
<< Job's resource requirements not satisfied: 1 host; >>
```

Job terminated by user

The following example shows messages displayed when a job in pending state is terminated by the user:

```
bsub -m hostA -b 13:00 -Is sh
Job <2015> is submitted to default queue <normal>.
Job will be scheduled after Fri Nov 19 13:00:00 2009
<<Waiting for dispatch ...>>

<< New job is waiting for scheduling >>
<< The job has a specified start time >>

bkill 2015
<< Job <2015> has been terminated by user or administrator >>

<<Terminated while pending>>
```

Job suspended then resumed

The following example shows messages displayed when a job is dispatched, suspended, and then resumed:

```
bsub -m hostA -Is sh
Job <2020> is submitted to default queue <normal>.
<<Waiting for dispatch ...>>

<< New job is waiting for scheduling >>
<<Starting on hostA>>
```

```
bstop 2020
<< The job was suspended by user >>

bresume 2020
<< Waiting for re-scheduling after being resumed by user >>
```

Run X applications with bsub

You can start an X session on the least loaded host by submitting it as a batch job:

```
bsub xterm
```

An `xterm` is started on the least loaded host in the cluster.

When you run X applications using **lsrun** or **bsub**, the environment variable `DISPLAY` is handled properly for you. It behaves as if you were running the X application on the local machine.

Configuring SSH X11 forwarding for jobs

Before you begin

X11 forwarding must already be working outside LSF.

Procedure

1. Install SSH and enable X11 forwarding for all hosts that will submit and run these jobs (UNIX hosts only).
2. (Optional) In `lsf.conf`, specify an SSH command for `LSB_SSH_XFORWARD_CMD`.
The command can include full `PATH` and options.

Write job scripts

You can build a job file one line at a time, or create it from another file, by running **bsub** without specifying a job to submit. When you do this, you start an interactive session in which **bsub** reads command lines from the standard input and submits them as a single batch job. You are prompted with **bsub>** for each line.

You can use the **bsub -Zs** command to spool a file.

For more details on **bsub** options, see the **bsub(1)** man page.

Writing a job file one line at a time

UNIX example:

```
% bsub -q simulation
bsub> cd /work/data/myhomedir bsub> myjob arg1 arg2 .....
bsub> rm myjob.log
bsub> ^D
Job <1234> submitted to queue <simulation>.
```

In the previous example, the 3 command lines run as a Bourne shell (`/bin/sh`) script. Only valid Bourne shell command lines are acceptable in this case.

Windows example:

```
C:\> bsub -q simulation
bsub> cd \\server\data\myhomedir
bsub> myjob arg1 arg2 .....
bsub> del myjob.log
bsub> ^Z
Job <1234> submitted to queue <simulation>.
```

In the previous example, the 3 command lines run as a batch (.bat) file. Note that only valid Windows batch file command lines are acceptable in this case.

Specifying embedded submission options

You can specify job submission options in scripts read from standard input by the **bsub** command using lines starting with **#BSUB**:

```
% bsub -q simulation bsub> #BSUB -q test
bsub> #BSUB -o outfile -R "mem>10"
bsub> myjob arg1 arg2
bsub> #BSUB -J simjob
bsub> ^D
Job <1234> submitted to queue <simulation>.
```

Note:

- Command-line options override embedded options. In this example, the job is submitted to the `simulation` queue rather than the `test` queue.
- Submission options can be specified anywhere in the standard input. In the above example, the `-J` option of **bsub** is specified after the command to be run.
- More than one option can be specified on one line, as shown in the previous example.

Specifying job options in a file

In this example, options to run the job are specified in the `options_file`.

```
% bsub -q simulation < options_file
Job <1234> submitted to queue <simulation>.
```

On UNIX, the `options_file` must be a text file that contains Bourne shell command lines. It cannot be a binary executable file.

On Windows, the `options_file` must be a text file containing Windows batch file command lines.

Spooling a job command file

Use **bsub -Zs** to spool a job command file to the directory specified by the `JOB_SPOOL_DIR` parameter in `lsb.params`, and use the spooled file as the command file for the job.

Use the **bmod -Zsn** command to modify or remove the command file after the job has been submitted. Removing or modifying the original input file does not affect the submitted job.

Redirecting a script to bsub standard input

You can redirect a script to the standard input of the **bsub** command:

```
% bsub < myscript
Job <1234> submitted to queue <test>.
```

In this example, the `myscript` file contains job submission options as well as command lines to execute. When the **bsub** command reads a script from its standard input, it can be modified right after **bsub** returns for the next job submission.

When the script is specified on the **bsub** command line, the script is not spooled:

```
% bsub myscript
Job <1234> submitted to default queue <normal>.
```

In this case the command line `myscript` is spooled, instead of the contents of the `myscript` file. Later modifications to the `myscript` file can affect job behavior.

Loading and running a job script file

If the `LSB_BSUB_PARSE_SCRIPT` parameter is set to `Y` in the `lsf.conf` file, you can use the **bsub** command to load, parse, and run job script files directly from the command line. Submit a job with the job script as a command. The job script must be an ASCII text file and not a binary file.

In this example, the `myscript` file contains job submission options as well as command lines to execute. Use the `#BSUB` imperative at the beginning of each line to specify embedded job submission options in the script.

When the script is specified in the **bsub** command line, the **bsub** command loads and parses the job script, then runs the script as the job itself:

```
% bsub myscript
Job <1234> submitted to default queue <normal>.
```

Running a job under a particular shell

By default, LSF runs batch jobs using the Bourne (`/bin/sh`) shell. You can specify the shell under which a job is to run. This is done by specifying an interpreter in the first line of the script.

For example:

```
% bsub
bsub> #!/bin/csh -f
bsub> set coredump='ls |grep core'
bsub> if ( "$coredump" != "" ) then
bsub> mv core.`date | cut -d" " -f1`
bsub> endif
bsub> myjob
bsub> ^D
Job <1234> is submitted to default queue <normal>.
```

The **bsub** command must read the job script from standard input to set the execution shell. If you do not specify a shell in the script, the script is run using `/bin/sh`. If the first line of the script starts with a `#` not immediately followed by an exclamation mark (`!`), then `/bin/csh` is used to run the job.

For example:

```
% bsub
bsub> # This is a comment line. This tells the system to use /bin/csh to
bsub> # interpret the script.
bsub>
bsub> setenv DAY `date | cut -d" " -f1`
bsub> myjob bsub> ^D
Job <1234> is submitted to default queue <normal>.
```

If running jobs under a particular shell is required frequently, you can specify an alternate shell using a command-level job starter and run your jobs interactively.

Register utmp file entries for interactive batch jobs

LSF administrators can configure the cluster to track user and account information for interactive batch jobs submitted with **bsub -lp** or **bsub -ls**. User and account information is registered as entries in the UNIX `utmp` file, which holds information for commands such as **who**. Registering user information for interactive batch jobs in `utmp` allows more accurate job accounting.

Configuration and operation

To enable `utmp` file registration, the LSF administrator sets the `LSB_UTMP` parameter in `lsf.conf`.

When LSB_UTMP is defined, LSF registers the job by adding an entry to the utmp file on the execution host when the job starts. After the job finishes, LSF removes the entry for the job from the utmp file.

Limitations

- Registration of utmp file entries is supported on the following platforms:
 - Solaris (all versions)
 - HP-UX (all versions)
 - Linux (all versions)
- utmp file registration is not supported in a multicluster environment.
- Because interactive batch jobs submitted with **bsub -I** are not associated with a pseudo-terminal, utmp file registration is not supported for these jobs.

Interactive and remote tasks

You can run tasks interactively and remotely with non-batch utilities such as **lrun** and **lsgun**.

- [Run remote tasks](#)
lrun is a non-batch utility to run tasks on a remote host. **lsgun** is a non-batch utility to run the same task on many hosts, in sequence one after the other, or in parallel.
- [Interactive tasks](#)
LSF supports transparent execution of tasks on all server hosts in the cluster. You can run your program on the best available host and interact with it just as if it were running directly on your workstation. Keyboard signals such as **CTRL-Z** and **CTRL-C** work as expected.
- [Load sharing interactive sessions](#)
There are different ways to use LSF to start an interactive session on the best available host.

Run remote tasks

lrun is a non-batch utility to run tasks on a remote host. **lsgun** is a non-batch utility to run the same task on many hosts, in sequence one after the other, or in parallel.

The default for **lrun** is to run the job on the host with the least CPU load (represented by the lowest normalized CPU run queue length) and the most available memory. Command-line arguments can be used to select other resource requirements or to specify the execution host.

To avoid typing in the **lrun** command every time you want to execute a remote job, you can also use a shell alias or script to run your job.

For a complete description of **lrun** and **lsgun** options, see the `lrun(1)` and `lsgun(1)` man pages.

- [Running a task on the best available host](#)
- [Running a task on a host with specific resources](#)
- [Running a task on a specific host](#)
- [Running a task by using a pseudo-terminal](#)
- [Running the same task on many hosts in sequence](#)
- [Running parallel tasks](#)
- [Running tasks on hosts specified by a file](#)

Running a task on the best available host

Procedure

Submit your task using **lsrun**.
`lsrun mytask`

LSF automatically selects a host of the same type as the local host, if one is available. By default the host with the lowest CPU and memory load is selected.

Running a task on a host with specific resources

About this task

If you want to run **mytask** on a host that meets specific resource requirements, you can specify the resource requirements using the `-R res_req` option of **lsrun**.

Procedure

```
lsrun -R 'cserver && swp>100' mytask
```

In this example **mytask** must be run on a host that has the resource **cserver** and at least 100 MB of virtual memory available.

- **Resource usage**
Resource reservation is only available for batch jobs. If you run jobs using only LSF Base, LIM uses resource usage to determine the placement of jobs. Resource usage requests are used to temporarily increase the load so that a host is not overloaded. When LIM makes a placement advice, external load indices are not considered in the resource usage string.

Resource usage

Resource reservation is only available for batch jobs. If you run jobs using only LSF Base, LIM uses resource usage to determine the placement of jobs. Resource usage requests are used to temporarily increase the load so that a host is not overloaded. When LIM makes a placement advice, external load indices are not considered in the resource usage string.

In this case, the syntax of the resource usage string is

```
res[=value]:res[=value]:...:res[=value]
```

The `res` is one of the resources whose value is returned by the **lsload** command.

```
rusage[r1m=0.5:mem=20:swp=40]
```

The above example indicates that the task is expected to increase the 1-minute run queue length by 0.5, consume 20 MB of memory and 40 MB of swap space.

If no value is specified, the task is assumed to be intensive in using that resource. In this case no more than one task will be assigned to a host regardless of how many CPUs it has.

The default resource usage for a task is `r15s=1.0:r1m=1.0:r15m=1.0`. This indicates a CPU-intensive task which consumes few other resources.

Running a task on a specific host

Procedure

If you want to run your task on a particular host, use the **lsrun -m** option:

```
lsrun -m hostD mytask
```

Running a task by using a pseudo-terminal

About this task

Submission of interaction jobs using pseudo-terminal is not supported for Windows for either **lsrun** or **bsub** LSF commands.

Some tasks, such as text editors, require special terminal handling. These tasks must be run using a pseudo-terminal so that special terminal handling can be used over the network.

Procedure

The **-P** option of **lsrun** specifies that the job should be run using a pseudo-terminal:

```
lsrun -P vi
```

Running the same task on many hosts in sequence

About this task

The **lsgrun** command allows you to run the same task on many hosts, one after the other, or in parallel.

Procedure

For example, to merge the `/tmp/out` file on hosts `hostA`, `hostD`, and `hostB` into a single file named `gout`, enter:

```
lsgrun -m "hostA hostD hostB" cat /tmp/out >> gout
```

Running parallel tasks

About this task

The **-p** option tells **lsgrun** that the task specified should be run in parallel. See `lsgrun(1)` for more details.

Procedure

To remove the `/tmp/core` file from all 3 hosts, enter:

```
lsgrun -m "hostA hostD hostB" -p rm -r /tmp/core
```

Running tasks on hosts specified by a file

Procedure

The **lsgrun -f *host_file*** option reads the *host_file* file to get a list of hosts on which to run the task.

Interactive tasks

LSF supports transparent execution of tasks on all server hosts in the cluster. You can run your program on the best available host and interact with it just as if it were running directly on your workstation. Keyboard signals such as **CTRL-Z** and **CTRL-C** work as expected.

Interactive tasks communicate with the user in real time. Programs like `vi` use a text-based terminal interface. Computer Aided Design and desktop publishing applications usually use a graphic user interface (GUI).

This section outlines issues for running interactive tasks with the non-batch utilities `lshrun`, **lsgrun**, etc. To run interactive tasks with these utilities, use the `-i` option.

For more details, see the `lshrun(1)` and `lsgrun(1)` man pages.

- [Redirecting streams to files](#)

Interactive tasks on remote hosts

Job controls

When you run an interactive task on a remote host, you can perform most of the job controls as if it were running locally. If your shell supports job control, you can suspend and resume the task and bring the task to background or foreground as if it were a local task.

For a complete description, see the `lshrun(1)` man page.

Hide remote execution

You can also write one-line shell scripts or **csh** aliases to hide remote execution. For example:

```
#!/bin/sh
#Script to remotely execute mytask exec
lshrun -m hostD mytask
```

or

```
alias mytask "lshrun -m hostD mytask"
```

Interactive processing and scheduling policies

LSF lets you run interactive tasks on any computer on the network, using your own terminal or workstation. Interactive tasks run immediately and normally require some input through a text-based or graphical user interface. All the input and output is transparently sent between the local host and the job execution host.

Shared files and user IDs

When LSF runs a task on a remote host, the task uses standard UNIX system calls to access files and devices. The user must have an account on the remote host. All operations on the remote host are done with the user's access permissions.

Tasks that read and write files access the files on the remote host. For load sharing to be transparent, your files should be available on all hosts in the cluster using a file sharing mechanism such as NFS or AFS. When your files are available on all hosts in the cluster, you can run your tasks on any host without worrying about how your task will access files.

LSF can operate correctly in cases where these conditions are not met, but the results may not be what you expect. For example, the `/tmp` directory is usually private on each host. If you copy a file into `/tmp` on a remote host, you can only read that file on the same remote host.

LSF can also be used when files are not available on all hosts. LSF provides the **lsrnp** command to copy files across LSF hosts. You can use pipes to redirect the standard input and output of remote commands, or write scripts to copy the data files to the execution host.

Shell mode for remote execution

On UNIX, shell mode support is provided for running interactive applications through RES.

Not supported for Windows.

Shell mode support is required for running interactive shells or applications that redefine the **CTRL-C** and **CTRL-Z** keys (for example, **jove**).

The **-S** option of **lsrun**, **ch** or **lsgrun** creates the remote task with shell mode support. The default is not to enable shell mode support.

Run windows

Some run windows are only applicable to batch jobs. Interactive jobs scheduled by LIM are controlled by another set of run windows.

Redirecting streams to files

About this task

By default, both standard error messages and standard output messages of interactive tasks are written to stdout on the submission host.

To separate stdout and stderr and redirect to separate files, set **LSF_INTERACTIVE_STDERR=y** in **lsf.conf** or as an environment variable.

Procedure

To redirect both stdout and stderr to different files with the parameter set:

```
lsrun mytask 2>mystderr 1>mystdout
```

The result of the above example is for stderr to be redirected to **mystderr**, and stdout to **mystdout**. Without **LSF_INTERACTIVE_STDERR** set, both stderr and stdout will be redirected to **mystdout**.

Load sharing interactive sessions

There are different ways to use LSF to start an interactive session on the best available host.

Note: The **lslogin** command is deprecated and might be removed in a future version of LSF.

- [Logging on to the least loaded host](#)
- [Logging on to a host with specific resources](#)

Logging on to the least loaded host

About this task

Note: The **lslogin** command is deprecated and might be removed in a future version of LSF.

Procedure

To log on to the least loaded host, use the **lslogin** command.

When you use **lslogin**, LSF automatically chooses the best host and does an **rlogin** to that host.

With no argument, **lslogin** picks a host that is lightly loaded in CPU, has few login sessions, and whose binary is compatible with the current host.

Logging on to a host with specific resources

About this task

Note: The **lslogin** command is deprecated and might be removed in a future version of LSF.

Procedure

If you want to log on a host that meets specific resource requirements, use the **lslogin -R res_req** option.

```
lslogin -R "solaris order[ls:cpu]"
```

This command opens a remote login to a host that has the **sunos** resource, few other users logged in, and a low CPU load level. This is equivalent to using **lsplace** to find the best host and then using **rlogin** to log in to that host:

```
rlogin `lsplace -R "sunos order[ls:cpu]"`
```

Configuring and sharing IBM Spectrum LSF job resources

Learn how to configure and allocate resources to your LSF jobs. Share compute resources fairly among users and projects. Apply resource allocation limits to your jobs, manage host and user groups, reserve resources, and specify resource requirements for jobs.

- [About LSF resources](#)
The LSF system uses built-in and configured resources to track job resource requirements and schedule jobs according to the resources available on individual hosts.
- [Representing job resources in LSF](#)
Learn how job resources are represented in LSF.
- [Plan-based scheduling and reservations](#)
Plan-based scheduling greatly improves the original scheduling and reservation features in LSF. Instead of looking only at current resource availability, the scheduler can plan job placements for the near future. Reservations are then based on these planned allocation. Plan-based scheduling is meant to be a replacement for legacy LSF reservation policies. When ALLOCATION_PLANNER is enabled, parameters related to the old reservation features are ignored.
- [Distributing job resources to users in LSF](#)
Learn how users can share job resources through LSF.
- [Global resources](#)
Global resources are resources that are shared between all connected clusters.

About LSF resources

The LSF system uses built-in and configured resources to track job resource requirements and schedule jobs according to the resources available on individual hosts.

- [Resource categories](#)
- [How LSF uses resources](#)
Jobs that are submitted through LSF have resource usage that is monitored while they are running. This information is used to enforce resource usage limits and load thresholds as well as for fair share scheduling.

Resource categories

By values

Boolean resources	Resources that denote the availability of specific features
Numerical resources	Resources that take numerical values, such as all the load indices, number of processors on a host, or host CPU factor
String resources	Resources that take string values, such as host type, host model, host status

By the way values change

Dynamic Resources	Resources that change their values dynamically: host status and all the load indices.
Static Resources	Resources that do not change their values: all resources except for load indices or host status.

By definitions

External Resources	Custom resources defined by user sites: external load indices and resources defined in the lsf.shared file (shared resources).
Built-In Resources	Resources that are always defined in LSF, such as load indices, number of CPUs, or total swap space.

By scope

Host-Based Resources	Resources that are not associated with individual hosts in the same way, but are owned by the entire cluster, or a subset of hosts within the cluster, such as shared file systems. An application can access such a resource from any host which is configured to share it, but doing so affects its value as seen by other hosts.
Shared Resources	

Boolean resources

Boolean resources (for example, server to denote LSF server hosts) have a value of one if they are defined for a host, and zero if they are not defined for the host. Use Boolean resources to configure host attributes to be used in selecting hosts to run jobs. For example:

- Machines may have different types and versions of operating systems.
- Machines may play different roles in the system, such as file server or compute server.
- Some machines may have special-purpose devices that are needed by some applications.
- Certain software packages may be available only on some of the machines.

Specify a Boolean resource in a resource requirement selection string of a job to select only hosts that can run the job.

Some examples of Boolean resources:

Resource Name	Describes	Meaning of Example Name
cs	Role in cluster	Compute server
fs	Role in cluster	File server

Resource Name	Describes	Meaning of Example Name
solaris	Operating system	Solaris operating system
frame	Available software	FrameMaker license

Shared resources

Shared resources are configured resources that are not tied to a specific host, but are associated with the entire cluster, or a specific subset of hosts within the cluster. For example:

- Disk space on a file server which is mounted by several machines
- The physical network connecting the hosts

LSF does not contain any built-in shared resources. All shared resources must be configured by the LSF administrator. A shared resource may be configured to be dynamic or static. In the preceding example, the total space on the shared disk may be static while the amount of space currently free is dynamic. A site may also configure the shared resource to report numeric, string, or Boolean values.

An application may use a shared resource by running on any host from which that resource is accessible. For example, in a cluster in which each host has a local disk but can also access a disk on a file server, the disk on the file server is a shared resource, and the local disk is a host-based resource. In contrast to host-based resources such as memory or swap space, using a shared resource from one machine affects the availability of that resource as seen by other machines. There is one value for the entire cluster which measures the utilization of the shared resource, but each host-based resource is measured separately.

The following restrictions apply to the use of shared resources in LSF products.

- A shared resource cannot be used as a load threshold in the Hosts section of the `lsf.cluster.cluster_name` file.
- A shared resource cannot be used in the `loadSched/loadStop` thresholds, or in the `STOP_COND` or `RESUME_COND` parameters in the queue definition in the `lsb.queues` file.

How LSF uses resources

Jobs that are submitted through LSF have resource usage that is monitored while they are running. This information is used to enforce resource usage limits and load thresholds as well as for fair share scheduling.

The following is the kind of information that LSF collects about resources:

- Total CPU time consumed by all processes in the job
- Total resident memory usage in KB of all currently running processes in a job
- Total virtual memory usage in KB of all currently running processes in a job
- Currently active process group ID in a job
- Currently active processes in a job

On UNIX and Linux, job-level resource usage is collected through a special process called PIM (Process Information Manager). PIM is managed internally by LSF.

See [Memory and swap limit enforcement based on Linux cgroup memory subsystem](#) for more information about memory usage and process tracking.

Representing job resources in LSF

Learn how job resources are represented in LSF.

- **[Batch built-in resources](#)**
The `slots` keyword lets you schedule jobs on the host with the fewest free slots first. This feature is useful for people who want to pack sequential jobs onto hosts with the least slots first, ensuring that more hosts will be available to run

parallel jobs. slots (unused slots) is supported in the `select[]` and `order[]` sections of the resource requirement string.

- **[Static resources](#)**

Static resources are built-in resources that represent host information that does not change over time, such as the maximum RAM available to user processes or the number of processors in a machine. Most static resources are determined by the LIM at start-up time, or when LSF detects hardware configuration changes.

- **[Load indices](#)**

Load indices are built-in resources that measure the availability of static or dynamic, non-shared resources on hosts in the LSF cluster.

- **[About configured resources](#)**

LSF schedules jobs that are based on available resources. There are many resources that are built into LSF, but you can also add your own resources, and then use them same way as built-in resources.

- **[Configure host resources](#)**

Add and remove hosts in your cluster. Configure LSF to run batch jobs on dynamic hosts. Configure a host to run one job at a time.

- **[Share resources in queues](#)**

Learn how to configure LSF queues. Use LSF commands to control queues (close, open, activate, inactivate). Configure dispatch and run windows in queues. Restrict which hosts can use queues. Restrict the job size requested by parallel jobs in a queue. Add queue administrators and give users access to queues. Control job order within queues and switch jobs from one queue to another. Configure an exclusive queue.

- **[Share resources with application profiles](#)**

Application profiles improve the management of applications by separating scheduling policies (for example, job preemption and fair share scheduling) from application-level requirements, such as pre-execution and post-execution commands, resource limits, or job controls.

Batch built-in resources

The slots keyword lets you schedule jobs on the host with the fewest free slots first. This feature is useful for people who want to pack sequential jobs onto hosts with the least slots first, ensuring that more hosts will be available to run parallel jobs. slots (unused slots) is supported in the `select[]` and `order[]` sections of the resource requirement string.

slots

slots is the number of unused slots on the host defined according to these values from bhosts for the host:

slots (Unused slots) = MAX – NJOBS

where NJOBS = RUN + SSUSP + USUSP + RSV

maxslots

maxslots is the maximum number of slots that can be used on a host according to the value from bhosts for the host.

maxslots (max slot) = MAX

where MAX is the value of the “MAX” column that is displayed by bhosts

maxslots is supported in the `select[]`, `order[]` and `same[]` sections of the resource requirement string.

You can specify slots in the order string. In the following example for reversed slots based ordering, hostA and hostB have 20 total slots each. There are currently no jobs in cluster. Then,

```
job1:bsub -n 10 sleep 10000 - runs on hostA
```

```
job2:bsub -n 1 sleep 10000 - might run on hostB
```

```
job3:bsub -n 20 sleep 10000 - will pend
```

If job2 runs on hostB, we can get a situation where job3, a large parallel job, never has a chance to run because neither host has 20 slots available. Essentially, job2 blocks job3 from running. However, with `order[-slots]`:

```
job1:bsub -n 10 -R "order[-slots]" sleep 10000 - runs on hostA
```

```
job2:bsub -n 1 -R "order[-slots]" sleep 10000 - will run on hostA
```

```
job3:bsub -n 20 -R "order[-slots]" sleep 10000 - will run on hostB
```

With reversed slots based ordering, job2 will run on hostA because hostA has the least available slots at this time (10 available versus 20 available for hostB). This allows job3 to run on hostB.

You can also specify maxslots in the order string. In the following example for reversed order on maxslots, hostA has 20 total slots, but hostB only has 10 slots in total, and currently no jobs in the cluster. Then,

```
job1:bsub -n 10 sleep 10000 - might run on hostA
```

```
job2:bsub -n 20 sleep 10000 - will pend
```

After job1 runs, both hostA and hostB have 10 available slots. Thus, job2 will pend (this is true with or without `order[-slots]`). However, with `order[-maxslots]`:

```
job1:bsub -n 10 -R "order[-maxslots]" sleep 10000 - will run on hostB
```

```
job2:bsub -n 20 -R "order[-maxslots]" sleep 10000 - will run on hostA
```

With reversed maxslots based order, job1 will run on hostB because it has fewer total slots than hostA. This saves hostA for the larger parallel job like job2.

You can have the combined effect of reverse ordering with slots and maxslots by using `order[-slots:maxslots]`.

Static resources

Static resources are built-in resources that represent host information that does not change over time, such as the maximum RAM available to user processes or the number of processors in a machine. Most static resources are determined by the LIM at start-up time, or when LSF detects hardware configuration changes.

Static resources can be used to select appropriate hosts for particular jobs based on binary architecture, relative CPU speed, and system configuration.

The resources `ncpus`, `nprocs`, `ncores`, `nthreads`, `maxmem`, `maxswp`, and `maxtmp` are not static on UNIX hosts that support dynamic hardware reconfiguration.

Static resources reported by LIM

Index	Measures	Units	Determined by
type	host type	string	configuration
model	host model	string	configuration
hname	host name	string	configuration
cpuf	CPU factor	relative	configuration
server	host can run remote jobs	Boolean	configuration
rexpri	execution priority	nice (2) argument	configuration
ncpus	number of processors	processors	LIM
ndisks	number of local disks	disks	LIM
nprocs	number of physical processors	processors	LIM
ncores	number of cores per physical processor	cores	LIM
nthreads	number of threads per processor core	threads	LIM
maxmem	maximum RAM	MB	LIM
maxswp	maximum swap space	MB	LIM
maxtmp	maximum space in /tmp	MB	LIM

Host type (type)

Host type is a combination of operating system and CPU architecture. All computers that run the same operating system on the same computer architecture are of the same type. You can add custom host types in the HostType section of lsf.shared. This alphanumeric value can be up to 39 characters long.

An example of host type is `LINUX86`.

Host model (model)

Host model is the combination of host type and CPU speed (CPU factor) of your machine. All hosts of the same relative type and speed are assigned the same host model. You can add custom host models in the HostModel section of lsf.shared. This alphanumeric value can be up to 39 characters long.

An example of host model is `Intel_IA64`.

Host name (hname)

Host name specifies the name with which the host identifies itself.

CPU factor (cpuf)

The CPU factor (frequently shortened to cpuf) represents the speed of the host CPU relative to other hosts in the cluster. For example, if one processor is twice the speed of another, its CPU factor should be twice as large. For multiprocessor hosts, the CPU factor is the speed of a single processor; LSF automatically scales the host CPU load to account for additional processors. The CPU factors are detected automatically or defined by the administrator.

Server

The `server` static resource is Boolean. It has the following values:

- 1 if the host is configured to run jobs from other hosts
- 0 if the host is an LSF client for submitting jobs to other hosts

Number of CPUs (ncpus)

By default, the number of CPUs represents the number of cores a machine has. As most CPUs consist of multiple cores, threads, and processors, ncpus can be defined by the cluster administrator (either globally or per-host) to consider one of the following:

- Processors
- Processors and cores
- Processors, cores, and threads

Globally, this definition is controlled by the parameter `EGO_DEFINE_NCPUS` in `lsf.conf` or `ego.conf`. The default behavior for ncpus is to consider the number of cores (`EGO_DEFINE_NCPUS=cores`).

Note:

1. On a machine running AIX, ncpus detection is different. Under AIX, the number of detected physical processors is always 1, whereas the number of detected cores is the number of cores across all physical processors. Thread detection is the same as other operating systems (the number of threads per core).
2. When `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the resource requirement string keyword ncpus refers to the number of slots instead of the number of CPUs, however **lshosts** output continues to show ncpus as defined by `EGO_DEFINE_NCPUS` in `lsf.conf`.

Number of disks (ndisks)

The number of disks specifies the number of local disks a machine has, determined by the LIM.

Maximum memory (maxmem)

Maximum memory is the total available memory of a machine, measured in megabytes (MB).

Maximum swap (maxswp)

Maximum swap is the total available swap space a machine has, measured in megabytes (MB).

Maximum temporary space (maxtmp)

Maximum temporary space is the total temporary space that a machine has, measured in megabytes (MB).

- [How LIM detects cores, threads, and processors](#)
Traditionally, the value of ncpus has been equal to the number of physical CPUs.
- [Defining ncpus: processors, cores, or threads](#)
- [Defining computation of ncpus on dynamic hosts](#)
- [Defining computation of ncpus on static hosts](#)

How LIM detects cores, threads, and processors

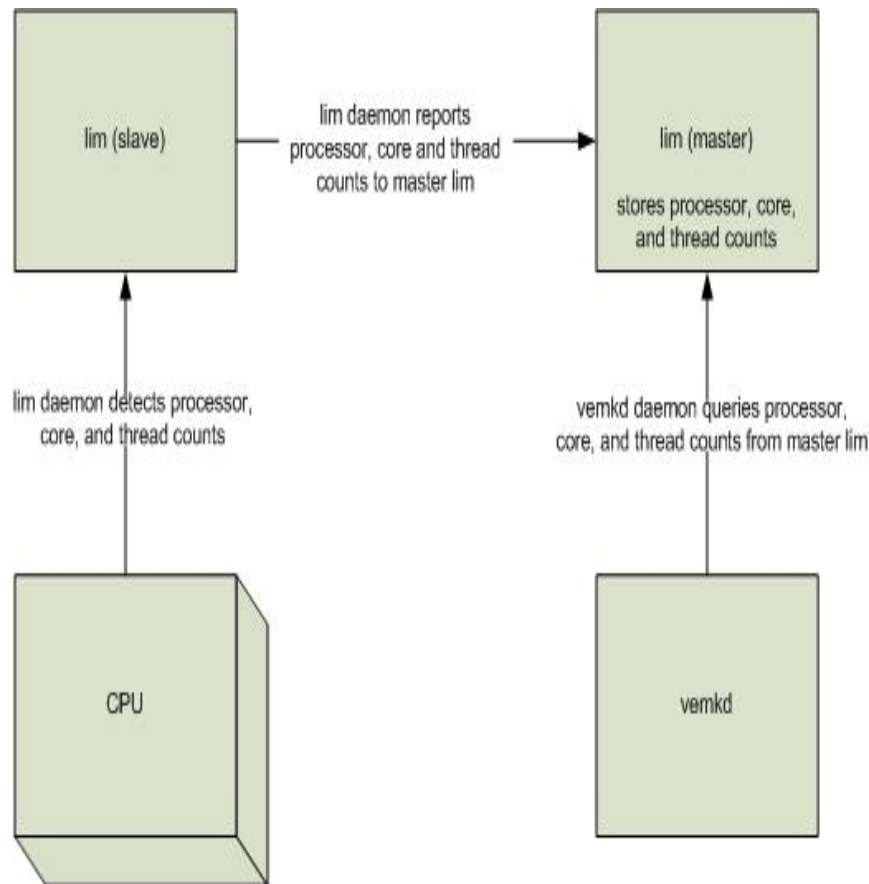
Traditionally, the value of ncpus has been equal to the number of physical CPUs.

However, many CPUs consist of multiple cores and threads, so the traditional 1:1 mapping is no longer useful. A more useful approach is to set ncpus to equal one of the following:

- The number of processors
- Cores: the number of cores (per processor) * the number of processors (this is the ncpus default setting)
- Threads: the number of threads (per core) * the number of cores (per processor) * the number of processors

A cluster administrator globally defines how ncpus is computed using the EGO_DEFINE_NCPUS parameter in lsf.conf or ego.conf (instead of LSF_ENABLE_DUALCORE in lsf.conf, or EGO_ENABLE_DUALCORE in ego.conf).

LIM detects and stores the number of processors, cores, and threads for all supported architectures. The following diagram illustrates the flow of information between daemons, CPUs, and other components.



Although the `ncpus` computation is applied globally, it can be overridden on a per-host basis.

To correctly detect processors, cores, and threads, LIM assumes that all physical processors on a single machine are of the same type.

In cases where CPU architectures and operating system combinations may not support accurate processor, core, thread detection, LIM uses the defaults of 1 processor, 1 core per physical processor, and 1 thread per core. If LIM detects that it is running in a virtual environment (for example, VMware®), each detected processor is similarly reported (as a single-core, single-threaded, physical processor).

LIM only detects hardware that is recognized by the operating system. LIM detection uses processor- or OS-specific techniques (for example, the Intel CPUID instruction, or Solaris `kstat()/core_id`). If the operating system does not recognize a CPU or core (for example, if an older OS does not recognize a quad-core processor and instead detects it as dual-core), then LIM does not recognize it either.

Note: RQL normalization never considers threads. Consider a hyper-thread enabled Pentium: Threads are not full-fledged CPUs, so considering them as CPUs would artificially lower the system load.

ncpus detection on AIX

On a machine running AIX, detection of `ncpus` is different. Under AIX, the number of detected physical processors is always 1, whereas the number of detected cores is always the number of cores across all physical processors. Thread detection is the same as other operating systems (the number of threads per core).

Defining ncpus: processors, cores, or threads

About this task

A cluster administrator must define how `ncpus` is computed. Usually, the number of available job slots is equal to the value of `ncpus`; however, slots can be redefined at the EGO resource group level. The `ncpus` definition is globally applied across the

cluster.

Procedure

1. Open `lsf.conf` or `ego.conf`.

- UNIX and Linux:
`LSF_CONFDIR/lsf.conf`

`LSF_CONFDIR/ego/cluster_name/kernel/ego.conf`

- Windows:
`LSF_CONFDIR\lsf.conf`

`LSF_CONFDIR\ego\cluster_name\kernel\ego.conf`

Important: You can set `EGO_DEFINE_NCPUS` in `ego.conf` only if EGO is enabled in the LSF cluster. If EGO is not enabled, you must set `EGO_DEFINE_NCPUS` in `lsf.conf`.

2. Define the parameter `EGO_DEFINE_NCPUS=[procs | cores | threads]`.

Set it to one of the following:

- `procs` (where `ncpus=procs`)
- `cores` (where `ncpus=procs * cores`)
- `threads` (where `ncpus=procs * cores * threads`)

By default, `ncpus` is set to `cores` (number of cores).

Note: In clusters with older LIMs that do not recognize cores and threads, `EGO_DEFINE_NCPUS` is ignored. In clusters where only the management host LIM recognizes cores and threads, the management host LIM assigns default values (for example, in LSF 6.2: 1 core, 1 thread).

3. Save and close `lsf.conf` or `ego.conf`.

Results

Tip: As a best practice, set `EGO_DEFINE_NCPUS` instead of `EGO_ENABLE_DUALCORE`. The functionality of `EGO_ENABLE_DUALCORE=y` is preserved by setting `EGO_DEFINE_NCPUS=cores`.

Interaction with `LSF_LOCAL_RESOURCES` in `lsf.conf`

If EGO is enabled, and `EGO_LOCAL_RESOURCES` is set in `ego.conf` and `LSF_LOCAL_RESOURCES` is set in `lsf.conf`, `EGO_LOCAL_RESOURCES` takes precedence.

Defining computation of `ncpus` on dynamic hosts

About this task

The `ncpus` global definition can be overridden on specified dynamic and static hosts in the cluster.

Procedure

1. Open `lsf.conf` or `ego.conf`.

- UNIX and Linux:
`LSF_CONFDIR/lsf.conf`

`LSF_CONFDIR/ego/cluster_name/kernel/ego.conf`

- Windows:
`LSF_CONFDIR\lsf.conf`

`LSF_CONFDIR\ego\cluster_name\kernel\ego.conf`

Important: You can set EGO_LOCAL_RESOURCES in ego.conf only if EGO is enabled in the LSF cluster. If EGO is not enabled, you must set EGO_LOCAL_RESOURCES in lsf.conf.

2. Define the parameter EGO_LOCAL_RESOURCES="[resource resource_name]".

Set resource_name to one of the following:

- define_ncpus_procs
- define_ncpus_cores
- define_ncpus_threads

Note: Resource definitions are mutually exclusive. Choose only one resource definition per host.

For example:

- Windows: EGO_LOCAL_RESOURCES="[type NTX86] [resource define_ncpus_procs]"
- Linux: EGO_LOCAL_RESOURCES="[resource define_ncpus_cores]"

3. Save and close ego.conf.

Results

Note: In multi-cluster environments, if ncpus is defined on a per-host basis (thereby overriding the global setting) the definition is applied to *all* clusters that the host is a part of. In contrast, globally defined ncpus settings *only* take effect within the cluster for which EGO_DEFINE_NCPUS is defined.

Defining computation of ncpus on static hosts

About this task

The ncpus global definition can be overridden on specified dynamic and static hosts in the cluster.

Procedure

1. Open lsf.cluster.cluster_name.
 - Linux: LSF_CONFDIR/lsf.cluster.cluster_name
 - Windows: LSF_CONFDIR\lsf.cluster.cluster_name
2. Find the host you for which you want to define ncpus computation. In the RESOURCES column, add one of the following definitions:
 - define_ncpus_procs
 - define_ncpus_cores
 - define_ncpus_threads

Note: Resource definitions are mutually exclusive. Choose only one resource definition per host.

For example:

```
Begin Host
HOSTNAME  model    type      rlm  mem  swp  RESOURCES  #Keywords
#lemon    PC200    LINUX86   3.5  1    2    (linux)
#plum     !        NTX86     3.5  1    2    (nt)
Host_name !        NTX86     -    -    -    (define_ncpus_procs)
End       Host
```

3. Save and close lsf.cluster.cluster_name.
4. Restart the management host.

Results

Note: In multi-cluster environments, if ncpus is defined on a per-host basis (thereby overriding the global setting) the definition is applied to *all* clusters that the host is a part of. In contrast, globally defined ncpus settings *only* take effect within the cluster for which EGO_DEFINE_NCPUS is defined.

Load indices

Load indices are built-in resources that measure the availability of static or dynamic, non-shared resources on hosts in the LSF cluster.

Load indices that are built into the LIM are updated at fixed time intervals.

External load indices are defined and configured by the LSF administrator, who writes an external load information manager (**elim**) executable. The **elim** collects the values of the external load indices and sends these values to the LIM.

Load indices collected by LIM

Index	Measures	Units	Direction	Averaged over	Update Interval
status	host status	string			15 seconds
r15s	run queue length	processes	increasing	15 seconds	15 seconds
r1m	run queue length	processes	increasing	1 minute	15 seconds
r15m	run queue length	processes	increasing	15 minutes	15 seconds
ut	CPU utilization	percent	increasing	1 minute	15 seconds
pg	paging activity	pages in + pages out per second	increasing	1 minute	15 seconds
ls	logins	users	increasing	N/A	30 seconds
it	idle time	minutes	decreasing	N/A	30 seconds
swp	available swap space	MB	decreasing	N/A	15 seconds
mem	available memory	MB	decreasing	N/A	15 seconds
tmp	available space in temporary file system	MB	decreasing	N/A	120 seconds
io	disk I/O (shown by lsload -l)	KB per second	increasing	1 minute	15 seconds
name	external load index configured by LSF administrator				site-defined

Status

The `status` index is a string indicating the current status of the host. This status applies to the LIM and RES.

The possible values for `status` are:

Status	Description
ok	The host is available to accept remote jobs. The LIM can select the host for remote execution.
-ok	When the status of a host is preceded by a dash (-), it means that LIM is available but RES is not running on that host or is not responding.
busy	The host is overloaded (busy) because a load index exceeded a configured threshold. An asterisk (*) marks the offending index. LIM will not select the host for interactive jobs.
lockW	The host is locked by its run window. Use <code>lshosts</code> to display run windows.
lockU	The host is locked by an LSF administrator or root.
unavail	The host is down or the LIM on the host is not running or is not responding.

Note: The term `available` is frequently used in command output titles and headings. `Available` means that a host is in any state except `unavail`. This means an `available` host could be, `locked`, `busy`, or `ok`.

CPU run queue lengths (r15s, r1m, r15m)

The `r15s`, `r1m` and `r15m` load indices are the 15-second, 1-minute, and 15-minute average CPU run queue lengths. This is the average number of processes ready to use the CPU during the given interval.

On UNIX, run queue length indices are not necessarily the same as the load averages printed by the `uptime (1)` command; `uptime` load averages on some platforms also include processes that are in short-term wait states (such as paging or disk I/O).

Effective run queue length

On multiprocessor systems, more than one process can execute at a time. LSF scales the run queue value on multiprocessor systems to make the CPU load of uniprocessors and multiprocessors comparable. The scaled value is called the effective run queue length.

Use **lsload -E** to view the effective run queue length.

Normalized run queue length

LSF also adjusts the CPU run queue that is based on the relative speeds of the processors (the CPU factor). The normalized run queue length is adjusted for both number of processors and CPU speed. The host with the lowest normalized run queue length runs a CPU-intensive job the fastest.

Use **lsload -N** to view the normalized CPU run queue lengths.

CPU utilization (ut)

The `ut` index measures CPU utilization, which is the percentage of time spent running system and user code. A host with no process running has a `ut` value of 0 percent; a host on which the CPU is completely loaded has a `ut` of 100 percent.

Paging rate (pg)

The `pg` index gives the virtual memory paging rate in pages per second. This index is closely tied to the amount of available RAM memory and the total size of the processes running on a host; if there is not enough RAM to satisfy all processes, the paging rate is high. Paging rate is a good measure of how a machine responds to interactive use; a machine that is paging heavily feels very slow.

Login sessions (ls)

The `ls` index gives the number of users logged in. Each user is counted once, no matter how many times they have logged into the host.

Interactive idle time (it)

On UNIX, the `it` index is the interactive idle time of the host, in minutes. Idle time is measured from the last input or output on a directly attached terminal or a network pseudo-terminal supporting a login session. This does not include activity directly through the X server such as CAD applications or `emacs` windows, except on Solaris and HP-UX systems.

On Windows, the `it` index is based on the time a screen saver has been active on a particular host.

Temporary directories (tmp)

The `tmp` index is the space available in MB or in units set in `LSF_UNIT_FOR_LIMITS` in `lsf.conf`) on the file system that contains the temporary directory:

- `/tmp` on UNIX
- `C:\temp` on Windows

Swap space (swp)

The `swp` index gives the currently available virtual memory (swap space) in MB or units set in `LSF_UNIT_FOR_LIMITS` in `lsf.conf`). This represents the largest process that can be started on the host.

Memory (mem)

The `mem` index is an estimate of the real memory currently available to user processes, measured in MB or in units set in `LSF_UNIT_FOR_LIMITS` in `lsf.conf`). This represents the approximate size of the largest process that could be started on a host without causing the host to start paging.

LIM reports the amount of free memory available. LSF calculates free memory as a sum of physical free memory, cached memory, buffered memory, and an adjustment value. The command **vmstat** also reports free memory but displays these values separately. There may be a difference between the free memory reported by LIM and the free memory reported by **vmstat** because of virtual memory behavior variations among operating systems. You can write an ELIM that overrides the free memory values that are returned by LIM.

I/O rate (io)

The **io** index measures I/O throughput to disks attached directly to this host, in KB per second. It does not include I/O to disks that are mounted from other hosts.

View information about load indices

lsinfo -l

The **lsinfo -l** command displays all information available about load indices in the system. You can also specify load indices on the command line to display information about selected indices:

```
lsinfo -l swp
RESOURCE_NAME: swp
DESCRIPTION: Available swap space (Mbytes) (alias: swap)
TYPE          ORDER  INTERVAL  BUILTIN  DYNAMIC  RELEASE
Numeric       Dec    60        Yes      Yes      NO
```

lsload -l

The **lsload -l** command displays the values of all load indices. External load indices are configured by your LSF administrator:

```
lsload
HOST_NAME  status  r15s  r1m  r15m  ut   pg   ls   it   tmp  swp  mem
hostN      ok      0.0   0.0  0.1   1%   0.0  1   224  43M  67M  3M
hostK      -ok     0.0   0.0  0.0   3%   0.0  3   0    38M  40M  7M
hostF      busy    0.1   0.1  0.3   7%   *17  6   0    9M   23M  28M
hostG      busy    *6.2  6.9  9.5   85%  1.1  30  0    5M   400M 385M
hostV      unavail
```

About configured resources

LSF schedules jobs that are based on available resources. There are many resources that are built into LSF, but you can also add your own resources, and then use them same way as built-in resources.

For maximum flexibility, you should characterize your resources clearly enough so that users have satisfactory choices. For example, if some of your machines are connected to both Ethernet and FDDI, while others are only connected to Ethernet, then you probably want to define a resource called **fddi** and associate the **fddi** resource with machines connected to FDDI. This way, users can specify resource **fddi** if they want their jobs to run on machines that are connected to FDDI.

- [Adding new resources to your cluster](#)
- [Configuring the **lsf.shared resource section**](#)
- [Configuring **lsf.cluster.cluster_name Host section**](#)
- [Configuring **lsf.cluster.cluster_name ResourceMap section**](#)
- [Reserving a static shared resource](#)
- [External load indices](#)

External load indices report the values of dynamic external resources. A dynamic external resource is a customer-defined resource with a numeric value that changes over time, such as the space available in a directory. Use the external load indices feature to make the values of dynamic external resources available to LSF, or to override the values reported for an LSF built-in load index.

- [External static load indices](#)

External static load indices report the values of static external resources. A static external resource is a user-defined resource which, once detected by LIM at start-up time, has a value that does not change, therefore, define an initial value for these resources. Use the external static load indices feature to make the values of user-defined static numeric or string resources available to LSF or to override the user-defined static resources.

- [Modify a built-in load index](#)

An **elim** executable can be used to override the value of a built-in load index.

Adding new resources to your cluster

Procedure

1. Log in to any host in the cluster as the LSF administrator.
2. Define new resources in the Resource section of `lsf.shared`. Specify at least a name and a brief description, which is displayed to a user by **lsinfo**.
3. For static Boolean resources and static or dynamic string resources, for all hosts that have the new resources, add the resource name to the RESOURCES column in the Host section of `lsf.cluster.cluster_name`.
4. For shared resources, for all hosts that have the new resources, associate the resources with the hosts (you might also have a reason to configure non-shared resources in this section).
5. Run **lsadmin reconfig** to reconfigure LIM.
6. Run **badmin mbdrestart** to restart mbatchd.

Configuring the lsf.shared resource section

About this task

Define configured resources in the Resource section of `lsf.shared`. There is no distinction between shared and non-shared resources. When optional attributes are not specified, the resource is treated as static and Boolean.

Procedure

1. Specify a name and description for the resource, using the keywords `RESOURCENAME` and `DESCRIPTION`. Resource names are case sensitive and can be up to 39 characters in length, with the following restrictions:

- Cannot begin with a number
- Cannot contain the following special characters

```
: . ( ) [ + - * / ! & | < > @ = ,
```

- Cannot be any of the following reserved names:

```
cpu cpuf io logins ls idle maxmem maxswp maxtmp type model
status it mem ncpus nprocs ncores nthreads
define_ncpus_cores define_ncpus_procs define_ncpus_threads
ndisks pg r15m r15s r1m swap swp tmp ut local
dchost jobvm
```

- Cannot begin with `inf` or `nan` (uppercase or lowercase). Use `-R "defined(infixx)"` or `-R "defined(nanxx)"` instead if required.
- For Solaris machines, the keyword `int` is reserved and cannot be used.

2. Optional. Specify optional attributes for the resource.
 - a. Set the resource type (`TYPE = Boolean | String | Numeric`). Default is Boolean.
 - b. For dynamic resources, set the update interval (`INTERVAL`, in seconds).
 - c. For numeric resources, set so that a higher value indicates greater load (`INCREASING = Y`)
 - d. For numeric shared resources, set so that LSF releases the resource when a job using the resource is suspended (`RELEASE = Y`).
 - e. Set resources as consumable in the `CONSUMABLE` column.
Static and dynamic numeric resources can be specified as consumable. A non-consumable resource should not be releasable and should be usable in order, select and same sections of a resource requirement string.

Defaults for built-in indices:

- The following are consumable: r15s, r1m, r15m, ut, pg, io, ls, it, tmp, swp, mem.
- All other built-in static resources are not consumable. (For example, ncpus, ndisks, maxmem, maxswp, maxtmp, cpuf, type, model, status, rexpri, server, hname).

Defaults for external shared resources:

- All numeric resources are consumable.
- String and boolean resources are not consumable.

Note: Non-consumable resources are ignored in rusage sections. LSF rejects resource requirement strings where an rusage section contains a non-consumable resource.

```
Begin Resource
RESOURCENAME  TYPE      INTERVAL INCREASING CONSUMABLE DESCRIPTION # Keywords
patchrev      Numeric   ()        Y           ()          (Patch revision)
specman       Numeric   ()        N           ()          (Specman)
switch        Numeric   ()        Y           N           (Network Switch)
rack          String    ()        ()          ()          (Server room rack)
owner         String    ()        ()          ()          (Owner of the host)
elimres       Numeric   10        Y           ()          (elim generated index)
End Resource
```

3. Run **lsinfo -l** to view consumable resources.

```
lsinfo -l switch
RESOURCE_NAME:  switch
DESCRIPTION: Network Switch
TYPE    ORDER    INTERVAL  BUILTIN  DYNAMIC  RELEASE  CONSUMABLE
Numeric  Inc          0         No       No       No       No

lsinfo -l specman
RESOURCE_NAME:  specman
DESCRIPTION: Specman
TYPE    ORDER    INTERVAL  BUILTIN  DYNAMIC  RELEASE  CONSUMABLE
Numeric  Dec          0         No       No       Yes      Yes
```

Resources required for JSDL

The following resources are pre-defined to support the submission of jobs using JSDL files.

```
Begin Resource
RESOURCENAME  TYPE      INTERVAL INCREASING DESCRIPTION
osname        String    600      ()          (OperatingSystemName)
osver         String    600      ()          (OperatingSystemVersion)
cpuarch       String    600      ()          (CPUArchitectureName)
cpuspeed      Numeric   60       Y           (IndividualCPUSpeed)
bandwidth     Numeric   60       Y           (IndividualNetworkBandwidth)
End Resource
```

Configuring lsf.cluster.cluster_name Host section

About this task

The Host section is the only required section in lsf.cluster.cluster_name. It lists all the hosts in the cluster and gives configuration information for each host. The Host section must precede the ResourceMap section.

Procedure

1. Define the resource names as strings in the Resource section of lsf.shared.
List any number of resources, enclosed in parentheses and separated by blanks or tabs.

Use the RESOURCES column to associate static Boolean resources with particular hosts.
2. Optional. To define shared resources across hosts, use the ResourceMap section.
String resources cannot contain spaces. Static numeric and string resources both use following syntax:

resource_name=resource_value

- *Resource_value* must be alphanumeric.
- For dynamic numeric and string resources, use *resource_name* directly.

Note: If resources are defined in both the resource column of the Host section and the ResourceMap section, the definition in the resource column takes effect.

Example

```
Begin      Host
HOSTNAME  model type server rlm  mem  swp RESOURCES  #Keywords
hostA     !      !    1      3.5  ()   ()  (mg elimres patchrev=3 owner=user1)
hostB     !      !    1      3.5  ()   ()  (specman=5 switch=1 owner=test)
hostC     !      !    1      3.5  ()   ()  (switch=2 rack=rack2_2_3 owner=test)
hostD     !      !    1      3.5  ()   ()  (switch=1 rack=rack2_2_3 owner=test)
End        Host
```

Configuring `lsf.cluster.cluster_name` ResourceMap section

About this task

Resources are associated with the hosts for which they are defined in the ResourceMap section of `lsf.cluster.cluster_name`.

Procedure

For each resource, specify the name (RESOURCE_NAME) and the hosts that have it (LOCATION).

Note: If the ResourceMap section is not defined, then any dynamic resources specified in `lsf.shared` are not tied to specific hosts, but are shared across all hosts in the cluster.

- RESOURCE_NAME: The name of the resource, as defined in `lsf.shared`.
- LOCATION: The hosts that share the resource. For a static resource, you must define an initial value here as well. Do not define a value for a dynamic resource.

Syntax:

```
([resource_value@][host_name... | all [~host_name]... | others | default] ...)
```

- For resource_value, square brackets are not valid.
- For static resources, you must include the resource value, which indicates the quantity of the resource.
- Type square brackets around the list of hosts, as shown. You can omit the parenthesis if you only specify one set of hosts.
- The same host cannot be in more than one instance of a resource, as indicated by square brackets. All hosts within the instance share the quantity of the resource indicated by its value.
- The keyword `all` refers to all the server hosts in the cluster, collectively. Use the not operator (`~`) to exclude hosts or host groups.
- The keyword `others` refers to all hosts not otherwise listed in the instance.
- The keyword `default` refers to each host in the cluster, individually.

Most resources specified in the ResourceMap section are interpreted by LSF commands as shared resources, which are displayed using **lsload -s** or **lshosts -s**.

The exceptions are:

- Non-shared static resources
- Dynamic numeric resources specified using the default keyword. These are host-based resources and behave like the built-in load indices such as `mem` and `swp`. They are viewed using **lsload -l** or **lsload -I**.

Example

A cluster consists of hosts `host1`, `host2`, and `host3`.

```
Begin ResourceMap
RESOURCENAME  LOCATION
verilog       (5@[all ~host1 ~host2])
synopsys      (2@[host1 host2] 2@[others])
console       (1@[host1] 1@[host2] 1@[host3])
xyz           (1@[default])
End ResourceMap
```

In this example:

- Five units of the `verilog` resource are defined on `host3` only (all hosts except `host1` and `host2`).
- Two units of the `synopsys` resource are shared between `host1` and `host2`. 2 more units of the `synopsys` resource are defined on `host3` (shared among all the remaining hosts in the cluster).
- One unit of the `console` resource is defined on each host in the cluster (assigned explicitly). 1 unit of the `xyz` resource is defined on each host in the cluster (assigned with the keyword `default`).

Restriction: For Solaris machines, the keyword `int` is reserved.

Resources required for JSDL

Procedure

To submit jobs using JSDL files, you must uncomment the following lines:

RESOURCENAME	LOCATION
<code>osname</code>	<code>[default]</code>
<code>osver</code>	<code>[default]</code>
<code>cpuarch</code>	<code>[default]</code>
<code>cpuspeed</code>	<code>[default]</code>
<code>bandwidth</code>	<code>[default]</code>

Reserving a static shared resource

About this task

Use resource reservation to prevent over-committing static shared resources when scheduling.

Procedure

To indicate that a shared resource is to be reserved while a job is running, specify the resource name in the `rusage` section of the resource requirement string.

Example

You configured licenses for the Verilog application as a resource called `verilog_lic`. To submit a job to run on a host when there is a license available:

```
bsub -R "select[defined(verilog_lic)] rusage[verilog_lic=1]" myjob
```

If the job can be placed, the license it uses are reserved until the job completes.

External load indices

External load indices report the values of dynamic external resources. A dynamic external resource is a customer-defined resource with a numeric value that changes over time, such as the space available in a directory. Use the external load indices

feature to make the values of dynamic external resources available to LSF, or to override the values reported for an LSF built-in load index.

If you have specific workload or resource requirements at your site, the LSF administrator can define *external resources*. You can use both built-in and external resources for job scheduling and host selection.

- [About external load indices](#)
LSF bases job scheduling and host selection decisions on the resources available within your cluster. A *resource* is a characteristic of a host (such as available memory) or a cluster that LSF uses to make job scheduling and host selection decisions.
- [Configuration to enable external load indices](#)
Enable and configure the parameters to use external load indices.
- [External load indices behavior](#)
- [Configuration to modify external load indices](#)
- [External load indices commands](#)

About external load indices

LSF bases job scheduling and host selection decisions on the resources available within your cluster. A *resource* is a characteristic of a host (such as available memory) or a cluster that LSF uses to make job scheduling and host selection decisions.

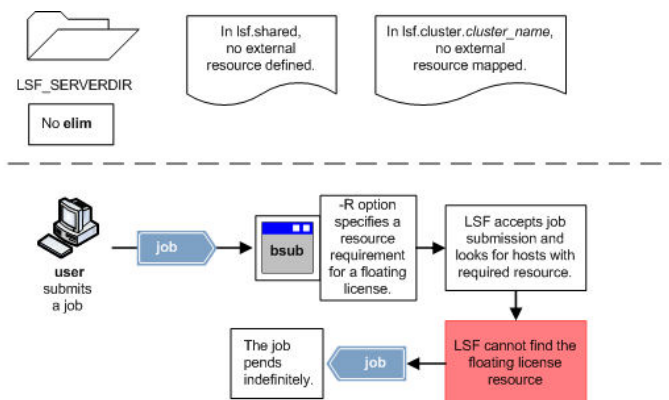
A *static resource* has a value that does not change, such as a host's maximum swap space. A *dynamic resource* has a numeric value that changes over time, such as a host's currently available swap space. *Load indices* supply the values of dynamic resources to a host's load information manager (LIM), which periodically collects those values.

LSF has a number of built-in load indices that measure the values of dynamic, *host-based resources* (resources that exist on a single host); for example, CPU, memory, disk space, and I/O. You can also define *shared resources* (resources that hosts in your cluster share) and make these values available to LSF to use for job scheduling decisions.

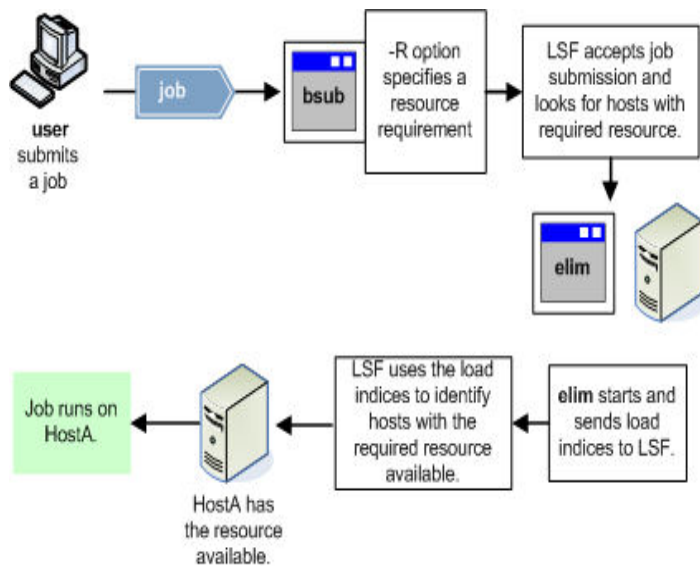
If you have specific workload or resource requirements at your site, the LSF administrator can define *external resources*. You can use both built-in and external resources for LSF job scheduling and host selection.

To supply the LIM with the values of dynamic external resources, either host-based or shared, the LSF administrator writes a site-specific executable called an *external load information manager (elim)* executable. The LSF administrator programs the **elim** to define external load indices, populate those indices with the values of dynamic external resources, and return the indices and their values to stdout. An **elim** can be as simple as a small script, or as complicated as a sophisticated C program. Note: LSF does not include a default **elim**; you should write your own executable to meet the requirements of your site. The following illustrations show the benefits of using the external load indices feature.

Default behavior (feature not enabled)



With external load indices enabled



Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> UNIX Windows A mix of UNIX and Windows hosts
Dependencies	<ul style="list-style-type: none"> UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs. All elim executables run under the same user account as the load information manager (LIM)—by default, the LSF administrator (lsfadmin) or root account. External dynamic resources (host-based or shared) must be defined in lsf.shared.

Configuration to enable external load indices

Enable and configure the parameters to use external load indices.

To enable the use of external load indices, you must

- Define the dynamic external resources in the lsf.shared file. By default, these resources are host-based (local to each host) until the LSF administrator configures a resource-to-host mapping in the ResourceMap section of lsf.cluster.cluster_name file. The presence of the dynamic external resource in the lsf.shared and lsf.cluster.cluster_name files triggers LSF to start the **elim** executable files.
- Map the external resources to hosts in your cluster in the lsf.cluster.cluster_name file.
Important: You must run the **lsadmin reconfig** and **badmin mbdrestart** commands after any resource changes in the lsf.cluster.cluster_name and lsf.shared files to synchronize resource information between the LIM and **mbatchd** daemons.
- Create one or more **elim** executable files in the directory that is specified by the LSF_SERVERDIR parameter. LSF does not include a default **elim**; write your own **elim** executable file to meet the requirements of your site.
- [Define a dynamic external resource](#)
To define a dynamic external resource for which **elim** collects an external load index value, define the following parameters in the Resource section of the lsf.shared file.

- [Map an external resource](#)
Once external resources are defined in `lsf.shared`, they must be mapped to hosts in the ResourceMap section of the `lsf.cluster.cluster_name` file.
- [Create an elim executable file](#)
You can write one or more **elim** executables. The load index names defined in your **elim** executables must be the same as the external resource names defined in the `lsf.shared` configuration file.
- [Overriding built-in load indices](#)
An **elim** executable can be used to override the value of a built-in load index. For example, if your site stores temporary files in the `/usr/tmp` directory, you might want to monitor the amount of space available in that directory. An **elim** can report the space available in the `/usr/tmp` directory as the value for the `tmp` built-in load index.
- [Setting up an ELIM to support JSDL](#)
To support the use of Job Submission Description Language (JSDL) files at job submission, LSF collects the following load indices.
- [Example of an elim executable file](#)
See the section [How environment variables determine elim hosts](#) for an example of a simple **elim** script.

Define a dynamic external resource

To define a dynamic external resource for which **elim** collects an external load index value, define the following parameters in the Resource section of the `lsf.shared` file.

Table 1. Parameters for dynamic external resources

Configuration file	Parameter and syntax	Description
lsf.shared	RESOURCENAME <i>Eresource_name</i>	<ul style="list-style-type: none"> Specifies the name of the external resource.
	TYPENumeric	<ul style="list-style-type: none"> Specifies the type of external resource: Numeric resources have numeric values. Specify Numeric for all dynamic resources.
	INTERVALseconds	<ul style="list-style-type: none"> Specifies the interval for data collection by an elim. For numeric resources, defining an interval identifies the resource as a dynamic resource with a corresponding external load index. <p>Important: You must specify an interval: LSF treats a numeric resource with no interval as a static resource and, therefore, does not collect load index values for that resource.</p>
	INCREASINGY N	<ul style="list-style-type: none"> Specifies whether a larger value indicates a greater load. <ul style="list-style-type: none"> Y: A larger value indicates a greater load. For example, if you define an external load index, the larger the value, the heavier the load. N: A larger value indicates a lighter load.
	RELEASEY N	<ul style="list-style-type: none"> For shared resources only, specifies whether LSF releases the resource when a job that uses the resource is suspended. <ul style="list-style-type: none"> Y: Releases the resource. N: Holds the resource.
	DESCRIPTION <i>d</i> <i>escription</i>	<ul style="list-style-type: none"> Enter a brief description of the resource. The lsinfo command and the <code>ls_info()</code> API call return the contents of the DESCRIPTION parameter.

Map an external resource

Once external resources are defined in `lsf.shared`, they must be mapped to hosts in the ResourceMap section of the `lsf.cluster.cluster_name` file.

Configuration file	Parameter and syntax	Default behavior
lsf.cluster. <i>cluster_name</i>	RESOURCENAME <i>Eresource_name</i>	Specifies the name of the external resource as defined in the Resource section of <code>lsf.shared</code> .
	LOCATION • ([all]) ([all ~host_name ...])	<ul style="list-style-type: none"> Maps the resource to the management host only; all hosts share a single instance of the dynamic external resource. To prevent specific hosts from accessing the resource, use the not operator (~) and specify one or more host names. All other hosts can access the resource.
	[default]	<ul style="list-style-type: none"> Maps the resource to all hosts in the cluster; every host has an instance of the dynamic external resource. If you use the default keyword for any external resource, all elim executable files in <code>LSF_SERVERDIR</code> run on all hosts in the cluster. For information about how to control which elim executable files run on each host, see the section How LSF determines which hosts should run an elim executable file.
	([host_name ...]) ([host_name ...] [host_name ...])	<ul style="list-style-type: none"> Maps the resource to one or more specific hosts. To specify sets of hosts that share a dynamic external resource, enclose each set in square brackets ([]) and use a space to separate each host name.

Create an elim executable file

You can write one or more **elim** executables. The load index names defined in your **elim** executables must be the same as the external resource names defined in the `lsf.shared` configuration file.

All **elim** executables must:

- Be located in `LSF_SERVERDIR` and follow these naming conventions:

Operating system	Naming convention
UNIX	<code>LSF_SERVERDIR\elim.application</code>
Windows	<code>LSF_SERVERDIR\elim.application.exe</code> or
	<code>LSF_SERVERDIR\elim.application.bat</code>

Restriction: The name **elim.user** is reserved for backward compatibility. Do not use the name **elim.user** for your application-specific **elim**.

Note: LSF invokes any **elim** that follows this naming convention: Move backup copies out of `LSF_SERVERDIR` or choose a name that does not follow the convention. For example, use `elim_backup` instead of `elim.backup`.

- Exit upon receipt of a SIGTERM signal from the load information manager (LIM).
- Periodically output a *load update string* to stdout in the format `number_indices index_name index_value [index_name index_value ...]` where

Value	Defines
<i>number_indices</i>	o The number of external load indices that are collected by the elim .
<i>index_name</i>	o The name of the external load index.
<i>index_value</i>	o The external load index value that is returned by your elim .

For example, the string

```
3 tmp2 47.5 nio 344.0 tmp 5
```

reports three indices: `tmp2`, `nio` and `tmp`, with values 47.5, 344.0, and 5, respectively.

- - The load update string must be end with only one `\n` or only one space. In Windows, `echo` will add `\n`.
 - The load update string must report values between `-INFINIT_LOAD` and `INFINIT_LOAD` as defined in the `lsf.h` header file.
 - The **elim** should ensure that the entire load update string is written successfully to stdout. Program the **elim** to exit if it fails to write the load update string to stdout.
 - If the **elim** executable is a C program, check the return value of `printf(3s)`.
 - If the **elim** executable is a shell script, check the return code of `/bin/echo(1)`.
 - If the **elim** executable is implemented as a C program, use `setbuf(3)` during initialization to send unbuffered output to stdout.
 - Each LIM sends updated load information to the management host LIM every 15 seconds; the **elim** executable should write the load update string at most once every 15 seconds. If the external load index values rarely change, program the `elim` to report the new values only when a change is detected.

If you map any external resource as default in `lsf.cluster.cluster_name`, all **elim** executables in `LSF_SERVERDIR` run on all hosts in the cluster. If `LSF_SERVERDIR` contains more than one **elim** executable, you should include a header that checks whether the **elim** is programmed to report values for the resources expected on the host. For detailed information about using a checking header, see the section [How environment variables determine elim hosts](#).

Overriding built-in load indices

An **elim** executable can be used to override the value of a built-in load index. For example, if your site stores temporary files in the `/usr/tmp` directory, you might want to monitor the amount of space available in that directory. An **elim** can report the space available in the `/usr/tmp` directory as the value for the `tmp` built-in load index.

To override a built-in load index value, write an **elim** executable that periodically measures the value of the dynamic external resource and writes the numeric value to standard output. The external load index must correspond to a numeric, dynamic external resource as defined by `TYPE` and `INTERVAL` in `lsf.shared`.

You can find the built-in load index type and name in the **lsinfo** output.

For example, an **elim** collects available space under `/usr/tmp` as 20M. Then, it can report the value as available `tmp` space (the built-in load index `tmp`) in the load update string: `1 tmp 20`.

The following built-in load indices cannot be overridden by **elim**: `logins`, `idle`, `cpu`, and `swap`

Setting up an ELIM to support JSDL

To support the use of Job Submission Description Language (JSDL) files at job submission, LSF collects the following load indices.

Attribute name	Attribute type	Resource name
OperatingSystemName	string	osname
OperatingSystemVersion	string	osver
CPUArchitectureName	string	cpuarch
IndividualCPUSpeed	int64	cpuspeed
IndividualNetworkBandwidth	int64	bandwidth (This is the maximum bandwidth).

The file `elim.jsdl` is automatically configured to collect these resources. To enable the use of `elim.jsdl`, uncomment the lines for these resources in the ResourceMap section of the file `lsf.cluster.cluster_name`.

Example of an elim executable file

See the section [How environment variables determine elim hosts](#) for an example of a simple **elim** script.

You can find more **elim** examples in the `LSF_MISC/examples` directory. The `elim.c` file is an **elim** written in C. You can modify this example to collect the external load indices that are required at your site.

External load indices behavior

How LSF manages multiple elim executables

The LSF administrator can write one **elim** executable to collect multiple external load indices, or the LSF administrator can divide external load index collection among multiple **elim** executables. On each host, the load information manager (LIM) starts a management **elim** (MELIM), which manages all **elim** executables on the host and reports the external load index values to the LIM. Specifically, the MELIM

- Starts **elim** executables on the host. The LIM checks the ResourceMap section LOCATION settings (default, all, or host list) and directs the MELIM to start **elim** executables on the corresponding hosts.
Note:
If the ResourceMap section contains even one resource mapped as default, and if there are multiple **elim** executables in `LSF_SERVERDIR`, the MELIM starts all of the **elim** executables in `LSF_SERVERDIR` on all hosts in the cluster. Not all of the **elim** executables continue to run, however. Those that use a checking header could exit with `ELIM_ABORT_VALUE` if they are not programmed to report values for the resources listed in `LSF_RESOURCES`.
- Restarts an **elim** if the **elim** exits. To prevent system-wide problems in case of a fatal error in the **elim**, the maximum restart frequency is once every 90 seconds. The MELIM does *not* restart any **elim** that exits with `ELIM_ABORT_VALUE`.
- Collects the load information reported by the **elim** executables.
- Checks the syntax of load update strings before sending the information to the LIM.
- Merges the load reports from each **elim** and sends the merged load information to the LIM. If there is more than one value reported for a single resource, the MELIM reports the latest value.
- Logs its activities and data into the log file `LSF_LOGDIR/melim.log.host_name`
- Increases system reliability by buffering output from multiple **elim** executables; failure of one **elim** does not affect other **elim** executables running on the same host.

How LSF determines which hosts should run an elim executable

LSF provides configuration options to ensure that your **elim** executables run only when they can report the resources values expected on a host. This maximizes system performance and simplifies the implementation of external load indices. To control which hosts run **elim** executables, you

- Must map external resource names to locations in `lsf.cluster.cluster_name`
- Optionally, use the environment variables `LSF_RESOURCES`, `LSF_MASTER`, and `ELIM_ABORT_VALUE` in your **elim** executables

How resource mapping determines elim hosts

The following table shows how the resource mapping defined in `lsf.cluster.cluster_name` determines the hosts on which your **elim** executables start.

If the specified LOCATION is ...	Then the elim executables start on ...
----------------------------------	--

If the specified LOCATION is ...	Then the elim executables start on ...
<ul style="list-style-type: none"> • ([all]) ([all ~host_name ...]) 	<ul style="list-style-type: none"> • The management host because all hosts in the cluster (except those identified by the not operator [~]) share a single instance of the external resource.
<ul style="list-style-type: none"> • [default] 	<ul style="list-style-type: none"> • Every host in the cluster because the default setting identifies the external resource as host-based. • If you use the default keyword for any external resource, all elim executables in LSF_SERVERDIR run on all hosts in the cluster. For information about how to program an elim to exit when it cannot collect information about resources on a host, see How environment variables determine elim hosts.
<ul style="list-style-type: none"> • ([host_name ...]) ([host_name ...] [host_name ...]) 	<ul style="list-style-type: none"> • On the specified hosts. • If you specify a set of hosts, the elim executables start on the first host in the list. For example, if the LOCATION in the ResourceMap section of lsf.cluster.cluster_name is ([hostA hostB hostC] [hostD hostE hostF]): <ul style="list-style-type: none"> ◦ LSF starts the elim executables on hostA and hostD to report values for the resources shared by that set of hosts. ◦ If the host reporting the external load index values becomes unavailable, LSF starts the elim executables on the next available host in the list. In this example, if hostA becomes unavailable, LSF starts the elim executables on hostB. ◦ If hostA becomes available again, LSF starts the elim executables on hostA and shuts down the elim executables on hostB.

How environment variables determine elim hosts

If you use the `default` keyword for any external resource in `lsf.cluster.cluster_name`, all **elim** executables in LSF_SERVERDIR run on all hosts in the cluster. You can control the hosts on which your **elim** executables run by using the environment variables LSF_MASTER, LSF_RESOURCES, and ELIM_ABORT_VALUE. These environment variables provide a way to ensure that **elim** executables run only when they are programmed to report the values for resources expected on a host.

- LSF_MASTER—You can program your **elim** to check the value of the LSF_MASTER environment variable. The value is Y on the management host and N on all other hosts. An **elim** executable can use this parameter to check the host on which the **elim** is currently running.
- LSF_RESOURCES—When the LIM starts an MELIM on a host, the LIM checks the resource mapping defined in the ResourceMap section of `lsf.cluster.cluster_name`. Based on the mapping location (default, all, or a host list), the LIM sets LSF_RESOURCES to the list of resources expected on the host.

When the location of the resource is defined as default, the resource is listed in LSF_RESOURCES on the server hosts. When the location of the resource is defined as all, the resource is only listed in LSF_RESOURCES on the management host.

Use LSF_RESOURCES in a checking header to verify that an **elim** is programmed to collect values for at least one of the resources listed in **LSF_RESOURCES**.

- ELIM_ABORT_VALUE—An **elim** should exit with ELIM_ABORT_VALUE if the **elim** is not programmed to collect values for at least one of the resources listed in LSF_RESOURCES. The MELIM does not restart an **elim** that exits with ELIM_ABORT_VALUE. The default value is 97.

The following sample code shows how to use a header to verify that an **elim** is programmed to collect load indices for the resources expected on the host. If the **elim** is not programmed to report on the requested resources, the **elim** does not need to run on the host.

```
#!/bin/sh
# list the resources that the elim can report to lim
my_resource="myrsc"
```

```
# do the check when $LSF_RESOURCES is defined by lim
if [ -n "$LSF_RESOURCES" ]; then
# check if the resources elim can report are listed in $LSF_RESOURCES
res_ok=`echo " $LSF_RESOURCES " | /bin/grep " $my_resource " `
# exit with $ELIM_ABORT_VALUE if the elim cannot report on at least
# one resource listed in $LSF_RESOURCES
    if [ "$res_ok" = "" ] ; then
        exit $ELIM_ABORT_VALUE
    fi
fi
while [ 1 ];do
# set the value for resource "myrsc"
val="1"
# create an output string in the format:
# number_indices index1_name index1_value...
reportStr="1 $my_resource $val"
echo "$reportStr"
# wait for 30 seconds before reporting again
sleep 30
done
```

Configuration to modify external load indices

Configuration file	Parameter and syntax	Behavior
lsf.cluster. <i>cluster_name</i> Parameters section	ELIMARGS= <i>cmd_line_args</i>	<ul style="list-style-type: none"> Specifies the command-line arguments that are required by an elim on startup.
	ELIM_POLL_INTERVAL= <i>seconds</i>	<ul style="list-style-type: none"> Specifies the frequency with which the LIM samples external load index information from the MELIM.
	LSF_ELIM_BLOCKTIME= <i>seconds</i>	<ul style="list-style-type: none"> UNIX only. Specifies how long the MELIM waits before restarting an elim that fails to send a complete load update string. The MELIM does not restart an elim that exits with ELIM_ABORT_VALUE.
	LSF_ELIM_DEBUG= <i>y</i>	<ul style="list-style-type: none"> UNIX only. Used for debugging; logs all load information received from elim executables to the MELIM log file (<i>melim.log.host_name</i>).
	LSF_ELIM_RESTARTS= <i>integer</i>	<ul style="list-style-type: none"> UNIX only. Limits the number of times an elim can be restarted. You must also define either LSF_ELIM_DEBUG or LSF_ELIM_BLOCKTIME. Defining this parameter prevents an ongoing restart loop in the case of a faulty elim.

External load indices commands

Commands to submit workload

Command	Description
---------	-------------

Command	Description
<code>bsub -R "res_req" [-R "res_req"] ...</code>	<ul style="list-style-type: none"> Runs the job on a host that meets the specified resource requirements. If you specify a value for a dynamic external resource in the resource requirements string, LSF uses the most recent values that are provided by your elim executables for host selection. For example: <ul style="list-style-type: none"> Define a dynamic external resource called "usr_tmp" that represents the space available in the /usr/tmp directory. Write an elim executable to report the value of usr_tmp to LSF. To run the job on hosts that have more than 15 MB available in the /usr/tmp directory, run the command <code>bsub -R "usr_tmp > 15" myjob</code> LSF uses the external load index value for usr_tmp to locate a host with more than 15 MB available in the /usr/tmp directory.

Commands to monitor

Command	Description
<code>lsload</code>	<ul style="list-style-type: none"> Displays load information for all hosts in the cluster on a per host basis.
<code>lsload -R "res_req"</code>	<ul style="list-style-type: none"> Displays load information for specific resources.

Commands to control

Command	Description
<code>lsadmin reconfig</code> followed by <code>badmin mbdrestart</code>	<ul style="list-style-type: none"> Applies changes when you modify <code>lsf.shared</code> or <code>lsf.cluster.cluster_name</code>.

Commands to display configuration

Command	Description
<code>lsinfo</code>	<ul style="list-style-type: none"> Displays configuration information for all resources, including the external resources that are defined in <code>lsf.shared</code>.
<code>lsinfo -l</code>	<ul style="list-style-type: none"> Displays detailed configuration information for external resources.
<code>lsinfo resource_name ...</code>	<ul style="list-style-type: none"> Displays configuration information for the specified resources.
<code>bhosts -s</code>	<ul style="list-style-type: none"> Displays information about numeric shared resources, including which hosts that share each resource.
<code>bhosts -s shared_resource_name ...</code>	<ul style="list-style-type: none"> Displays configuration information for the specified resources.

External static load indices

External static load indices report the values of static external resources. A static external resource is a user-defined resource which, once detected by LIM at start-up time, has a value that does not change, therefore, define an initial value for these

resources. Use the external static load indices feature to make the values of user-defined static numeric or string resources available to LSF or to override the user-defined static resources.

Typical examples of static external resources are `ostype` and `osver`, which come from LSF.

The external static resource value is reported by the external static LIM (**eslim**), which is a type of ELIM, and does not change until you restart the LIM service. Use a static external resource for job scheduling as you would for other user-defined external resources and LSF built-in resources.

- [Configuration to enable external static load indices](#)
Enable external static load indices by defining the static external resources and creating **eslim** executable files.
- [Create esim executable files](#)
Create esim executable files to report external static resource values.
- [Example of an esim executable file](#)
Use the following resource configuration and **eslim** script as an example for your own **eslim** files.

Configuration to enable external static load indices

Enable external static load indices by defining the static external resources and creating **eslim** executable files.

To enable the use of external static load indices, you must perform the following:

- Define the static external resources in the `lsf.shared` file. By default, these resources are host-based (local to each host). Important: You must run the **lsadmin reconfig** and **badmin mbdrestart** commands after any resource changes in the `lsf.cluster.cluster_name` and `lsf.shared` files to synchronize resource information between the LIM and **mbatchd** daemons.
- Create one or more **eslim** executable files in the `LSF_SERVERDIR` directory. LSF does not include a default **eslim**; write your own **eslim** executable file to meet the requirements of your site.

Create esim executable files

Create esim executable files to report external static resource values.

The names of the external static load indices whose values are collected by your **eslim** executable files must be the same as the names of the external resources that are defined in the `lsf.shared` configuration file.

All **eslim** executable files must

- Be located in the `LSF_SERVERDIR` directory and follow these naming conventions:

Operating system	Naming convention
UNIX	<code>LSF_SERVERDIR\eslim.application</code>
Windows	<code>LSF_SERVERDIR\eslim.application.exe</code> or <code>LSF_SERVERDIR\eslim.application.bat</code>

Restriction: The name **eslim.user** is reserved for backward compatibility. Do not use the name **eslim.user** for your application-specific **eslim**.

Note: LSF invokes any **eslim** that follows this naming convention: Move backup copies out of `LSF_SERVERDIR` or choose a name that does not follow the convention. For example, use `eslim_backup` instead of `eslim.backup`.

- Output a *load update string* only once to stdout in the format `number_indices index_name index_value [index_name index_value ...]` where

Value	Defines
<code>number_indices</code>	◦ The number of external static load indices that are collected by the eslim .
<code>index_name</code>	◦ The name of the external static load index.

Value	Defines
<i>index_value</i>	◦ The external static load index value that is returned by your eslim .

- Exit after reporting the load update string once.

For example, the string

```
3 tmp2 47.5 nio 344.0 tmp 5
```

reports three indices: `tmp2`, `nio` and `tmp`, with values 47.5, 344.0, and 5, respectively.

- - The load update string must end with only one `\n` or only one space. In Windows, `echo` will add `\n`.
 - The load update string must report values between `-INFINIT_LOAD` and `INFINIT_LOAD` as defined in the `lsf.h` header file.
 - The **eslim** should ensure that the entire load update string is written successfully to `stdout`. Program the **eslim** to exit if it fails to write the load update string to `stdout`.
 - If the **eslim** executable is a C program, check the return value of `printf(3s)`.
 - If the **eslim** executable is a shell script, check the return code of `/bin/echo(1)`.
 - If the **eslim** executable is implemented as a C program, use `setbuf(3)` during initialization to send unbuffered output to `stdout`.

If a static **eslim** program fails or times out, the resources that it reports will not be available on that host. If this occurs, fix the **eslim** program, then run the **lsadmin reconfig** and **badadmin mbdrestart** commands to make the resources available again.

If you map any external resource as default in `lsf.cluster.cluster_name`, all **eslim** executable files in the `LSF_SERVERDIR` directory run on all hosts in the cluster. If `LSF_SERVERDIR` contains more than one **eslim** executable, you should include a header that checks whether the **eslim** is programmed to report values for the resources expected on the host.

Example of an eslim executable file

Use the following resource configuration and **eslim** script as an example for your own **eslim** files.

Example static external resource configuration

The following example adds a resource named `ostype` to the Resource section of the `lsf.shared` file.

```
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION
...
ostype String () () ()
...
End Resource
```

Example eslim script

The following example illustrates how to use the example `ostype` resource in the `lsf.shared` file.

```
#!/bin/sh

resname=ostype

if [ -f /etc/redhat-release ]
then
    # Get the release string
    rel=`head -1 /etc/redhat-release`
    # If the release string is well formed, get the distribution name and
    # version.
    if echo "$rel" | grep ' release ' > /dev/null 2>&1
    then
        dist=`echo "$rel" | sed -e "s/(.*) release.*/\1/"`
        ver=`echo "$rel" | sed -e "s/.* release \([^ ]*\) .*/\1/"`
```

```

        # Special case for some RHEL distributions, strip the AS.
        ver=`echo "$ver" | sed "s/AS$//"`
    else
        # If the release string is not well formed,
        # just pass the text to the user.
        dist="$rel"
        ver=""
    fi
    # Check for an update number.
    if echo "$rel" | grep '(. * Update .*)' > /dev/null 2>&1
    then
        upd=`echo "$rel" | sed -e "s/. * Update \(.*\)\/\1/"`
        ver="$ver.$upd"
    fi
    if echo "$dist" | grep "CentOS" > /dev/null 2>&1
    then
        dist="CENT"
    elif echo "$dist" | grep "^Red Hat Enterprise Linux" > /dev/null 2>&1
    then
        dist="RHEL"
    elif echo "$dist" | grep "^Red Hat Linux" > /dev/null 2>&1
    then
        dist="RHAT"
    elif echo "$dist" | grep "^Fedora" > /dev/null 2>&1
    then
        dist="FEDC"
    else
        dist="RHunknown"
    fi
elif [ -f /etc/SuSE-release ]
then
    # Get the release string
    line1=`cat /etc/SuSE-release | head -1`
    line2=`cat /etc/SuSE-release | head -2 | tail -1`
    line3=`cat /etc/SuSE-release | head -3 | tail -1`

    dist="SLES"

    if echo "$line2" | grep "^VERSION = " > /dev/null 2>&1
    then
        ver=`echo "$line2" | sed -e 's/VERSION = \(.*\)\/\1/'`
    else
        ver=""
    fi

    if echo "$line3" | grep "^PATCHLEVEL = " > /dev/null 2>&1
    then
        patchlev=`echo $line3 | sed -e 's/PATCHLEVEL = \(.*\)\/\1/'`
        ver="$ver.$patchlev"
    fi

elif [ "`uname -s`" = "AIX" ]
then
    dist="AIX"
    ver=`uname -v`.`uname -r`
else
    dist=`uname -s`
    ver=`uname -r`
fi

# Compose the ostype
type="$dist$ver"

# Strip illegal characters
type=`echo $type | tr -cd '[:alnum:]._- '`

# Report the result.
echo "1 $resname $type"

```

After creating the eslim script

After you define the resources and create the **eslim** script, run the following commands to synchronize resource information between the LIM and **mbatchd** daemons:

```
lsadmin reconfig  
badmin mbdrestart
```

To view information on the static external resources that you defined, run the **lshosts -s** or **-l** command options.

Modify a built-in load index

An **elim** executable can be used to override the value of a built-in load index.

For example, if your site stores temporary files in the `/usr/tmp` directory, you might want to monitor the amount of space available in that directory. An **elim** can report the space available in the `/usr/tmp` directory as the value for the `tmp` built-in load index. For detailed information about how to use an **elim** to override a built-in load index, see [External Load Indices](#).

Configure host resources

Add and remove hosts in your cluster. Configure LSF to run batch jobs on dynamic hosts. Configure a host to run one job at a time.

- [Adding a host](#)
Use the LSF installation script **lsfinstall** to add new hosts and host types to your cluster, and the **hostsetup** script to setup LSF to start automatically.
- [Removing a host](#)
Removing a host from LSF involves preventing any additional jobs from running on the host, removing the host from LSF, and removing the host from the cluster. To remove a host from your cluster, remove references to a host in your cluster from `lsf.cluster.cluster_name` and other configuration files.

Adding a host

Use the LSF installation script **lsfinstall** to add new hosts and host types to your cluster, and the **hostsetup** script to setup LSF to start automatically.

- [Dynamically adding hosts](#)
By default, all configuration changes made to LSF are static. To add or remove hosts within the cluster, you must manually change the configuration and restart all management candidates.
- [Adding a host to the cluster using bconf](#)

Adding a host of an existing type with lsfinstall

Use the LSF installation script **lsfinstall** to add more hosts of the same host type to your cluster, and the **hostsetup** script to set up LSF to start automatically.

About this task

Restriction: **lsfinstall** is not compatible with clusters installed with the old **lsfsetup** script. To add a host to a cluster originally installed with **lsfsetup**, you must upgrade your cluster.

Procedure

1. Make sure that the host type exists in your cluster:
 - a. Log on to any host in the cluster. You do not need to be root.

- b. List the contents of the LSF_TOP/10.1.0 directory and confirm that there is already a subdirectory with the name of the host type.

The default LSF_TOP/10.1.0 directory is /usr/share/lsf/10.1.0.

2. Add the host information to lsf.cluster.cluster_name:
- a. Log on to the LSF management host as root.
- b. Edit LSF_CONFDIR/lsf.cluster.cluster_name, and specify the following properties for the host in the Host section:
- The name of the host.
 - The model and type, or specify ! to automatically detect the type or model.
 - Specify 1 for LSF server or 0 for LSF client.

```
Begin Host
HOSTNAME  model  type      server  rlm  mem  RESOURCES  REXPRI
hosta     !      SUNSOL    1       1.0  4    ()         0
hostb     !      AIX       0       1.0  4    ()         0
hostc     !      HPPA      1       1.0  4    ()         0
hostd     !      LINUX     1       1.0  4    ()         0
End Host
```

- c. Save your changes.

3. Run **lsadmin reconfig** to reconfigure LIM.
4. Run **badmin mbdrestart** to restart **mbatchd**.
5. Run **hostsetup** to set up the new host and configure the daemons to start automatically at boot time.
- Important: Before you run **hostsetup**, make sure that the hosts you want to set up are in lsf.cluster.cluster_name.
- For example, run the following commands to use the LSF cluster installed in /usr/share/lsf and configure LSF daemons to start automatically at boot time:

```
# cd /usr/share/lsf/10.1.0/install
# ./hostsetup --top="/usr/share/lsf" --boot="y"
```

For complete **hostsetup** usage, enter **hostsetup -h**.

6. Start LSF on the new host:

```
bctrld start lim
bctrld start res
bctrld start sbd
```

7. Run **bhosts** and **lshosts** to verify your changes.

Adding a host of a new type with lsfinstall

Use the LSF installation script **lsfinstall** to add new host types to your cluster, and the **hostsetup** script to set up LSF to start automatically..

About this task

Restriction:

lsfinstall is not compatible with clusters installed with the old **lsfsetup** script. To add a host to a cluster originally installed with **lsfsetup**, you must upgrade your cluster.

Procedure

1. Make sure that the host type does not exist in your cluster:
 - a. Log on to any host in the cluster. You do not need to be root.
 - b. List the contents of the LSF_TOP/10.1.0 directory. The default is /usr/share/lsf/10.1.0. If the host type currently exists, there is a subdirectory with the name of the host type.
2. Get the LSF distribution file for the host type you want to add.
3. Log on as root to any host that can access the LSF installation directory.
4. Change to the LSF installation directory.

```
% cd /usr/share/lsf/10.1.0/install
```

5. Edit install.config.
 - a. For LSF_TARDIR, specify the path to the directory that contains the distribution file.

```
LSF_TARDIR="/usr/share/lsf_distrib/10.1.0"
```

b. For LSF_ADD_SERVERS, list the new host names that are enclosed in quotation marks and separated by spaces.

```
LSF_ADD_SERVERS="hosta hostb"
```

c. Run **./lsfinstall -f install.config**. The host information is automatically created in `lsf.cluster.cluster_name`.

6. Run **lsadmin reconfig** to reconfigure LIM.

7. Run **badmin reconfig** to reconfigure **mbatchd**.

8. Run **hostsetup** to set up the new host and configure the daemons to start automatically at boot time.

Important: Before you run **hostsetup**, make sure that the hosts you want to set up are in `lsf.cluster.cluster_name`.

For example, run the following commands to use the LSF cluster installed in `/usr/share/lsf` and configure LSF daemons to start automatically at boot time:

```
# cd /usr/share/lsf/10.1.0/install
# ./hostsetup --top="/usr/share/lsf" --boot="y"
```

For complete **hostsetup** usage, enter **hostsetup -h**.

9. Start LSF on the new host:

```
bctrld start lim
bctrld start res
bctrld start sbd
```

10. Run **bhosts** and **lshosts** to test your changes.

Dynamically adding hosts

By default, all configuration changes made to LSF are static. To add or remove hosts within the cluster, you must manually change the configuration and restart all management candidates.

Dynamic host configuration allows you to add and remove hosts without manual reconfiguration. To enable dynamic host configuration, all of the parameters that are described in the following table must be defined.

Parameter	Configuration file	Description
LSF_MASTER_LIST	lsf.conf	Defines a list of management host candidates. These hosts receive information when a dynamic host is added to or removed from the cluster. Do not add dynamic hosts to this list, because dynamic hosts cannot be management hosts.
LSF_DYNAMIC_HOST_WAIT_TIME	lsf.conf	Defines the length of time a dynamic host waits before sending a request to the management host LIM to add the host to the cluster.
LSF_HOST_ADD_RANGE	lsf.cluster.cluster_name	Identifies the range of IP addresses for hosts that can dynamically join or leave the cluster.

Important: If you choose to enable dynamic hosts when you install LSF, the installer adds the `LSF_HOST_ADD_RANGE` parameter to `lsf.cluster.cluster_name` file using a default value that allows any host to join the cluster. To enable security, configure `LSF_HOST_ADD_RANGE` in the `lsf.cluster.cluster_name` file after installation to restrict the hosts that can join your cluster.

- [Configuring and running batch jobs on dynamic hosts](#)

Before you run batch jobs on a dynamic host, complete one or all of the following steps, depending on your cluster configuration.

- [Changing a dynamic host to a static host](#)

If you want to change a dynamic host to a static host, first use the **badmin hghostdel** command to remove the dynamic host from any host group that it belongs to, and then configure the host as a static host in `lsf.cluster.cluster_name`.

- [Adding a dynamic host in a shared file system environment](#)

In a shared file system environment, you do not need to install LSF on each dynamic host. The management host will recognize a dynamic host as an LSF host when you start the daemons on the dynamic host.

- [Adding a dynamic host in a non-shared file system environment](#)

In a non-shared file system environment, you must install LSF binaries, a localized `lsf.conf` file, and shell environment scripts (`cschrc.lsf` and `profile.lsf`) on each dynamic host.

Dynamic host configuration

Dynamic hosts are configured as follows:

Management host LIM

The primary management host LIM runs on the management host for the cluster. The management host LIM receives requests to add hosts, and tells the management host candidates defined by the `LSF_MASTER_LIST` parameter to update their configuration information when a host is dynamically added or removed.

Upon startup, both static and dynamic hosts wait to receive an acknowledgment from the management host LIM. This acknowledgment indicates that the management host LIM has added the host to the cluster. Static hosts normally receive an acknowledgment because the management host LIM has access to static host information in the LSF configuration files. Dynamic hosts do not receive an acknowledgment, however, until they announce themselves to the management host LIM. The `LSF_DYNAMIC_HOST_WAIT_TIME` parameter in the `lsf.conf` file determines how long a dynamic host waits before sending a request to the management host LIM to add the host to the cluster.

Management host candidate host LIMs

The `LSF_MASTER_LIST` parameter defines the list of management host candidates. These hosts receive updated host information from the primary management host LIM so that any management host candidate can take over as the primary management host for the cluster.

Important: Primary management candidate hosts should share LSF configuration and binaries.

Dynamic hosts cannot be primary management host candidates. By defining the `LSF_MASTER_LIST` parameter, you ensure that LSF limits the list of management host candidates to specific, static hosts.

mbatchd

The **mbatchd** daemon receives host information from the management host LIM; when it detects the addition or removal of a dynamic host within the cluster, **mbatchd** automatically reconfigures itself.

Tip: After adding a host dynamically, you might have to wait for **mbatchd** to detect the host and reconfigure. Depending on system load, **mbatchd** might wait up to ten minutes before reconfiguring.

lsadmin command

Use the **bctrl start lim** command to start the LIM on a newly added dynamic host.

Allow only certain hosts to join the cluster

By default, any host can be dynamically added to the cluster. To enable security, define a value for the `LSF_HOST_ADDR_RANGE` parameter in the `lsf.cluster.cluster_namefile` to identify a range of IP addresses for hosts that are allowed to dynamically join the cluster as LSF hosts. IP addresses can have either a dotted quad notation (IPv4) or IP Next Generation (IPv6) format. You can use IPv6 addresses if you define the `LSF_ENABLE_SUPPORT_IPV6` parameter in the `lsf.conf` file. You do not require mapping IPv4 addresses to an IPv6 format.

Configuring and running batch jobs on dynamic hosts

Before you run batch jobs on a dynamic host, complete one or all of the following steps, depending on your cluster configuration.

Procedure

- Configure queues to accept all hosts by defining the `HOSTS` parameter in the `lsb.queues` files with the keyword `all`. Jobs submitted to this queue can run on dynamic hosts.
- Define host groups that accept wildcard characters in the `HostGroup` section of the `lsb.hosts` file. For example, define a host group named `linux_hosts` and specify a group member `linuxrack*` in the `GROUP_MEMBER` parameter in the host group definition.

Jobs submitted a queue that defines the **HOSTS=linux_hosts** host group (which contains `linuxrack*` dynamic hosts) can run on dynamic hosts.

- Add a dynamic host to a host group by using the **badmin hghostadd** command.

Results

To run jobs on the dynamic hosts, submit a job directly to the host group at job level or to the host group defined at the queue level.

Changing a dynamic host to a static host

If you want to change a dynamic host to a static host, first use the **badmin hghostdel** command to remove the dynamic host from any host group that it belongs to, and then configure the host as a static host in `lsf.cluster.cluster_name`.

Adding a dynamic host in a shared file system environment

In a shared file system environment, you do not need to install LSF on each dynamic host. The management host will recognize a dynamic host as an LSF host when you start the daemons on the dynamic host.

Procedure

1. In the `lsf.conf` configuration file on the management host, define the `LSF_DYNAMIC_HOST_WAIT_TIME` parameter, in seconds, and assign a value greater than zero.

`LSF_DYNAMIC_HOST_WAIT_TIME` specifies how long a dynamic host waits before sending a request to the management host **LIM** to add the host to the cluster.

For example:

```
LSF_DYNAMIC_HOST_WAIT_TIME=60
```

2. Define the `LSF_DYNAMIC_HOST_TIMEOUT` parameter.

`LSF_DYNAMIC_HOST_TIMEOUT` specifies how long LSF waits before the management host automatically removes unavailable dynamic host. Each time LSF removes a dynamic host, **mbatchd** daemon automatically reconfigures itself.

Note: For very large clusters, defining this parameter could decrease system performance

For example:

```
LSF_DYNAMIC_HOST_TIMEOUT=60m
```

3. In the `lsf.cluster.cluster_name` configuration file on the management host, define the `LSF_HOST_ADDR_RANGE` parameter.

`LSF_HOST_ADDR_RANGE` enables security by defining a list of hosts that can join the cluster. Specify IP addresses or address ranges for hosts that you want to allow in the cluster.

Note: If you define the `LSF_ENABLE_SUPPORT_IPV6` parameter in the `lsf.conf` file, IP addresses can have either a dotted quad notation (**IPv4**) or IP Next Generation (**IPv6**) format; you do not have to map **IPv4** addresses to an **IPv6** format.

For example:

```
LSF_HOST_ADDR_RANGE=100-110.34.1-10.4-56
```

All hosts belonging to a domain with an address having the first number between 100 and 110, then 34, then a number between one and ten, and then a number between four and 56, will be allowed access. In this example, no **IPv6** hosts are allowed.

4. Log on as root to each host you want to join the cluster.
5. Source the LSF environment:
 - For **csh** or **tcsh**:

```
source LSF_TOP/conf/cshrc.lsf
```

- For **sh**, **ksh**, or **bash**:

```
. LSF_TOP/conf/profile.lsf
```

6. Consider if you want LSF to start automatically when the host reboots.
 - If do not want automatic restarting, go to the next step.
 - If you want automatic restarting, run the **hostsetup** command. For example:

```
cd /usr/share/lsf/10.1.0/install  
./hostsetup --top="/usr/share/lsf" --boot="y"
```

For complete **hostsetup** usage, enter **hostsetup -h**.

7. Use the following commands to start LSF:

```
bctrld start lim  
bctrld start res  
bctrld start sbd
```

Adding a dynamic host in a non-shared file system environment

In a non-shared file system environment, you must install LSF binaries, a localized `lsf.conf` file, and shell environment scripts (`cshrc.lsf` and `profile.lsf`) on each dynamic host.

Specifying installation options in the `server.config` file

All dynamic hosts are server hosts because they cannot serve as management host candidates. The `server.config` file contains parameters for configuring all server hosts.

Procedure

1. Define the required parameters:
 - **LSF_SERVER_HOSTS**="*host_name* [*host_name* ...]"
 - **LSF_ADMINS**="*user_name* [*user_name* ...]"
 - **LSF_TOP**="*/path*"
2. Optionally define the **LSF_LIM_PORT** parameter:

```
LSF_LIM_PORT=port_number
```

Important: If the management host does not use the default **LSF_LIM_PORT**, you must specify the same **LSF_LIM_PORT** defined in `lsf.conf` on the management host.

Adding local resources on a dynamic host to the cluster

Ensure that the resource name and type are defined in `lsf.shared` file, and that the **ResourceMap** section of the `lsf.cluster.cluster_name` file contains at least one resource mapped to at least one static host. LSF can add local resources as long as the **ResourceMap** section is defined; you do not need to map the local resources.

Procedure

In the `server.config` file, define the **LSF_LOCAL_RESOURCES** parameter.
For numeric resources, define name-value pairs:

```
"[resourcemap value*resource_name]"
```

For Boolean resources, the value is the resource name in the following format:

```
"[resource resource_name]"
```

For example:

```
LSF_LOCAL_RESOURCES="[resourcemap 1*verilog] [resource linux]"
```

Tip: If LSF_LOCAL_RESOURCES are already defined in a local `lsf.conf` file on the dynamic host, **lsfinstall** does not add resources you define in LSF_LOCAL_RESOURCES in the `server.config` file.

When the dynamic host sends a request to the management host to add it to the cluster, the dynamic host also reports its local resources. If the local resource is already defined in the `lsf.cluster.cluster_name` file as **default** or **all**, it cannot be added as a local resource.

Installing LSF on a dynamic host

Procedure

Run **lsfinstall -s -f server.config**.

lsfinstall creates a local `lsf.conf` file for the dynamic host, which sets the following parameters:

- **LSF_CONFDIR**="/path"
- **LSF_GET_CONF**=lim
- **LSF_LIM_PORT**=port_number (same as the management host **LIM** port number)
- **LSF_LOCAL_RESOURCES**="resource ..."
Tip: Do not duplicate LSF_LOCAL_RESOURCES entries in the `lsf.conf` file. If local resources are defined more than once, only the last definition is valid.
- **LSF_SERVER_HOSTS**="host_name [host_name
...]"
- **LSF_VERSION**=10.1.0

Configuring dynamic host parameters

Procedure

1. In the `lsf.conf` configuration file on the management host, define the **LSF_DYNAMIC_HOST_WAIT_TIME** parameter, in seconds, and assign a value greater than zero.

LSF_DYNAMIC_HOST_WAIT_TIME specifies how long a dynamic host waits before sending a request to the management host **LIM** to add the host to the cluster.

For example:

```
LSF_DYNAMIC_HOST_WAIT_TIME=60
```

2. Define the **LSF_DYNAMIC_HOST_TIMEOUT** parameter.

LSF_DYNAMIC_HOST_TIMEOUT specifies how long LSF waits before the management host automatically removes unavailable dynamic host. Each time LSF removes a dynamic host, **mbatchd** daemon automatically reconfigures itself.

Note: For very large clusters, defining this parameter could decrease system performance

For example:

```
LSF_DYNAMIC_HOST_TIMEOUT=60m
```

3. In the `lsf.cluster.cluster_name` configuration file on the management host, define the **LSF_HOST_ADDR_RANGE** parameter.

LSF_HOST_ADDR_RANGE enables security by defining a list of hosts that can join the cluster. Specify IP addresses or address ranges for hosts that you want to allow in the cluster.

Note: If you define the **LSF_ENABLE_SUPPORT_IPV6** parameter in the `lsf.conf` file, IP addresses can have either a dotted quad notation (**IPv4**) or IP Next Generation (**IPv6**) format; you do not have to map **IPv4** addresses to an **IPv6** format.

For example:

```
LSF_HOST_ADDR_RANGE=100-110.34.1-10.4-56
```

All hosts belonging to a domain with an address having the first number between 100 and 110, then 34, then a number between one and ten, and then a number between four and 56, will be allowed access. In this example, no **IPv6** hosts

are allowed.

Starting LSF daemons

Procedure

1. Log on as root to each host you want to join the cluster.
2. Source the LSF environment:

- For **cs**h or **tc**sh:

```
source LSF_TOP/conf/cshrc.lsf
```

- For **sh**, **ksh**, or **bash**:

```
. LSF_TOP/conf/profile.lsf
```

3. Consider if you want LSF to start automatically when the host reboots.
 - If do not want automatic restarting, go to the next step.
 - If you want automatic restarting, run the **hostsetup** command. For example:

```
cd /usr/share/lsf/10.1.0/install  
./hostsetup --top="/usr/share/lsf" --boot="y"
```

For complete **hostsetup** usage, enter **hostsetup -h**.

4. Start the daemons.
Assuming RSH (or password-less SSH) is set up, run the startup commands so that they take effect on the host being added to the cluster.

- To run the commands on the host being added:

```
bctrld start lim  
bctrld start res  
bctrld start sbd
```

- To run the commands from another host (for example, if you want to start the daemons on **hostB** from **hostA**):

```
rsh hostB bctrld start lim  
rsh hostB bctrld start res  
rsh hostB bctrld start sbd
```

Adding a host to the cluster using bconf

About this task

You can add a new server host with boolean resources to your cluster using live reconfiguration.

Procedure

Run **bconf add host=hostname**

For example:

```
bconf add host=host24 "MXJ=21;RESOURCES=bigmem"  
bconf: Request for host <host24> accepted
```

Results

Restriction: If default is already defined in `lsb.hosts` without a model or type line, no new line is added to the `lsb.hosts` file.
(Applies to hosts added without batch parameters.)

When using multicluster you cannot add leased hosts or any hosts from another cluster.

Newly added hosts do not join an existing advance reservation, or run existing pending jobs submitted to a host group with **bsub -m** where more than one host or host group is specified.

Adding a faster host to the cluster does not update the RUNLIMIT definition in the queue to normalize with the new cpu factor.

Removing a host

Removing a host from LSF involves preventing any additional jobs from running on the host, removing the host from LSF, and removing the host from the cluster. To remove a host from your cluster, remove references to a host in your cluster from `lsf.cluster.cluster_name` and other configuration files.

About this task

CAUTION:

Never remove the management host from LSF. If you want to remove your current default management host from LSF, change `lsf.cluster.cluster_name` to assign a different default management host. Then, remove the host that was once the management host.

Procedure

1. Log on to the LSF host as root.
 2. Run **badadmin hclose** to close the host. Closing the host prevents jobs from being dispatched to the host and allows running jobs to finish.
 3. Stop all running daemons manually.
 4. Remove any references to the host in the Host section of `LSF_CONFDIR/lsf.cluster.cluster_name`.
 5. Remove any other references to the host, if applicable, from the following LSF configuration files:
 - `LSF_CONFDIR/lsf.shared`
 - `LSB_CONFDIR/cluster_name/configdir/lsb.hosts`
 - `LSB_CONFDIR/cluster_name/configdir/lsb.queues`
 - `LSB_CONFDIR/cluster_name/configdir/lsb.resources`
 6. Log off the host to be removed, and log on as `root` or the primary LSF administrator to any other host in the cluster.
 7. Run **lsadmin reconfig** to reconfigure LIM.
 8. Run **badadmin mbdrestart** to restart **mbatchd**.
 9. If you configured LSF daemons to start automatically at system start, remove the LSF section from the host's system start files.
- [Removing a host from management candidate list](#)
 - [Removing dynamic hosts](#)
- Removing a dynamic host involves setting a timeout value (`LSF_DYNAMIC_HOST_TIMEOUT`) to notify LSF about when to remove the host. If a host is in unavailable status for longer than the time specified for the timeout value, then LSF marks the host as expired and removes it.

Removing a host from management candidate list

About this task

You can remove a host from the management candidate list so that it can no longer be the management host should failover occur. You can choose to either keep it as part of the cluster or remove it.

Procedure

1. Shut down the current LIM:
limshutdown *host_name*

If the host was the current management host, failover occurs.

2. In `lsf.conf`, remove the host name from `LSF_MASTER_LIST`.
3. Run **lsadmin reconfig** for the remaining management candidates.
4. If the host you removed as a management candidate still belongs to the cluster, start up the LIM again:
limstartup *host_name*

Removing dynamic hosts

Removing a dynamic host involves setting a timeout value (`LSF_DYNAMIC_HOST_TIMEOUT`) to notify LSF about when to remove the host. If a host is in unavailable status for longer than the time specified for the timeout value, then LSF marks the host as expired and removes it.

About this task

Tip: Alternatively, to remove a dynamic host as an administrator, shut down LSF daemons on the host, run the `lsfadmin expire hostname` command to mark the host as expired, and then remove the host.

The [LSF_DYNAMIC_HOST_TIMEOUT](#) parameter in the `lsf.conf` configuration file specifies the length of time (minimum 10 minutes) that a dynamic host is unavailable before the management host removes it from the cluster.

Tip: For very large clusters, defining this parameter could decrease system performance. If you want to use this parameter to remove dynamic hosts from a very large cluster, disable the parameter after LSF has removed the unwanted hosts.

Procedure

In the `lsf.conf` file on the management host, define the `LSF_DYNAMIC_HOST_TIMEOUT` parameter.
To specify minutes rather than hours, append `m` or `M` to the value.

For example, to specify 60 minutes:

```
LSF_DYNAMIC_HOST_TIMEOUT=60m
```

Tip: An administrator can remove dynamic hosts by editing the `hostcache` file manually, rather than use this timeout mechanism. See the [post configuration steps for administrators](#) for details.

What to do next

The `LSF_DYNAMIC_HOST_TIMEOUT` value handles notifying LSF of when to remove dynamic host that is unavailable. You should not require manually removing the host after Fix Pack 14. However, if advised by an administrator or IBM support, you can do so, as follows:

1. Shut down the cluster:

```
lsfshutdown
```

This shuts down LSF on all hosts in the cluster and prevents LIMs from trying to write to the `hostcache` file while you edit it.

2. In the `hostcache` file, delete the line for the dynamic host that you removed:
 - If EGO is enabled, the `hostcache` file is in `$EGO_WORKDIR/lim` directory.
 - If EGO is not enabled, the `hostcache` file is in `$LSB_SHAREDIR` directory.
3. Save and close the `hostcache` file, and then start the cluster:

```
lsfrestart
```

Share resources in queues

Learn how to configure LSF queues. Use LSF commands to control queues (close, open, activate, inactivate). Configure dispatch and run windows in queues. Restrict which hosts can use queues. Restrict the job size requested by parallel jobs in a queue. Add queue administrators and give users access to queues. Control job order within queues and switch jobs from one queue to another. Configure an exclusive queue.

- [Controlling queues](#)
Queues are controlled by an LSF administrator or root queue control command or with dispatch and run windows configured in the queue. Use LSF commands and configuration to close, open, deactivate, and activate a queue. Add and remove queues and queue administrators. Configure dispatch and run windows. Restrict hosts and jobs that can use queues.
- [Change job order within queues](#)
By default, LSF dispatches jobs in a queue in the order of arrival (that is, first-come, first-served), subject to availability of suitable server hosts.
- [Switch jobs from one queue to another](#)
You can use the commands **bswitch** and **bmod** to change jobs from one queue to another. This is useful if you submit a job to the wrong queue, or if the job is suspended because of queue thresholds or run windows and you would like to resume the job.

Controlling queues

Queues are controlled by an LSF administrator or root queue control command or with dispatch and run windows configured in the queue. Use LSF commands and configuration to close, open, deactivate, and activate a queue. Add and remove queues and queue administrators. Configure dispatch and run windows. Restrict hosts and jobs that can use queues.

- [Closing a queue](#)
Close a queue to prevent jobs from being submitted to the queue.
- [Opening a queue](#)
Open a closed queue so users can submit jobs to it.
- [Deactivating a queue](#)
Deactivate a queue to stop submitted jobs from being dispatched from the queue.
- [Activating a queue](#)
Activate a deactivated queue so that submitted jobs are dispatched from the queue.
- [Logging a comment on a queue control command](#)
When you open, close, activate, or deactivate a queue, add a comment to give more information about the queue control action.
- [Configuring dispatch windows](#)
A dispatch window specifies one or more time periods during which batch jobs are dispatched to run on hosts.
- [Configuring run windows](#)
A run window specifies one or more time periods during which jobs dispatched from a queue are allowed to run.
- [Adding a queue](#)
Edit the `lsb.queues` file to add the new queue definition. Adding a queue does not affect pending or running jobs.
- [Removing a queue](#)
Edit `lsb.queues` to remove a queue definition.
- [Restricting which hosts can use queues](#)
You might want a host to be used only to run jobs that are submitted to specific queues.
- [Restricting job size requested by parallel jobs in a queue](#)
When users submit, modify, or switch parallel jobs with the **bsub** and **bmod -n** option to explicitly request a job slot size, or with the **-R** option to specify resource requirements, administrators can restrict the number of job slots that are requested for the queue.
- [Adding queue administrators](#)
Queue administrators have limited privileges; they can perform administrative operations (open, close, activate, deactivate) on the specified queue, or on jobs that are running in the specified queue. Queue administrators are optionally configured after installation.

Closing a queue

Close a queue to prevent jobs from being submitted to the queue.

Procedure

Run **badadmin qclose**:

```
Queue <normal> is closed
```

When a user tries to submit a job to a closed queue, the following message is displayed:

```
bsub -q normal ...  
normal: Queue has been closed
```

Opening a queue

Open a closed queue so users can submit jobs to it.

Procedure

Run **badadmin qopen**:

```
badadmin qopen normal  
Queue <normal> is opened
```

Deactivating a queue

Deactivate a queue to stop submitted jobs from being dispatched from the queue.

Procedure

Run **badadmin qinact**:

```
badadmin qinact normal  
Queue <normal> is inactivated
```

Activating a queue

Activate a deactivated queue so that submitted jobs are dispatched from the queue.

Procedure

Run **badadmin qact**:

```
badadmin qact normal  
Queue <normal> is activated
```

Logging a comment on a queue control command

When you open, close, activate, or deactivate a queue, add a comment to give more information about the queue control action.

Procedure

1. Use the **-C** option of **badmin** queue commands **qclose**, **qopen**, **qact**, and **qinact** to log an administrator comment in `lsb.events`.

```
badmin qclose -C "change configuration" normal
```

The comment text `change configuration` is recorded in `lsb.events`.

A new event record is recorded for each queue event. For example, the following commands generate records in `lsb.events`:

```
badmin qclose -C "add user" normal
badmin qclose -C "add user user1" normal
```

The following records are generated:

```
"QUEUE_CTRL" "10.1.0 1050082373 1 "normal" 32185 "lsfadmin" "add user"
"QUEUE_CTRL" "10.1.0 1050082380 1 "normal" 32185 "lsfadmin" "add user user1"
```

2. Use **badmin hist** or **badmin qhist** to display administrator comments for closing and opening hosts.

```
badmin qhist
Fri Apr 4 10:50:36: Queue <normal> closed by administrator <lsfadmin> change
configuration.
```

bqueues -l also displays the comment text:

```
bqueues -l normal
QUEUE: normal
-- For normal low priority jobs, running only if hosts are lightly loaded. Th is is the
default queue.

PARAMETERS/STATISTICS
PRIO NICE STATUS          MAX JL/U JL/P JL/H NJOBS  PEND  RUN  SSUSP USUSP  RSV
30   20  Closed:Active      -   -   -   -   0     0    0    0    0    0
Interval for a host to accept two jobs is 0 seconds
THREADLIMIT
7

SCHEDULING PARAMETERS
          r15s  r1m  r15m  ut      pg      io      ls      it      tmp      swp      mem
loadSched -    -    -    -      -      -      -      -      -      -      -
loadStop  -    -    -    -      -      -      -      -      -      -      -

          cpuspeed  bandwidth
loadSched          -          -
loadStop           -          -

JOB EXCEPTION PARAMETERS
          OVERRUN(min)  UNDERRUN(min)  IDLE(cputime/runtime)
Threshold             -           2          -
Jobs                  -           0          -

USERS:  all users
HOSTS:  all
RES_REQ: select[type==any]

ADMIN ACTION COMMENT: "change configuration"
```

Configuring dispatch windows

A dispatch window specifies one or more time periods during which batch jobs are dispatched to run on hosts.

About this task

Jobs are not dispatched outside of configured windows. Dispatch windows do not affect job submission and running jobs (they are allowed to run until completion). By default, queues are always Active; you must explicitly configure dispatch windows in the queue to specify a time when the queue state is Inactive.

Procedure

1. Edit `lsb.queues`
2. Create a `DISPATCH_WINDOW` keyword for the queue and specify one or more time windows.

```
Begin Queue
QUEUE_NAME    = queue1
PRIORITY      = 45
DISPATCH_WINDOW = 4:30-12:00
End Queue
```

3. Reconfigure the cluster:
 - a. Run **lsadmin reconfig**.
 - b. Run **badmin reconfig**.
4. Run **bqueues -l** to display the dispatch windows.

Configuring run windows

A run window specifies one or more time periods during which jobs dispatched from a queue are allowed to run.

About this task

When a run window closes, running jobs are suspended, and pending jobs remain pending. The suspended jobs are resumed when the window opens again. By default, queues are always Active and jobs can run until completion. You must explicitly configure run windows in the queue to specify a time when the queue state is Inactive.

Procedure

1. Edit `lsb.queues`.
2. Create a `RUN_WINDOW` keyword for the queue and specify one or more time windows.

```
Begin Queue
QUEUE_NAME    = queue1
PRIORITY      = 45
RUN_WINDOW    = 4:30-12:00
End Queue
```

3. Reconfigure the cluster:
 - a. Run **lsadmin reconfig**.
 - b. Run **badmin reconfig**.
4. Run **bqueues -l** to display the run windows.

Adding a queue

Edit the `lsb.queues` file to add the new queue definition. Adding a queue does not affect pending or running jobs.

Procedure

1. Log in as the administrator on any host in the cluster.
2. Edit the `LSB_CONFDIR/cluster_name/configdir/lsb.queues` file to add the new queue definition.
You can copy another queue definition from this file as a starting point. Remember to change the `QUEUE_NAME` parameter of the copied queue.
3. Save the changes to the `lsb.queues` file.
4. When the configuration files are ready, run the **badadmin ckconfig** command to check the new queue definition.
If any errors are reported, fix the problem and check the configuration again.
5. Run the **badadmin reconfig** command to reconfigure the cluster.

```
% badadmin reconfig
Checking configuration files ...
No errors found.
Do you want to reconfigure? [y/n] y
Reconfiguration initiated
```

The **badadmin reconfig** command also checks for configuration errors. If no unrecoverable errors are found, you are asked to confirm reconfiguration. If unrecoverable errors are found, reconfiguration exits.

Results

If you get errors, see [Troubleshooting LSF problems](#) for help with some common configuration errors.

- For more information about the [lsb.queues](#) file, see the [Configuration Reference](#).
- For more information about the **badadmin reconfig** command, see the [Command Reference](#).

Example

```
Begin Queue
QUEUE_NAME = normal
PRIORITY = 30
STACKLIMIT= 2048
DESCRIPTION = For normal low priority jobs, running only if hosts are lightly loaded.
QJOB_LIMIT = 60      # job limit of the queue
PJOB_LIMIT = 2       # job limit per processor
ut = 0.2
io = 50/240
USERS = all
HOSTS = all
NICE = 20
End Queue
```

Removing a queue

Edit `lsb.queues` to remove a queue definition.

Before you begin

Important: Before you remove a queue, make sure that no jobs are running in the queue.

Use the **bqueues** command to view a list of existing queues and the jobs that are running in those queues. If jobs are in the queue that you want to remove, you must switch pending and running jobs to another queue, then remove the queue. If you remove a queue that has pending jobs in it, the jobs are temporarily moved to a `lost_and_found` queue. The job state does not change. Running jobs continue, and jobs that are pending in the original queue are pending in the `lost_and_found` queue. Jobs remain pending until the user or the queue administrator uses the **bswitch** command to switch the jobs into a regular queue. Jobs in other queues are not affected.

Procedure

1. Log in as the primary administrator on any host in the cluster.

2. Close the queue to prevent any new jobs from being submitted.

```
badadmin qclose night
Queue night is closed
```

3. Switch all pending and running jobs into another queue.

For example, the **bswitch -q night idle 0** command chooses jobs from the `night` queue to the `idle` queue. The job ID number 0 switches all jobs.

```
bjobs -u all -q night
JOBID USER  STAT  QUEUE FROM_HOST EXEC_HOST  JOB_NAME  SUBMIT_TIME
5308  user5  RUN   night hostA   hostD     job5      Nov 21 18:16
5310  user5  PEND  night hostA   hostC     job10     Nov 21 18:17

bswitch -q night idle 0
Job <5308> is switched to queue <idle>
Job <5310> is switched to queue <idle>
```

4. Edit the `LSB_CONFDIR/cluster_name/configdir/lsb.queues` file and remove or comment out the definition for the queue that you want to remove.
5. Save the changes to the `lsb.queues` file.
6. Run the **badadmin reconfig** command to reconfigure the cluster.

```
% badadmin reconfig
Checking configuration files ...
No errors found.
Do you want to reconfigure? [y/n] y
Reconfiguration initiated
```

The **badadmin reconfig** command checks for configuration errors. If no unrecoverable errors are found, you are asked to confirm reconfiguration. If unrecoverable errors are found, reconfiguration exits.

Results

If you get errors, see [Troubleshooting LSF problems](#) for help with some common configuration errors.

- For more information about the [lsb.queues](#) file, see the [Configuration Reference](#).
- For more information about the **badadmin reconfig** command, see the [Command Reference](#).

Restricting which hosts can use queues

You might want a host to be used only to run jobs that are submitted to specific queues.

About this task

For example, if you just added a host for a specific department such as engineering, you might want only jobs that are submitted to the queues `engineering1` and `engineering2` to be able to run on the host.

Procedure

1. Log on as root or the LSF administrator on any host in the cluster.
2. Edit `lsb.queues`, and add the host to the `HOSTS` parameter of specific queues.

```
Begin Queue
QUEUE_NAME = queue1
...
HOSTS=mynewhost hostA hostB
...
End Queue
```

3. Save the changes to `lsb.queues`.

4. Use **badadmin ckconfig** to check the new queue definition. If any errors are reported, fix the problem and check the configuration again.
5. Run **badadmin reconfig** to reconfigure **mbatchd**.
6. If you add a host to a queue, the new host is recognized by jobs that were submitted before you reconfigured. If you want the new host to be recognized, you must use the command **badadmin mbdrestart**.

Restricting job size requested by parallel jobs in a queue

When users submit, modify, or switch parallel jobs with the **bsub** and **bmod -n** option to explicitly request a job slot size, or with the **-R** option to specify resource requirements, administrators can restrict the number of job slots that are requested for the queue.

About this task

LSF rejects job submission or pends existing jobs that request job slot sizes that are not in this list. LSF also rejects jobs that request multiple job slot sizes. The first slot size in this list is the default job size, which is the job size that is assigned to jobs that do not explicitly request a job size. The rest of the list can be defined in any order.

For example, if the job size list for the `queue1` queue allows 2, 4, 8, and 16 job slots, and you submit a parallel job that requests 10 job slots in this queue (`bsub -q queue1 -n 10 ...`), that job is rejected because the job size of 10 is not explicitly allowed in the list. To assign a default job size of 4, specify 4 as the first value in the list. Job submissions that do not use **-n** are automatically assigned a job size of 4.

When you use resource requirements to specify job slot size, the request must specify a single fixed number of job slots and not multiple values or a range of values:

- When you use compound resource requirements with the **-n** and **-R** options, make sure that the compound resource requirement matches the **-n** value, which must match a value in the job size list.
- When you use compound resource requirements without **-n**, the compound resource requirement must imply a fixed number of job slots. The implied total number of job slots must match a value in the job size list.
- When you use alternative resource requirements, each of the alternatives must request a fixed number of slots, and all alternative values must match the values in the job size list.

Procedure

1. Log on as root or the LSF administrator on any host in the cluster.
2. Edit `lsb.queues`, and define the `JOB_SIZE_LIST` parameter in specific queues.

```
JOB_SIZE_LIST=default_size [size ...]
```

```
Begin Queue
QUEUE_NAME = queue1
...
JOB_SIZE_LIST=4 2 8 16
...
End Queue
```

3. Save the changes to `lsb.queues`.
4. Use **badadmin ckconfig** to check the new queue definition. If any errors are reported, fix the problem and check the configuration again.
5. Run **badadmin reconfig** to reconfigure **mbatchd**.

Adding queue administrators

Queue administrators have limited privileges; they can perform administrative operations (open, close, activate, deactivate) on the specified queue, or on jobs that are running in the specified queue. Queue administrators are optionally configured after installation.

About this task

Queue administrators cannot modify configuration files, or operate on LSF daemons or on queues they are not configured to administer.

To switch a job from one queue to another, you must have administrator privileges for both queues.

Procedure

In the `lsb.queues` file, between `Begin Queue` and `End Queue` for the appropriate queue, specify the `ADMINISTRATORS` parameter, followed by the list of administrators for that queue. Separate the administrator names with a space. You can specify user names and group names.

```
Begin Queue
ADMINISTRATORS = User1 GroupA
End Queue
```

Change job order within queues

By default, LSF dispatches jobs in a queue in the order of arrival (that is, first-come, first-served), subject to availability of suitable server hosts.

Use the **btop** and **bbot** commands to change the position of pending jobs, or of pending job array elements, to affect the order in which jobs are considered for dispatch. Users can only change the relative position of their own jobs, and LSF administrators can change the position of any users' jobs.

bbot

Moves jobs relative to your last job in the queue.

If invoked by a regular user, **bbot** moves the selected job after the last job with the same priority submitted by the user to the queue.

If invoked by the LSF administrator, **bbot** moves the selected job after the last job with the same priority submitted to the queue.

btop

Moves jobs relative to your first job in the queue.

If invoked by a regular user, **btop** moves the selected job before the first job with the same priority submitted by the user to the queue.

If invoked by the LSF administrator, **btop** moves the selected job before the first job with the same priority submitted to the queue.

Move a job to the top of the queue

In the following example, job 5311 is moved to the top of the queue. Since job 5308 is already running, job 5311 is placed in the queue after job 5308.

Note that **user1**'s job is still in the same position on the queue. **user2** cannot use **btop** to get extra jobs at the top of the queue; when one of his jobs moves up the queue, the rest of his jobs move down.

```
bjobs -u all
JOBID  USER  STAT  QUEUE   FROM_HOST  EXEC_HOST  JOB_NAME    SUBMIT_TIME
5308   user2  RUN   normal  hostA      hostD      /s500       Oct 23 10:16
5309   user2  PEND  night   hostA      /s200       Oct 23 11:04
```

```

5310 user1 PEND night hostB /myjob Oct 23 13:45
5311 user2 PEND night hostA /s700 Oct 23 18:17

```

btop 5311
Job <5311> has been moved to position 1 from top.

```

bjobs -u all

```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
5308	user2	RUN	normal	hostA	hostD	/s500	Oct 23 10:16
5311	user2	PEND	night	hostA		/s200	Oct 23 18:17
5310	user1	PEND	night	hostB		/myjob	Oct 23 13:45
5309	user2	PEND	night	hostA		/s700	Oct 23 11:04

Switch jobs from one queue to another

You can use the commands **bswitch** and **bmod** to change jobs from one queue to another. This is useful if you submit a job to the wrong queue, or if the job is suspended because of queue thresholds or run windows and you would like to resume the job.

- [Switching a single job to a different queue](#)
- [Switching all jobs to a different queue](#)
- [Use external job switch controls](#)

Use the external job switch controls to use external, site-specific binary files or scripts that are associated with the switch request.

Switching a single job to a different queue

Procedure

Run **bswitch** or **bmod** to move pending and running jobs from queue to queue. By default, LSF dispatches jobs in a queue in order of arrival, so a pending job goes to the last position of the new queue, no matter what its position was in the original queue.

In the following example, job 5309 is switched to the **priority** queue:

```

bswitch priority 5309
Job <5309> is switched to queue <priority>
bjobs -u all

```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
5308	user2	RUN	normal	hostA	hostD	/job500	Oct 23 10:16
5309	user2	RUN	priority	hostA	hostB	/job200	Oct 23 11:04
5311	user2	PEND	night	hostA		/job700	Oct 23 18:17
5310	user1	PEND	night	hostB		/myjob	Oct 23 13:45

Switching all jobs to a different queue

Procedure

Run **bswitch -q from_queue to_queue 0** to switch all the jobs in a queue to another queue.

The **-q** option is used to operate on all jobs in a queue. The job ID number 0 specifies that all jobs from the night queue should be switched to the idle queue:

The following example selects jobs from the **night** queue and switches them to the **idle** queue.

```

bswitch -q night idle 0
Job <5308> is switched to queue <idle>
Job <5310> is switched to queue <idle>

```

Use external job switch controls

Use the external job switch controls to use external, site-specific binary files or scripts that are associated with the switch request.

A user runs the **bswitch** command to switch jobs to a different queue. External job switch controls allow users to specify an application-specific external executable file (**eswitch**) to associate with the switch request.

LSF first invokes the executable file named **eswitch** (without *.application_name* in the file name) if it exists in the LSF_SERVERDIR directory. If an LSF administrator specifies one or more mandatory **eswitch** executable files using the LSB_ESWITCH_METHOD parameter in the lsf.conf file, LSF then invokes the mandatory executable files. Finally, LSF invokes any application-specific **eswitch** executable files (with *.application_name* in the file name) specified by the **bswitch -a** option.

An **eswitch** is run only once, even if it is specified by both the **bswitch -a** option and the LSB_ESWITCH_METHOD parameter.

By writing external job switch executable files, you can accept, reject, or change the destination queue for any **bswitch** request. If the **eswitch** executable file exits with the same return code as the value of the LSB_SWITCH_ABORT_VALUE environment variable, the job switch request is ignored and LSF outputs a failure message.

- [Configuration to enable job switch controls](#)
Enable job switch controls with at least one **eswitch** executable file in the directory specified by the parameter LSF_SERVERDIR in the lsf.conf file.
- [Configuration to modify job switch controls](#)
The LSB_ESWITCH_METHOD configuration parameter modifies the job switch controls behavior by defining mandatory **eswitch** programs that apply to all job switch requests.
- [Command arguments for job switch controls](#)
eswitch arguments provide flexibility for filtering and modifying job switch requests by letting you specify options for **eswitch** programs.

Configuration to enable job switch controls

Enable job switch controls with at least one **eswitch** executable file in the directory specified by the parameter LSF_SERVERDIR in the lsf.conf file.

LSF does not include a default **eswitch**; write your own programs to meet the job requirements of your site.

Executable file	UNIX naming convention	Windows naming convention
eswitch	LSF_SERVERDIR/eswitch. <i>application</i>	LSF_SERVERDIR\eswitch. <i>application</i> .exe LSF_SERVERDIR\eswitch. <i>application</i> .bat

The name of your **eswitch** indicates the application with which it runs. For example: eswitch.fluent.

Restriction: The name eswitch.user is reserved. Do not use the name eswitch.user for an application-specific **eswitch**. Valid file names contain only alphanumeric characters, underscores (_), and hyphens (-).

Once the LSF_SERVERDIR contains one or more **eswitch** executable files, users can specify the **eswitch** programs that are associated with each job that they switch.

Environment variables used by eswitch

When you write an **eswitch**, you can use the following environment variables provided by LSF for the **eswitch** execution environment:

LSB_SWITCH_PARM_FILE

Points to a temporary file that LSF uses to store the **bswitch** options entered in the command line. An **eswitch** reads this file at the job **bswitch** level and either accepts or changes the values.

An **eswitch** can change the target queue name by writing to the file that is specified by the `LSB_SWITCH_MODIFY_FILE` environment variable.

The temporary file pointed to by `LSB_SWITCH_PARM_FILE` stores the following information:

Option	bswitch option	Description
LSB_SWITCH		If set to <code>Y</code> , indicates that this is a bswitch request.
LSB_SWITCH_ADDITIONAL	<code>-a</code>	The name of the eswitch script.
LSB_SWITCH_JOB	bswitch job ID command argument	The ID of the job, job array, or job array elements.
LSB_SWITCH_QUEUE	bswitch queue name command argument	The name of the target queue.

LSB_SWITCH_MODIFY_FILE

Points to the file that the **eswitch** uses to modify the **bswitch** target queue values.

When your **eswitch** runs, LSF checks the `LSB_SWITCH_MODIFY_FILE` and applies changes so that the job switches to the revised queue.

LSB_SWITCH_ABORT_VALUE

Indicates to LSF that a job switch request should be rejected. For example, if you want LSF to reject a job switch request, your **eswitch** must contain the line

```
exit $LSB_SWITCH_ABORT_VALUE
```

Configuration to modify job switch controls

The `LSB_ESWITCH_METHOD` configuration parameter modifies the job switch controls behavior by defining mandatory **eswitch** programs that apply to all job switch requests.

Configuration to define a mandatory eswitch

Configuration file	Parameter and syntax	Behavior
lsf.conf	<code>LSB_ESWITCH_METHOD="application_name [application_name] ..."</code>	<ul style="list-style-type: none">The specified eswitch run for all job switch requests.For example, to specify a mandatory eswitch named <code>eswitch.fluent</code>, define <code>LSB_ESWITCH_METHOD=fluent</code>

Command arguments for job switch controls

eswitch arguments provide flexibility for filtering and modifying job switch requests by letting you specify options for **eswitch** programs.

The **bswitch -a** command option supports arguments for a given **eswitch**. Users can customize their **eswitch** applications, put them under `LSF_SERVERDIR`, and then submit jobs as `bswitch -a "application_name" queue_name job_id`.

The **bswitch -a** option functions the same as the **bsub -a** option, except that it controls **eswitch** files instead of **esub/epsub** files.

Share resources with application profiles

Application profiles improve the management of applications by separating scheduling policies (for example, job preemption and fair share scheduling) from application-level requirements, such as pre-execution and post-execution commands, resource limits, or job controls.

- [Manage application profiles](#)
Use application profiles to map common execution requirements to application-specific job containers. Add, remove, and set default application profiles.
- [Submitting jobs to application profiles](#)
- [How application profiles interact with queue and job parameters](#)
Application profiles operate in conjunction with queue and job-level options. In general, you use application profile definitions to refine queue-level settings, or to exclude some jobs from queue-level parameters.

Manage application profiles

Use application profiles to map common execution requirements to application-specific job containers. Add, remove, and set default application profiles.

For example, you can define different job types according to the properties of the applications that you use; your FLUENT jobs can have different execution requirements from your CATIA jobs, but they can all be submitted to the same queue.

The following application profile defines the execution requirements for the FLUENT application:

```
Begin Application
NAME          = fluent
DESCRIPTION   = FLUENT Version 6.2
CPULIMIT      = 180/hostA      # 3 hours of host hostA
FILELIMIT     = 20000
DATALIMIT     = 20000          # jobs data segment limit
CORELIMIT     = 20000
TASKLIMIT     = 5              # job processor limit
PRE_EXEC      = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
REQUEUE_EXIT_VALUES = 55 34 78
End Application
```

See the `lsb.applications` template file for additional application profile examples.

Tip: To use application profiles in an LSF multicluster environment, configure the application profile on both the submission and execution clusters. The application profile option that is specified for a job will be forwarded with the job. If the application profile does not exist in the execution cluster, the job pends with a special reason.

Use the **bclusters -app** command to see the application profiles configured in a remote cluster.

- [Adding an application profile to the `lsb.applications` file](#)
Add new application profile definitions to the `lsb.applications` file.

Adding an application profile to the `lsb.applications` file

Add new application profile definitions to the `lsb.applications` file.

Procedure

1. Log in as the LSF administrator on any host in the cluster.
2. Edit the `lsb.applications` file to add the new application profile definition.
You can copy another application profile definition from this file as a starting point.
Remember: Change the name of the copied profile in the `NAME` parameter.
3. Save the changes to the `lsb.applications` file.
4. Run the **admin reconfig** command to reconfigure the `mbatchd` daemon.

Results

Adding an application profile does not affect pending or running jobs.

Remove an application profile

Remove application profile definitions from the `lsb.applications` file.

Before you begin

Before you remove an application profile, make sure that no pending jobs are associated with the application profile.

About this task

If jobs are in the application profile, use the **bmod -app** command to move pending jobs to another application profile, then remove the application profile. Running jobs are not affected by removing the application profile associated with them.

Restriction: You cannot remove a default application profile.

Procedure

1. Log in as the LSF administrator on any host in the cluster.
2. Run the **bmod -app** command to move all pending jobs into another application profile.
If you leave pending jobs associated with an application profile that has been removed, they remain pending with the following pending reason:

`Specified application profile does not exist`

3. Edit the `lsb.applications` file and delete or comment out the definition for the application profile you want to remove.
4. Save the changes to the `lsb.applications` file.
5. Run the **badadmin reconfig** command to reconfigure the **mbatchd** daemon.

Define a default application profile

Set the `DEFAULT_APPLICATION` parameter in the `lsb.params` file to define a default application profile that is used when a job is submitted without specifying an application profile.

Procedure

1. Log in as the LSF administrator on any host in the cluster.
2. Specify the name of the default application profile in the `DEFAULT_APPLICATION` parameter in the `lsb.params` file.

`DEFAULT_APPLICATION=catia`

3. Save the changes to the `lsb.params` file.
4. Run the **badadmin reconfig** command to reconfigure the `mbatchd` daemon.

Submitting jobs to application profiles

About this task

Use the **-app** option of **bsub** to specify an application profile for the job.

Procedure

Run **bsub -app** to submit jobs to an application profile.

`bsub -app fluent -q overnight myjob`

LSF rejects the job if the specified application profile does not exist.

Modifying the application profile associated with a job

Before you begin

You can only modify the application profile for pending jobs.

Procedure

Run **bmod -app** *application_profile_name* to modify the application profile of the job.

The -appn option dissociates the specified job from its application profile. If the application profile does not exist, the job is not modified

Example

```
bmod -app fluent 2308
```

Associates job 2308 with the application profile *fluent*.

```
bmod -appn 2308
```

Dissociates job 2308 from the application profile *fluent*.

Controlling jobs associated with application profiles

About this task

bstop, **bresume**, and **bkill** operate on jobs associated with the specified application profile. You must specify an existing application profile. If *job_ID* or 0 is not specified, only the most recently submitted qualifying job is operated on.

Procedure

1. Run **bstop -app** to suspend jobs in an application profile.

```
bstop -app fluent 2280
```

Suspends job 2280 associated with the application profile *fluent*.

```
bstop -app fluent 0
```

Suspends all jobs that are associated with the application profile *fluent*.

2. Run **bresume -app** to resume jobs in an application profile.

```
bresume -app fluent 2280
```

Resumes job 2280 associated with the application profile *fluent*.

3. Run **bkill -app** to kill jobs in an application profile.

```
bkill -app fluent
```

Kills the most recently submitted job that is associated with the application profile *fluent* for the current user.

```
bkill -app fluent 0
```

Kills all jobs that are associated with the application profile *fluent* for the current user.

How application profiles interact with queue and job parameters

Application profiles operate in conjunction with queue and job-level options. In general, you use application profile definitions to refine queue-level settings, or to exclude some jobs from queue-level parameters.

- [Application profile settings that override queue settings](#)
The following application profile parameters override the corresponding queue setting:
- [Application profile limits and queue limits](#)
The following application profile limits override the corresponding queue-level soft limits:
- [Define application-specific environment variables](#)
You can use application profiles to pass application-specific tuning and runtime parameters to the application by defining application-specific environment variables. Once an environment variable is set, it applies for each job that uses the same application profile. This provides a simple way of extending application profiles to include additional information.
- [Absolute run limits](#)
If you want the scheduler to treat any run limits as absolute, define `ABS_RUNLIMIT=Y` in `lsb.params` or in `lsb.applications` for the application profile that is associated with your job. When `ABS_RUNLIMIT=Y` is defined in `lsb.params` or in the application profile, the run time limit is not normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run limit configured.
- [Pre-execution](#)
Queue-level pre-execution commands run *before* application-level pre-execution commands. Job level pre-execution commands (**bsub -E**) override application-level pre-execution commands.
- [Post-execution](#)
When a job finishes, post-execution commands run. For the order in which these commands run, refer to the section on Pre-Execution and Post-Execution Processing.
- [Re-runnable jobs](#)
`RERUNNABLE` in an application profile overrides queue-level job rerun, and allows you to submit re-runnable jobs to a non re-runnable queue. Job-level rerun (**bsub -r** or **bsub -rn**) overrides both the application profile and the queue.
- [Resource requirements](#)
Application-level resource requirements can be simple (one requirement for all slots) or compound (different requirements for specified numbers of slots). When resource requirements are set at the application-level as well as the job-level or queue-level, the requirements are combined in different ways depending on whether they are simple or compound.
- [Estimated job run time and runtime limits](#)
Instead of specifying an explicit runtime limit for jobs, you can specify an *estimated run time* for jobs. LSF uses the estimated value for job scheduling purposes only, and does not kill jobs that exceed this value unless the jobs also exceed a defined runtime limit.

Application profile settings that override queue settings

The following application profile parameters override the corresponding queue setting:

- `CHKPNT_DIR`—overrides queue `CHKPNT=chkpnt_dir`
- `CHKPNT_PERIOD`—overrides queue `CHKPNT=chkpnt_period`
- `GPU_REQ`
- `JOB_STARTER`
- `LOCAL_MAX_PREEEXEC_RETRY`
- `LOCAL_MAX_PREEEXEC_RETRY_ACTION`
- `MAX_JOB_PREEMPT`
- `MAX_JOB_REQUEUE`
- `MAX_PREEEXEC_RETRY`
- `MAX_TOTAL_TIME_PREEMPT`
- `MIG`
- `NICE`
- `NO_PREEMPT_INTERVAL`
- `REMOTE_MAX_PREEEXEC_RETRY`
- `REQUEUE_EXIT_VALUES`
- `RESUME_CONTROL`—overrides queue `JOB_CONTROLS`
- `SUSPEND_CONTROL`—overrides queue `JOB_CONTROLS`
- `TERMINATE_CONTROL`—overrides queue `JOB_CONTROLS`

Application profile limits and queue limits

The following application profile limits override the corresponding queue-level soft limits:

- CORELIMIT
- CPULIMIT
- DATALIMIT
- FILELIMIT
- MEMLIMIT
- PROCESSLIMIT
- RUNLIMIT
- STACKLIMIT
- SWAPLIMIT
- THREADLIMIT

Job-level limits can override the application profile limits. The application profile limits cannot override queue-level hard limits.

Define application-specific environment variables

You can use application profiles to pass application-specific tuning and runtime parameters to the application by defining application-specific environment variables. Once an environment variable is set, it applies for each job that uses the same application profile. This provides a simple way of extending application profiles to include additional information.

Environment variables can also be used with MPI to pass application-specific tuning or runtime parameters to MPI jobs. For example, when using a specific MPI version and trying to get the best performance for Abaqus, you need to turn on specific flags and settings which must be in both the **mpirun** command line and in the Abaqus launcher. Both **mpirun** and Abaqus allow you to define switches and options within an environment variable, so you can set both of these in the application profile and they are used automatically.

To set your own environment variables for each application, use the **ENV_VARS** parameter in **lsb.applications**. The value for **ENV_VARS** also applies to the job's pre-execution and post-execution environment. For example, a license key can be accessed by passing the license key location to the job.

To use **ENV_VARS** in an application profile:

1. Configure the **ENV_VARS** parameter in **lsb.applications**.
2. Run **badmin reconfig** to have the changes take effect.
3. Optional: Run **bapp -l** to verify that the application is created and the variables are set:

```
bapp -l myapp
APPLICATION NAME: myapp
-- Test abc, solution 123
STATISTICS:
  NJOBS    PEND    RUN    SSUSP    USUSP    RSV
    0         0      0        0        0      0
PARAMETERS:
ENV_VARS: "TEST_FRUIT='apple',TEST_CAR='civic'"
```

4. Submit your job to the application.

```
admin@hostA: bsub -I -app myapp 'echo $TEST_FRUIT'
Job <316> is submitted to default queue <interactive>
<<Waiting for dispatch...>>
<<Starting on hostA>>
apple
```

When changing the value for ENV_VARS, note the following:

- Once the job is running, you cannot change the defined values for any of the variables. However, you can still change them while the job is in PEND state.
- If you change the value for ENV_VARS before a checkpointed job resumes but after the initial job has run, then the job will use the new value for ENV_VARS.
- If you change the value for ENV_VARS then requeue a running job, the job will use the new value for ENV_VARS during the next run.
- Any variable set in the user's environment will overwrite the value in ENV_VARS. The application profile value will overwrite the execution host environment value.
- If the same environment variable is named multiple times in ENV_VARS and given different values, the last value in the list will be the one which takes effect.
- Do not redefine existing LSF environment variables in ENV_VARS.
- [Task limits](#)
TASKLIMIT in an application profile specifies the maximum number of tasks that can be allocated to a job. For parallel jobs, TASKLIMIT is the maximum number of tasks that can be allocated to the job.

Task limits

TASKLIMIT in an application profile specifies the maximum number of tasks that can be allocated to a job. For parallel jobs, TASKLIMIT is the maximum number of tasks that can be allocated to the job.

You can optionally specify the minimum and default number of tasks. All limits must be positive integers greater than or equal to 1 that satisfy the following relationship:

$$1 \leq \text{minimum} \leq \text{default} \leq \text{maximum}$$

Job-level tasks limits (**bsub -n**) override application-level TASKLIMIT, which overrides queue-level TASKLIMIT. Job-level limits must fall within the maximum and minimum limits of the application profile and the queue.

Absolute run limits

If you want the scheduler to treat any run limits as absolute, define ABS_RUNLIMIT=Y in lsb.params or in lsb.applications for the application profile that is associated with your job. When ABS_RUNLIMIT=Y is defined in lsb.params or in the application profile, the run time limit is not normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run limit configured.

Pre-execution

Queue-level pre-execution commands run *before* application-level pre-execution commands. Job level pre-execution commands (**bsub -E**) override application-level pre-execution commands.

Post-execution

When a job finishes, post-execution commands run. For the order in which these commands run, refer to the section on Pre-Execution and Post-Execution Processing.

If both application-level and job-level job-based post-execution commands (**bsub -Ep**) are specified, job level post-execution overrides application-level post-execution commands. Only the first host is over-ridden. Application level host-based post execution commands are not overwritten by -Ep.

Re-runnable jobs

RERUNNABLE in an application profile overrides queue-level job rerun, and allows you to submit re-runnable jobs to a non re-runnable queue. Job-level rerun (**bsub -r** or **bsub -rn**) overrides both the application profile and the queue.

Resource requirements

Application-level resource requirements can be simple (one requirement for all slots) or compound (different requirements for specified numbers of slots). When resource requirements are set at the application-level as well as the job-level or queue-level, the requirements are combined in different ways depending on whether they are simple or compound.

Simple job-level, application-level, and queue-level resource requirements are merged in the following manner:

- If resource requirements are not defined at the application level, simple job-level and simple queue-level resource requirements are merged.
- When simple application-level resource requirements are defined, simple job-level requirements usually take precedence. Specifically:

Section	Simple resource requirement multi-level behavior
select	All levels satisfied
same	All levels combined
order span	Job-level section overwrites application-level section, which overwrites queue-level section (if a given level is present)
cu	
rusage	All levels merge If conflicts occur the job-level section overwrites the application-level section, which overwrites the queue-level section.
affinity	Job-level section overwrites application-level section, which overwrites queue-level section (if a given level is present)

Compound application-level resource requirements are merged in the following manner:

- When a compound resource requirement is set at the application level, it will be ignored if any job-level resource requirements (simple or compound) are defined.
- In the event no job-level resource requirements are set, the compound application-level requirements interact with queue-level resource requirement strings in the following ways:
- - If no queue-level resource requirement is defined or a compound queue-level resource requirement is defined, the compound application-level requirement is used.
 - If a simple queue-level requirement is defined, the application-level and queue-level requirements combine as follows:

Section	Compound application and simple queue behavior
---------	--

Section	Compound application and simple queue behavior
select	Both levels satisfied; queue requirement applies to all compound terms
same	Queue level ignored
order	Application-level section overwrites queue-level section (if a given level is present); queue requirement (if used) applies to all compound terms
span	
cu	
rusage	<ul style="list-style-type: none"> Both levels merge Queue requirement if a job-based resource is applied to the first compound term, otherwise applies to all compound terms If conflicts occur the application-level section overwrites the queue-level section. <p>For example: if the application-level requirement is $\text{num1} * \{\text{rusage}[\text{R1}]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$ and the queue-level requirement is $\text{rusage}[\text{RQ}]$ where RQ is a job resource, the merged requirement is $\text{num1} * \{\text{rusage}[\text{merge}(\text{R1}, \text{RQ})]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$</p>
affinity	Job-level section overwrites application-level section, which overwrites queue-level section (if a given level is present)

For internal load indices and duration, jobs are rejected if they specify resource reservation requirements at the job level or application level that exceed the requirements specified in the queue.

If RES_REQ is defined at the queue level and there are no load thresholds that are defined, the pending reasons for each individual load index will not be displayed by **bjobs**.

Resource requirement strings in select sections must conform to a more strict syntax. The strict resource requirement syntax only applies to the select section. It does not apply to the other resource requirement sections (order, rusage, same, span, or cu). LSF rejects resource requirement strings where an rusage section contains a non-consumable resource.

When the parameter RESRSV_LIMIT in lsb.queues is set, the merged application-level and job-level rusage consumable resource requirements must satisfy any limits set by RESRSV_LIMIT, or the job will be rejected.

Estimated job run time and runtime limits

Instead of specifying an explicit runtime limit for jobs, you can specify an *estimated run time* for jobs. LSF uses the estimated value for job scheduling purposes only, and does not kill jobs that exceed this value unless the jobs also exceed a defined runtime limit.

The format of runtime estimate is same as the run limit set by the bsub -W option or the RUNLIMIT parameter in the lsb.queues and lsb.applications file.

Use the JOB_RUNLIMIT_RATIO parameter in the lsb.params file to limit the runtime estimate users can set. If the JOB_RUNLIMIT_RATIO=0 parameter is set, no restriction is applied to the runtime estimate. The ratio does not apply to the RUNTIME parameter in the lsb.applications file.

The job-level runtime estimate setting overrides the RUNTIME setting in an application profile in the lsb.applications file.

The following LSF features use the estimated runtime value to schedule jobs:

- Job chunking
- Advance reservation
- SLA
- Slot reservation
- Backfill

Define a runtime estimate

Define the **RUNTIME** parameter at the application level. Use the **bsub -We** option at the job-level.

You can specify the runtime estimate as hours and minutes, or minutes only. The following examples show an application-level runtime estimate of 3 hours and 30 minutes:

- `RUNTIME=3:30`
- `RUNTIME=210`

Configure normalized run time

LSF uses normalized run time for scheduling to account for different processing speeds of the execution hosts.

Tip:

If you want the scheduler to use wall-clock (absolute) run time instead of normalized run time, define the `ABS_RUNLIMIT=Y` parameter in the `lsb.params` or the `lsb.applications` file for the queue or application that is associated with your job.

LSF calculates the normalized run time by using the following formula:

$$\text{NORMALIZED_RUN_TIME} = \text{RUNTIME} * \text{CPU_Factor_Normalization_Host} / \text{CPU_Factor_Execute_Host}$$

You can specify a host name or host model with the runtime estimate so that LSF uses a specific host name or model as the normalization host. If you do not specify a host name or host model, LSF uses the CPU factor for the default normalization host as described in the following table.

Parameter defined	File	Result
DEFAULT_HOST_SPEC		LSF selects the default normalization host for the queue.
DEFAULT_HOST_SPEC	lsb.params	LSF selects the default normalization host for the cluster.
No default host at either the queue or cluster level		LSF selects the submission host as the normalization host.

To specify a host name (defined in `lsf.cluster.clustname`) or host model (defined in the `lsf.shared` file) as the normalization host, insert the slash (/) character between the minutes value and the host name or model, as shown in the following examples:

```
RUNTIME=3:30/hostA
bsub -We 3:30/hostA
```

LSF calculates the normalized run time by using the CPU factor that is defined for `hostA`.

```
RUNTIME=210/Ultra5S
bsub -We 210/Ultra5S
```

LSF calculates the normalized run time by using the CPU factor that is defined for host model `Ultra5S`.

Tip:

Use the **lsinfo** command to see host name and host model information.

Guidelines for defining a runtime estimate

1. You can define an estimated run time, along with a runtime limit (at job level with the **bsub -W** command, at application level with the `RUNLIMIT` in the `lsb.applicationsfile`, or at queue level with the `RUNLIMIT` parameter in the `lsb.queues` file).
2. If the runtime limit is defined, the job-level (-We) or application-level `RUNTIME` value must be less than or equal to the run limit. LSF ignores the estimated runtime value and uses the run limit value for scheduling in either of the following situations:
 - The estimated runtime value exceeds the run limit value
 - An estimated runtime value is not definedNote: When LSF uses the run limit value for scheduling, and the run limit is defined at more than one level, LSF uses the smallest run limit value to estimate the job duration.

How estimated run time interacts with run limits

The following table includes all the expected behaviors for the combinations of job-level runtime estimate (-We), job-level run limit (-W), application-level runtime estimate (RUNTIME), application-level run limit (RUNLIMIT), queue-level run limit (RUNLIMIT, both default and hard limit). *Ratio* is the value of JOB_RUNLIMIT_RATIO parameter that is defined in the lsb.params file. The dash (–) indicates that no value is defined for the job.

Job-runtime estimate	Job-run limit	Application runtime estimate	Application run limit	Queue default run limit	Queue hard run limit	Result
T1	–	–	–	–	–	Job is accepted Jobs running longer than $T1 * ratio$ are killed
T1	$T2 > T1 * ratio$	–	–	–	–	Job is rejected
T1	$T2 \leq T1 * ratio$	–	–	–	–	Job is accepted Jobs running longer than T2 are killed
T1	$T2 \leq T1 * ratio$	T3	T4	–	–	Job is accepted Jobs running longer than T2 are killed T2 overrides T4 or $T1 * ratio$ overrides T4 T1 overrides T3
T1	$T2 \leq T1 * ratio$	–	–	T5	T6	Job is accepted Jobs running longer than T2 are killed If $T2 > T6$, the job is rejected
T1	–	T3	T4	–	–	<ul style="list-style-type: none"> • Job is accepted • Jobs running longer than $T1 * ratio$ are killed • T2 overrides T4 or $T1 * ratio$ overrides T4 • T1 overrides T3

Job-runtime estimate	Job-run limit	Application runtime estimate	Application run limit	Queue default run limit	Queue hard run limit	Result
T1	—	—	—	T5	T6	<ul style="list-style-type: none"> • Job is accepted • Jobs running longer than $T1 * ratio$ are killed • If $T1 * ratio > T6$, the job is rejected

Plan-based scheduling and reservations

Plan-based scheduling greatly improves the original scheduling and reservation features in LSF. Instead of looking only at current resource availability, the scheduler can plan job placements for the near future. Reservations are then based on these planned allocation. Plan-based scheduling is meant to be a replacement for legacy LSF reservation policies. When ALLOCATION_PLANNER is enabled, parameters related to the old reservation features are ignored.

- [Enabling plan-based scheduling and reservations](#)
Enabling plan-based scheduling and reservations involves the parameters ALLOCATION_PLANNER and PLAN = Y. When plan-based scheduling is enabled, parameters related to the old reservation features are ignored.
- [Plan-based allocation](#)
Plan-based scheduling and reservations requires an allocation plan.
- [Plan-based scheduling run time](#)
Plan-based scheduling provides an estimated run time for jobs when configured.
- [Plan-based scheduling limits and prioritization](#)
Plan-based scheduling follows specific limits and prioritization rules for LSF.
- [Reserving resources for an allocation plan](#)
In order to enact the current allocation plan, LSF uses the existing reservation mechanism to hold resources idle as needed for plans. As a general principle, LSF will try to reserve as few resources as possible in order to enact the plan.
- [Canceling planned allocations](#)
In an ideal world, once a planned allocation is created it will live until the job is dispatched on the resources of the planned allocation. However, in cases where LSF detects that the planned allocation has become (or may have become) invalid, the planned allocation is canceled.
- [Delaying planning for jobs](#)
In high throughput environments, it may be desirable for most jobs to use the standard LSF scheduling, without a plan. The planning based reservation can be used only to help in cases of job starvation (that is, if it is pending for longer than expected).
- [Limiting the number of planned jobs](#)
When a reservation is enabled, this has the potential to decrease cluster utilization since LSF holds resources idle in order to avoid job starvation. For this reason, some sites may wish to limit the number of jobs allowed in an allocation plan.
- [Adjusting the plan window](#)
The default plan-based scheduling window is 1 hour. The ALLOCATION_PLANNER parameter allows you to adjust this window.

Enabling plan-based scheduling and reservations

Enabling plan-based scheduling and reservations involves the parameters `ALLOCATION_PLANNER` and `PLAN = Y`. When plan-based scheduling is enabled, parameters related to the old reservation features are ignored.

Enabling planning based scheduling

By default, planning based scheduling is disabled. To enable, in the `lsb.params` file, set the `ALLOCATION_PLANNER` parameter to `Y`.

By itself, this parameter does not have any effect on scheduling behavior. You need to separately identify sets of candidate jobs to consider in the planner.

When planning based scheduled is enabled, it is recommended to enable the parameter `BACKFILL=Y` in the `lsb.queues` file.

Identifying jobs as candidates for planned reservations

To identify sets of jobs as candidates that should be considered by the scheduling planner, configure the parameter `PLAN = Y` at the application level (`lsb.applications` file), at the queue level (`lsb.queues` file), or cluster level (`lsb.params` file).

When the parameter is configured at multiple levels, the application-level setting overrides the queue-level setting, which overrides the cluster-level setting.

For jobs identified as candidates by this parameter, LSF will consider creation of an allocation plan. LSF will then reserve for jobs as needed in order to execute the plan.

Plan-based allocation

Plan-based scheduling and reservations requires an allocation plan.

Viewing the allocation plan

Whenever the allocation plan is updated, the planner process dumps a snapshot of the allocation plan in JSON format to the LSF working directory, located at `$LSF_ENVDIR/./work/<clustername>/logdir/lsb.allocation.plan.json`.

For example, the allocation plan with plans for two jobs would appear like the following:

```
> cat $LSF_ENVDIR/./work/montreal/logdir/lsb.allocation.plan.json
{
  "TIMESTAMP": "Mar 29 15:07:50 2018",
  "CHILD_PID": "18939",
  "NUM_PLANS": "2",
  "PLANS": [
    {
      "JOB_ID": "montreal@172024",
      "RANK": "1",
      "START_TIME": "1522351070",
      "START_TIME_H": "Mar 29 15:17:50 2018",
      "FINISH_TIME": "1522351670",
      "FINISH_TIME_H": "Mar 29 15:27:50 2018",
      "ALLOC": [
        {
          "HOST": "host0",
          "RSRC": [
            {
              "NAME": "slots",
              "AMOUNT": 23
            }
          ]
        }
      ]
    },
    {
      "JOB_ID": "montreal@172025",
```

```

"RANK": "2",
"START_TIME": "1522351752",
"START_TIME_H": "Mar 29 15:29:12 2018",
"FINISH_TIME": "1522352352",
"FINISH_TIME_H": "Mar 29 15:39:12 2018",
"ALLOC": [
  {
    "HOST": "host0",
    "RSRC": [
      {
        "NAME": "slots",
        "AMOUNT": 23
      }
    ]
  }
]
}
]
}
}

```

Viewing the allocation plan for a job

Users can view the allocation plan for individual jobs using the **bjobs** command with the **-plan** option. Use **bjobs -plan** to display PEND jobs with allocation plans.

The **bjobs -l** command also displays allocation plan:

- **bjobs -l** : display the planned start time for all jobs with a plan.
- **bjobs -l -plan** : filter for jobs with plans, displaying the planned start time and planned allocation for each job.

Example: Filtering for jobs with a plan:

```

username1@intel4-74: bjobs -plan
JOBID  USER      STAT  QUEUE   FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
172024 username1 PEND  normal  intel4     x           Mar 29 15:07
172025 username1 PEND  normal  intel4     x           Mar 29 15:07

```

Example: Viewing the planned start time for a job:

```

username1@intel4-71: bjobs -l 172024

```

```

Job <172024>, User <uesrname1>, Project <default>, Status <PEND>, Queue <normal>
, Job Priority <500>, Command <x>
Thu Mar 29 15:07:26: Submitted from host <intel4>, CWD <$HOME>, 23 Task(s), Spe
cified Hosts <host0>;

```

ESTIMATED_RUNTIME

10.0 min of intel4 based on user provided runtime estimation.

PENDING REASONS:

Not enough processors to meet the job's spanning requirement: 1 host;

Not specified in job submission: 4607 hosts;

Closed by LSF administrator: 1 host;

SCHEDULING PARAMETERS:

	r15s	rlm	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

RESOURCE REQUIREMENT DETAILS:

Combined: select[type == any] order[r15s:pg]

Effective: -

PLANNED ALLOCATION:

```

Rank: 1
Start Time: Mar 29 15:21:42 2018
Finish Time: Mar 29 15:31:42 2018

```

Example: Viewing the planned start time and planned allocation for a job:

```

username1@intel4-73: bjobs -l -plan 172024

```

```

Job <172024>, User <username1>, Project <default>, Status <PEND>, Queue <normal>
, Job Priority <500>, Command <x>
Thu Mar 29 15:07:26: Submitted from host <intel4>, CWD <$HOME>, 23 Task(s), Spe
cified Hosts <host0>;

ESTIMATED_RUNTIME
10.0 min of intel4 based on user provided runtime estimation.
PENDING REASONS:
Not enough processors to meet the job's spanning requirement: 1 host;
Not specified in job submission: 4607 hosts;
Closed by LSF administrator: 1 host;

SCHEDULING PARAMETERS:
      r15s  rlm  r15m  ut      pg    io    ls    it    tmp    swp    mem
loadSched  -    -    -    -      -    -    -    -    -    -    -
loadStop   -    -    -    -      -    -    -    -    -    -    -

RESOURCE REQUIREMENT DETAILS:
Combined: select[type == any] order[r15s:pg]
Effective: -

PLANNED ALLOCATION:
Rank:      1
Start Time: Mar 29 15:22:15 2018
Finish Time: Mar 29 15:32:15 2018
Resources:
  host0
    slots: 23

```

Plan-based scheduling run time

Plan-based scheduling provides an estimated run time for jobs when configured.

Estimating run time of jobs

To make allocation plans, LSF needs estimated run times for all (or most) jobs. To simplify this, the parameter `ESTIMATED_RUNTIME` can be used to have default estimated run times for classes of jobs.

The `ESTIMATED_RUNTIME` parameter can be specified at the application level (`lsb.applications`), the queue level (`lsb.queue`s), or the cluster level (`lsb.params`). When configured at multiple levels the application-level setting overrides the queue-level setting, which overrides the cluster-level setting.

Alternatively, a job level estimated runtime can be set for a job with the **`bsub -We`** command. The job level setting overrides the configured settings for estimated runtime.

Note: The `ESTIMATED_RUNTIME` parameter replaces the `RUNTIME` in `lsb.applications`.

If a job does not have an explicit estimated runtime, then the scheduler will estimate the runtime based on the run limit, CPU limit, and termination time. When a job has none of these, the run times is assumed to be infinite.

Viewing the estimated run time of a job

The **`bjobs -l`** command displays the estimated run time of a job with the `ESTIMATED_RUNTIME` section in the output. This output displays the estimated run time, and also how this estimated run time is derived.

Example: Viewing the estimated run time of a job:

```

username1@intel4-79: bjobs -l 172025

Job <172025>, User <username1>, Project <default>, Status <PEND>, Queue<normal>
, Job Priority <500>, Command <x>
Thu Mar 29 15:07:27: Submitted from host <intel4>, CWD <$HOME>, 23 Task(s), Spe
cified Hosts <host0>;

ESTIMATED_RUNTIME:
10.0 min of intel4 based on user provided runtime estimation.

```

PENDING REASONS:

Not enough processors to meet the job's spanning requirement: 1 host;

Not specified in job submission: 4607 hosts;

Closed by LSF administrator: 1 host;

...

Plan-based scheduling limits and prioritization

Plan-based scheduling follows specific limits and prioritization rules for LSF.

Limits and the allocation plan

The planner maintains an allocation plan for jobs into the near future. As a general principle the planner should obey resource constraints and configured limits for each point in time.

For example, if there is a limit configured of 20 jobs for a particular user, then in any plan generated by LSF the number of jobs concurrently running in the plan at any point in time must be 20 or less.

Job prioritization

For jobs without a planned allocation, the jobs are scheduled using the standard LSF job prioritization policies in both the main scheduler (parent mbschd) as well as the planner (child mbschd) process.

As usual, LSF considers jobs for dispatch queue-by-queue. When LSF schedules the jobs in a queue, it will first try to dispatch jobs with planned allocations on their planned. After those jobs are completed, LSF will then enter its standard scheduling flow and will consider jobs without planned allocations.

For two jobs with a planned allocation and within the same queue, LSF orders them by *rank*. The rank of a planned allocation indicates the order in which it was created, relative to other planned allocations from the same queue.

When a new planned allocation is created by the planner, it is assigned a rank larger than all other planned allocations from the same queue. The rank of a particular planned allocation can change from plan to plan, but the relative orders between pairs is preserved.

The rank of a job's future allocation can be viewed using the **bjobs -l** command.

- [Configuring extendable run limits](#)

A job with an extendable run limit is allowed to continue running unless the resources that are occupied by the job are needed by another job in a queue with the same or higher priority.

Configuring extendable run limits

A job with an extendable run limit is allowed to continue running unless the resources that are occupied by the job are needed by another job in a queue with the same or higher priority.

Before you begin

Ensure that `ALLOCATION_PLANNER=Y` is defined in the `lsb.params` file to enable the allocation planner.

About this task

You can configure the LSF allocation planner to extend the run limits of a job by changing its soft run limit. A soft run limit can be extended, while a hard run limit cannot be extended. The allocation planner looks at job plans to determine if there are any other jobs that require the current job's resources.

Procedure

1. Edit `lsb.queues`.
2. Specify the `EXTENDABLE_RUNLIMIT` parameter for the queue and specify the base limit and other keywords for the run limit.

```
EXTENDABLE_RUNLIMIT=BASE[minutes] INCREMENT[minutes] GRACE[minutes] REQUEUE[Y | N]
```

This parameter uses the following keywords:

BASE [minutes]

The initial soft run limit that is imposed on jobs in the queue. Whenever the job reaches the soft run limit, the allocation planner considers whether the resources that are held by the job are needed by another job in the queue by looking at plans for the other jobs. If the resources are not required, LSF extends the soft run limit for the current job. Otherwise, LSF sets a hard run limit.

Specify an integer value for the initial soft run limit.

INCREMENT [minutes]

If LSF decides to extend the soft run limit for the job, this keyword specifies the amount of time that LSF extends the soft run limit.

Specify an integer value for the soft run limit extension time. The default value is the value of the `BASE[]` keyword.

GRACE [minutes]

If LSF decides not to extend the soft run limit for the job, a hard run limit is set for this amount of minutes from the time the decision is made.

The default value is 0 (the job is terminated or requeued immediately).

REQUEUE [Y | N]

Specifies the action that LSF takes when a job reaches its hard run limit. If set to `N`, LSF terminates the job. If set to `Y` LSF requeues the job.

The default value is `N` (LSF terminates the job once the job reaches its hard run limit).

For example,

```
Begin Queue
QUEUE_NAME = queue_extendable
PRIORITY = 10
EXTENDABLE_RUNLIMIT = BASE[60] INCREMENT[30] GRACE[10]
End Queue
```

3. Reconfigure the cluster:
 - a. Run **lsadmin reconfig**.
 - b. Run **badmin reconfig**.
4. Run **bqueues -l** to display the extendable run limit settings.

Reserving resources for an allocation plan

In order to enact the current allocation plan, LSF uses the existing reservation mechanism to hold resources idle as needed for plans. As a general principle, LSF will try to reserve as few resources as possible in order to enact the plan.

When deciding how many resources to reserve, LSF considers not only the raw resource availability (that is, slots, memory, licenses, etc.), but also things like configured limits. For example, if a slot limit is configured for a user, and in the plan the user uses up to 100 slots concurrently, then LSF may need to reserve up to 100 slots for that user's `PEND` jobs in order to ensure that the plan can be carried out.

Due to such concerns, the amount of resources reserved on a host may exceed the capacity of the host in some cases. Reservations may also exceed configured limits.

As in LSF releases before version 10.1.0.5, the resources reserved for a job can be viewed using the **bjobs -l** command. The slots / tasks reserved on a host can be viewed using the **bhosts** command.

When a job is reserving resources for a planned allocation, the planned allocation becomes *sticky* in that LSF will not consider searching for a better (earlier) planned allocation for the job once it starts reserving. Contrast this with jobs that do not reserve. For these, LSF will continuously look in the planner to place the job where it will start earliest.

Canceling planned allocations

In an ideal world, once a planned allocation is created it will live until the job is dispatched on the resources of the planned allocation. However, in cases where LSF detects that the planned allocation has become (or may have become) invalid, the planned allocation is canceled.

When this happens, the job becomes treated like a regular job without a future allocation, but may get another future allocation in the future.

Examples of cases where a planned allocation will be canceled:

- The job is modified, switched, stopped, or killed.
- Some host of the future allocation is no longer usable. The host may have died, been closed, exceeded a load threshold, or the LSF daemons may be down.
- A host of the allocation no longer satisfies the job's requirements, such as the *select* requirement.
- The estimated start time for the planned allocation has moved outside the planning window (default is 1 hour). For example, this could happen if higher priority jobs are started on the resources of the planned allocation.

Delaying planning for jobs

In high throughput environments, it may be desirable for most jobs to use the standard LSF scheduling, without a plan. The planning based reservation can be used only to help in cases of job starvation (that is, if it is pending for longer than expected).

To support this case, the `PLAN = DELAY` parameter setting is provided at the application, queue, or cluster level:

```
PLAN = DELAY[<minutes>]
```

This causes LSF to wait for the specified number of minutes following the submission of a job, to consider making a planned allocation for that job.

In the case that eligible scheduling time is enabled, LSF waits until the job's eligible pending time has reached the delay time before it considers making a planned allocation for the job.

When delay is specified at multiple levels, the application-level setting overrides the queue-level setting, which overrides the cluster-level setting.

Limiting the number of planned jobs

When a reservation is enabled, this has the potential to decrease cluster utilization since LSF holds resources idle in order to avoid job starvation. For this reason, some sites may wish to limit the number of jobs allowed in an allocation plan.

This can be controlled with the `PLAN` parameter setting at the application, queue, or cluster level::

```
PLAN = MAX_JOBS[<num>]
```

When configured at multiple levels, the strictest of these will apply.

Adjusting the plan window

The default plan-based scheduling window is 1 hour. The `ALLOCATION_PLANNER` parameter allows you to adjust this window.

To adjust the window use the `ALLOCATION_PLANNER` parameter setting in the `lsb.params` configuration file:

```
ALLOCATION_PLANNER = WINDOW[<minutes>]
```

A larger window has the potential to cause more computational effort in the planner process, and may result in a larger number of jobs with planned allocations.

Distributing job resources to users in LSF

Learn how users can share job resources through LSF.

- [**Configure resource consumers**](#)
Learn how to configure and manage LSF user groups. Configure user group administrators and user group administrator rights. Import external user groups. Configure existing user groups as LSF user groups. Add consumers to a guaranteed resource pool.
- [**Specifying resource requirements**](#)
- [**Reserving resources**](#)
- [**Limiting job resource allocations**](#)
Resource allocation limits configured in the `lsb.resources` file restrict the maximum amount of a resource requested by a job that can be allocated during job scheduling for different classes of jobs to start. Configured limits also specify which resource consumers the limits apply to. Configure all resource allocation limits in one or more Limit sections in the `lsb.resources` file.
- [**Make sure LSF resources are distributed fairly**](#)
Use runtime resource usage limits to control how much resource can be consumed by running jobs. Configure load thresholds so that Jobs running under LSF can be automatically suspended and resumed based on the load conditions on the execution hosts. Learn about dispatch and run windows and deadline constraint scheduling.
- [**Preemptive scheduling**](#)
The preemptive scheduling feature allows a pending high-priority job to preempt a running job of lower priority. The lower-priority job is suspended and is resumed as soon as possible. Use preemptive scheduling if you have long-running, low-priority jobs causing high-priority jobs to wait an unacceptably long time.
- [**Goal-oriented SLA-driven scheduling**](#)

Configure resource consumers

Learn how to configure and manage LSF user groups. Configure user group administrators and user group administrator rights. Import external user groups. Configure existing user groups as LSF user groups. Add consumers to a guaranteed resource pool.

- [**Managing LSF user groups**](#)
Learn how to configure LSF user groups and how to configure existing system user groups as LSF user groups. Use the external host and user groups feature to maintain group definitions for your site in a location external to LSF.
- [**Job groups**](#)
A collection of jobs can be organized into job groups for easy management. A job group is a container for jobs in much the same way that a directory in a file system is a container for files. For example, a payroll application may have one group of jobs that calculates weekly payments, another job group for calculating monthly salaries, and a third job group that handles the salaries of part-time or contract employees. Users can submit, view, and control jobs according to their groups rather than looking at individual jobs.
- [**Host groups**](#)
Host groups gather similar resources to the same group of hosts (for example, all hosts with big memory)- Use host groups to manage dedicated resources for a single organization or to share resources across organizations. You can add limits to host groups, or define host groups in queues to constrain jobs for a scheduling policy that is defined over a specific set of hosts.

Managing LSF user groups

Learn how to configure LSF user groups and how to configure existing system user groups as LSF user groups. Use the external host and user groups feature to maintain group definitions for your site in a location external to LSF.

- [User groups in LSF](#)
User groups act as aliases for lists of users. Administrators can also limit the total number of running jobs belonging to a user or a group of users.
- [Existing user groups as LSF user groups](#)
User groups already defined in your operating system often reflect existing organizational relationships among users. It is natural to control computer resource access using these existing groups.
- [External host and user groups](#)
Use the external host and user groups feature to maintain group definitions for your site in a location external to LSF, and to import the group definitions on demand.
- [Creating a user group using bconf](#)

User groups in LSF

User groups act as aliases for lists of users. Administrators can also limit the total number of running jobs belonging to a user or a group of users.

- [How to define user groups](#)
You can define an LSF user group within LSF or use an external executable to retrieve user group members.
- [Where to configure user groups](#)
LSF user groups can be used in defining several configurations.
- [Configuring user groups](#)
- [Configuring user group administrators](#)
- [Import external user groups \(egroup\)](#)
When the membership of a user group changes frequently, or when the group contains a large number of members, you can use an external executable called **egroup** to retrieve a list of members rather than having to configure the group membership manually. You can write a site-specific egroup executable that retrieves user group names and the users that belong to each group.

How to define user groups

You can define an LSF user group within LSF or use an external executable to retrieve user group members.

User groups configured within LSF can have user group administrators configured, delegating responsibility for job control away from cluster administrators.

Use **bugroup** to view user groups and members, use **busers** to view all users in the cluster.

You can define user groups in LSF in several ways:

- Use existing user groups in the configuration files
- Create LSF-specific user groups
- Use an external executable to retrieve user group members

You can use all three methods, provided that the user and group names are different.

Where to configure user groups

LSF user groups can be used in defining several configurations.

The following parameters in LSF configuration files can specify user groups:

- The USERS and ADMINISTRATORS parameters in the lsb.queues file. The ADMINISTRATORS parameter is optional.
- The USER_NAME parameter for user job slot limits in the lsb.users file.
- The USER_SHARES parameter (optional) for host partitions in the lsb.hosts file or for queue fair share policies in lsb.queues file.
- The USERS parameter and the PER_USER parameter for resource limits or resource reservation in the lsb.resources file.
- The USER_GROUP parameter and the ACCESS_CONTROL parameter for SLA access in the lsb.serviceclasses file.

Note: If you are using existing OS-level user groups instead of LSF-specific user groups, you can also specify the names of these groups in the files mentioned above.

Configuring user groups

Procedure

1. Log in as the LSF administrator to any host in the cluster.
2. Open lsb.users.
3. If the **UserGroup** section does not exist, add it:

```
Begin UserGroup
GROUP_NAME      GROUP_MEMBER      USER_SHARES
financial        (user1 user2 user3)      ([user1, 4] [others, 10])
system          (all)                      ([user2, 10] [others, 15])
regular_users    (user1 user2 user3 user4) -
part_time_users (!)          -
End UserGroup
```

4. Specify the group name under the GROUP_NAME column.
External user groups must also be defined in the **egroup** executable.
5. Specify users in the GROUP_MEMBER column.
For external user groups, put an exclamation mark (!) in the GROUP_MEMBER column to tell LSF that the group members should be retrieved using egroup.

Note: If **ENFORCE_UG_TREE=Y** is defined in lsb.params, all user groups must conform to a tree-like structure, and a user group can appear in GROUP_MEMBER once at most. The second and subsequent occurrence of a user group in GROUP_MEMBER is ignored.

6. Optional: To enable hierarchical fair share, specify share assignments in the USER_SHARES column.
7. Save your changes.
8. Run **badmadmin ckconfig** to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
9. Run **badmadmin reconfig** to reconfigure the cluster.

Configuring user group administrators

About this task

By default, user group administrators can control all jobs that are submitted by users who are members of the user group.

Define **STRICT_UG_CONTROL=Y** in lsb.params to:

- Configure user group administrators for user groups with all as a member
- Limit user group administrators to controlling jobs in the user group when jobs are submitted with **bsub -G**.

Procedure

1. Log in as the LSF administrator to any host in the cluster.
2. Open `lsb.users`.
3. Edit the UserGroup section:

```
Begin UserGroup
GROUP_NAME  GROUP_MEMBER  GROUP_ADMIN
ugAdmins    (Toby Steve)      ( )
marketing    (user1 user2)      (shelley ugAdmins)
financial    (user3 user1 ugA) (john)
engineering  (all)                ( )
End UserGroup
```

4. To enable user group administrators, specify users or user groups in the GROUP_ADMIN column. Separate users and user groups with spaces, and enclose each GROUP_ADMIN entry in brackets.
5. Save your changes.
6. Run **badadmin ckconfig** to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
7. Run **badadmin reconfig** to reconfigure the cluster.

Example

For example, for the configuration shown and the default setting `STRICT_UG_CONTROL=N` in `lsb.params`, user1 submits a job:

```
bsub -G marketing job1.
```

job1 can be controlled by user group administrators for both the marketing and financial user groups since user1 is a member of both groups.

With `STRICT_UG_CONTROL=Y` defined, only the user group administrators for marketing can control job1. In addition, a user group administrator can be set for the group engineering which has all as a member.

- [Configuring user group administrator rights](#)

Configuring user group administrator rights

About this task

User group administrators with rights assigned can adjust user shares, adjust group membership, and create new user groups.

Procedure

1. Log in as the LSF administrator to any host in the cluster.
2. Open `lsb.users`.
3. Edit the UserGroup section:

```
Begin UserGroup
GROUP_NAME  GROUP_MEMBER  GROUP_ADMIN
ugAdmins    (Toby Steve)      ( )
marketing    (user1 user2)      (shelley[full] ugAdmins)
financial    (user3 ugA)      (john ugAdmins[usershares])
End UserGroup
```

4. To enable user group administrator rights, specify users or user groups in the GROUP_ADMIN column with the rights in square brackets.
 - no rights specified: user group admins can control all jobs submitted to the user group.
 - usershares: user group admins can adjust usershares using **bconf** and control all jobs submitted to the user group.

- full: user group admins can create new user groups, adjust group membership, and adjust usershares using **bconf**, as well as control all jobs submitted to the user group.
User group admins with full rights can only add a user group member to the user group if they also have full rights for the member user group.
5. Save your changes.
 6. Run **badadmin ckconfig** to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
 7. Run **badadmin reconfig** to reconfigure the cluster.

Import external user groups (egroup)

When the membership of a user group changes frequently, or when the group contains a large number of members, you can use an external executable called **egroup** to retrieve a list of members rather than having to configure the group membership manually. You can write a site-specific egroup executable that retrieves user group names and the users that belong to each group.

For information about how to use the external host and user groups feature, see [External host and user groups](#).

Existing user groups as LSF user groups

User groups already defined in your operating system often reflect existing organizational relationships among users. It is natural to control computer resource access using these existing groups.

You can specify existing UNIX user groups anywhere an LSF user group can be specified.

How LSF recognizes UNIX user groups

Only group members listed in the `/etc/group` file or the file `group.byname` NIS map are accepted. The user's primary group as defined in the `/etc/passwd` file is ignored.

The first time you specify a UNIX user group, LSF automatically creates an LSF user group with that name, and the group membership is retrieved by **getgrnam(3)** on the management host at the time **mbatchd** starts. The membership of the group might be different from the one on another host. Once the LSF user group is created, the corresponding UNIX user group might change, but the membership of the LSF user group is not updated until you reconfigure LSF (**badadmin**). To specify a UNIX user group that has the same name as a user, use a slash (/) immediately after the group name: `group_name/`.

Requirements

UNIX group definitions referenced by LSF configuration files must be uniform across all hosts in the cluster. Unexpected results can occur if the UNIX group definitions are not homogeneous across machines.

How LSF resolves users and user groups with the same name

If an individual user and a user group have the same name, LSF assumes that the name refers to the individual user. To specify the group name, append a slash (/) to the group name.

For example, if you have both a user and a group named `admin` on your system, LSF interprets `admin` as the name of the user, and `admin/` as the name of the group.

Where to use existing user groups

Existing user groups can be used in defining the following parameters in LSF configuration files:

- USERS in lsb.queues for authorized queue users
- USER_NAME in lsb.users for user job slot limits
- USER_SHARES (optional) in lsb.hosts for host partitions or in lsb.queues or lsb.users for queue fair share policies

External host and user groups

Use the external host and user groups feature to maintain group definitions for your site in a location external to LSF, and to import the group definitions on demand.

- [About external host and user groups](#)
LSF provides you with the option to configure host groups, user groups, or both. When the membership of a host or user group changes frequently, or when the group contains a large number of members, you can use an external executable called **egroup** to retrieve a list of members rather than having to configure the group membership manually. You can write a site-specific egroup executable that retrieves host or user group names and the hosts or users that belong to each group.
- [Configuration to enable external host and user groups](#)
- [External host and user groups behavior](#)
On restart and reconfiguration, **mbatchd** invokes the egroup executable to retrieve external host and user groups and then creates the groups in memory; **mbatchd** does *not* write the groups to lsb.hosts or lsb.users. The **egroup** executable runs under the same user account as **mbatchd**. By default, this is the primary cluster administrator account.
- [Between-host user Account mapping](#)
The between-host user account mapping feature enables job submission and execution within a cluster that has different user accounts assigned to different hosts. Using this feature, you can map a local user account to a different user account on a remote host.
- [Cross-cluster user account mapping](#)
The cross-cluster user account mapping feature enables cross-cluster job submission and execution for a multicluster environment that has different user accounts assigned to different hosts. Using this feature, you can map user accounts in a local cluster to user accounts in one or more remote clusters.
- [UNIX and Windows user account mapping](#)
The UNIX and Windows user account mapping feature enables cross-platform job submission and execution in a mixed UNIX and Windows environment. Using this feature, you can map Windows user accounts, which include a domain name, to UNIX user accounts, which do not include a domain name, for user accounts with the same user name on both operating systems.

About external host and user groups

LSF provides you with the option to configure host groups, user groups, or both. When the membership of a host or user group changes frequently, or when the group contains a large number of members, you can use an external executable called **egroup** to retrieve a list of members rather than having to configure the group membership manually. You can write a site-specific egroup executable that retrieves host or user group names and the hosts or users that belong to each group.

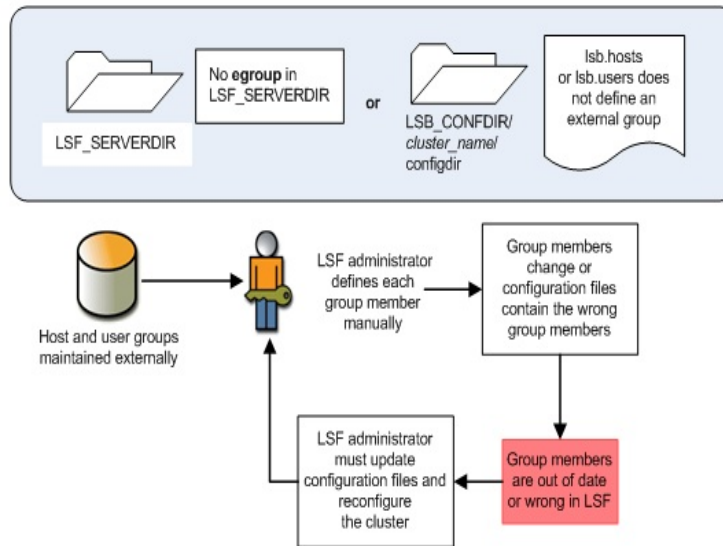
You can write your **egroup** executable to retrieve group members for:

- One or more host groups
- One or more user groups
- Any combination of host and user groups

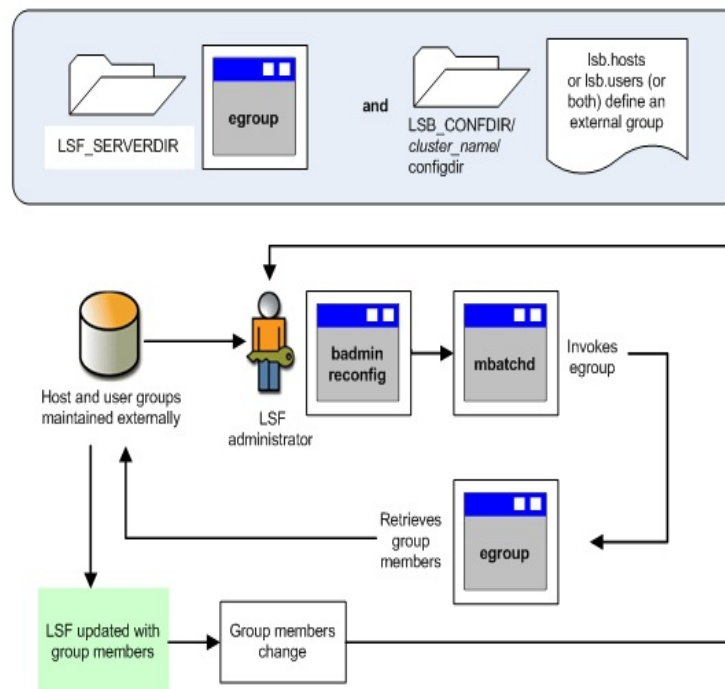
LSF does not include a default **egroup**; you should write your own executable to meet the requirements of your site.

Default behavior (feature not enabled)

The following illustrations show the benefits of using the external host and user groups feature.



With external host and user groups enabled



Scope

Applicability	Details
Operating system	<ul style="list-style-type: none">• UNIX• Windows• A mix of UNIX and Windows hosts
Dependencies	<ul style="list-style-type: none">• UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs.• The cluster must be reconfigured if you want to run the egroup executable to retrieve user group members. With a time interval specified in <code>EGROUP_UPDATE_INTERVAL</code>, egroup members can be updated automatically.
Limitations	<ul style="list-style-type: none">• The egroup executable works with static hosts only; you cannot use an egroup executable to add a dynamically added host to a host group.
Not used with	<ul style="list-style-type: none">• Host groups when you have configured EGO-enabled service-level agreement (SLA) scheduling because EGO resource groups replace LSF host groups.

Configuration to enable external host and user groups

To enable the use of external host and user groups, you must

- Define the host group in `lsb.hosts`, or the user group in `lsb.users`, and put an exclamation mark (!) in the `GROUP_MEMBER` column.
- Create an **egroup** executable in the directory specified by the environment variable `LSF_SERVERDIR` (set by `cschrc.lsf` and `profile.lsf`). LSF does not include a default **egroup**; you should write your own executable to meet the requirements of your site.
- Run the **badmadmin reconfig** command first to reconfigure the cluster, then wait for the cluster to be automatically reconfigured with the updated external user groups.
- The reconfiguration for external user groups (**egroups**) is done automatically according to the time interval you specify in `EGROUP_UPDATE_INTERVAL`.

Define an external host or user group

External host groups are defined in `lsb.hosts`, and external user groups are defined in `lsb.users`. Your **egroup** executable must define the same group names that you use in the `lsb.hosts` and `lsb.users` configuration files.

Configuration file	Parameter and syntax	Default behavior
<code>lsb.hosts</code>	<code>GROUP_NAME</code> and <code>GROUP_MEMBER <i>hostgroup_name</i> (!)</code>	<ul style="list-style-type: none">• Enables the use of an egroup executable to retrieve external host group members.• The <i>hostgroup_name</i> specified in <code>lsb.hosts</code> must correspond to the group name defined by the egroup executable.• You can configure one or more host groups to use the egroup executable.• LSF does not support the use of external host groups that contain dynamically added hosts.• You cannot set the preference level for a host group member, if its group members is part of an egroup.

Configuration file	Parameter and syntax	Default behavior
lsb.users	GROUP_NAME GROUP or GROUP_MEMBER <i>usergroup_name</i> (!)	<ul style="list-style-type: none"> Enables the use of an egroup executable to retrieve external user group members. The <i>usergroup_name</i> specified in lsb.users must correspond to the group name defined by the egroup executable. You can configure one or more user groups to use the egroup executable.

Create an egroup executable

The **egroup** executable must

- Be located in LSF_SERVERDIR and follow these naming conventions:

Operating system	Naming convention
UNIX	LSF_SERVERDIR/egroup
Windows	LSF_SERVERDIR\egroup.exe or LSF_SERVERDIR\egroup.bat

- Run when invoked by the commands **egroup -m** *hostgroup_name* and **egroup -u** *usergroup_name*. When mbatchd finds an exclamation mark (!) in the GROUP_MEMBER column of lsb.hosts or lsb.users, mbatchd runs the **egroup** command to invoke your **egroup** executable.
- Output a space-delimited list of group members (hosts, users, or both) to stdout.
- Retrieve a list of static hosts only. You cannot use the **egroup** executable to retrieve hosts that have been dynamically added to the cluster.

The following example shows a simple **egroup** script that retrieves both host and user group members:

```
#!/bin/sh
if [ "$1" = "-m" ]; then #host group
    if [ "$2" = "linux_grp" ]; then #Linux hostgroup
        echo "linux01 linux 02 linux03 linux04"
    elif [ "$2" = "sol_grp" ]; then #Solaris hostgroup
        echo "Sol02 Sol02 Sol03 Sol04"
    fi
else #user group
    if [ "$2" = "srv_grp" ]; then #srvgrp user group
        echo "userA userB userC userD"
    elif [ "$2" = "dev_grp" ]; then #devgrp user group
        echo "user1 user2 user3 user4"
    fi
fi
```

External host and user groups behavior

On restart and reconfiguration, **mbatchd** invokes the **egroup** executable to retrieve external host and user groups and then creates the groups in memory; **mbatchd** does *not* write the groups to lsb.hosts or lsb.users. The **egroup** executable runs under the same user account as **mbatchd**. By default, this is the primary cluster administrator account.

Once LSF creates the groups in memory, the external host and user groups work the same way as any other LSF host and user groups, including configuration and batch command usage.

Between-host user Account mapping

The between-host user account mapping feature enables job submission and execution within a cluster that has different user accounts assigned to different hosts. Using this feature, you can map a local user account to a different user account on a remote host.

- [About between-host user account mapping](#)
For clusters with different user accounts assigned to different hosts., between-host user account mapping allows you to submit a job from a local host and run the job as a different user on a remote host.
- [Configuration to enable between-host user account mapping](#)
Between-host user account mapping can be configured in one of the two ways.
- [Between-host user account mapping behavior](#)
- [Between-host user account mapping commands](#)

About between-host user account mapping

For clusters with different user accounts assigned to different hosts., between-host user account mapping allows you to submit a job from a local host and run the job as a different user on a remote host.

There are two types of between-host user account mapping:

- Local user account mapping: for UNIX or Windows hosts, a user can map the local user account to a different user on a remote host
- Windows workgroup account mapping: allows LSF administrators to map all Windows workgroup users to a single Windows system account, eliminating the need to create multiple users and passwords in LSF. Users can submit and run jobs using their local user names and passwords, and LSF runs the jobs using the mapped system account name and password. With Windows workgroup account mapping, all users have the same permissions because all users map to the same Windows system account.

Figure 1. Default behavior (feature not enabled)

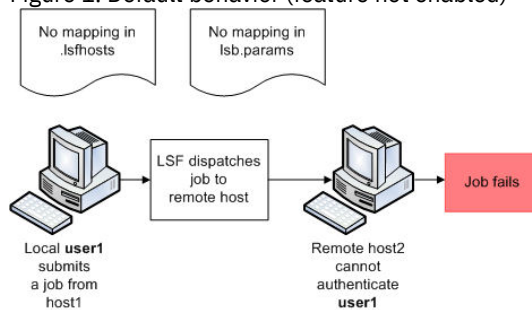


Figure 2. With local user account mapping enabled

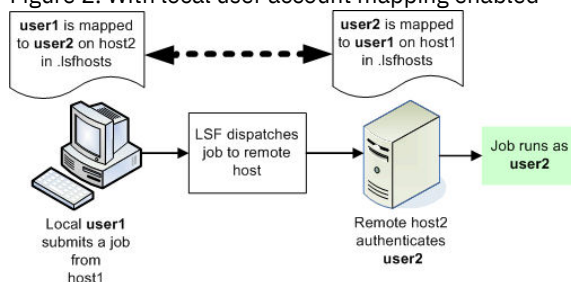
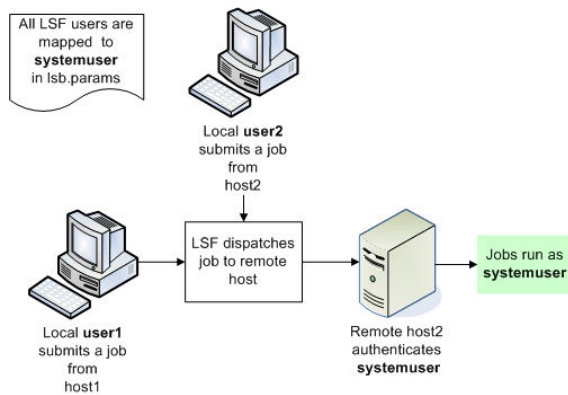


Figure 3. With Windows workgroup account mapping enabled



Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX hosts • Windows hosts • A mix of UNIX and Windows hosts within a single clusters
Not required for	<ul style="list-style-type: none"> • A cluster with a uniform user name space • A mixed UNIX/Windows cluster in which user accounts have the same user name on both operating systems
Dependencies	<ul style="list-style-type: none"> • UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs. • For clusters that include both UNIX and Windows hosts, you must also enable the UNIX/Windows user account mapping feature.
Limitations	<ul style="list-style-type: none"> • For a MultiCluster environment that has different user accounts assigned to different hosts, you must also enable the cross-cluster user account mapping feature. Do not configure between-host user account mapping if you want to use system-level mapping in a multicluster environment; LSF ignores system-level mapping if mapping local user mapping is also defined in .lsfhosts. • For Windows workgroup account mapping in a Windows workgroup environment, all jobs run using the permissions associated with the specified system account.

Configuration to enable between-host user account mapping

Between-host user account mapping can be configured in one of the two ways.

- Users can map their local accounts at the user level in the file .lsfhosts. This file must reside in the user's home directory with owner read/write permissions for UNIX and owner read-write-execute permissions for Windows. It must not be readable and writable by any other user other than the owner. Save the .lsfhosts file without a file extension. Both the remote and local hosts must have corresponding mappings in their respective .lsfhosts files.
- LSF administrators can set up Windows workgroup account mapping at the system level in lsb.params.

Local user account mapping configuration

Local user account mapping is enabled by adding lines to the file .lsfhosts. Both the remote and local hosts must have corresponding mappings in their respective .lsfhosts files.

Configuration file	Syntax	Behavior
.lsfhosts	<i>host_name user_name</i> send	<ul style="list-style-type: none"> Jobs sent from the local account run as <i>user_name</i> on <i>host_name</i>
	<i>host_name user_name</i> recv	<ul style="list-style-type: none"> The local account can run jobs that are received from <i>user_name</i> submitted on <i>host_name</i>
	<i>host_name user_name</i>	<ul style="list-style-type: none"> The local account can send jobs to and receive jobs from <i>user_name</i> on <i>host_name</i>
	+ +	<ul style="list-style-type: none"> The local account can send jobs to and receive jobs from any user on any LSF host

Windows workgroup account mapping

Windows workgroup account mapping is enabled by defining the parameter SYSTEM_MAPPING_ACCOUNT in the file lsb.params.

Configuration file	Parameter and syntax	Default behavior
lsb.params	SYSTEM_MAPPING_ACCOUNT =account	<ul style="list-style-type: none"> Enables Windows workgroup account mapping Windows local user accounts run LSF jobs using the system account name and permissions

Between-host user account mapping behavior

Local user account mapping example

The following example describes how local user account mapping works when configured in the file .lsfhosts in the user's home directory. Only mappings configured in .lsfhosts on both the local and remote hosts work.

In the following example, the cluster contains hostA, hostB, and hostC. The account user1 is valid on all hosts except hostC, which requires a user account name of user99.

To allow ...	On ...	In the home directory oflsfhosts must contain the line ...
The account user1 to run jobs on all hosts within the cluster:			
<ul style="list-style-type: none"> user1 to send jobs to user99 on hostC 	hostA	user1	hostC user99 send
	hostB	user1	hostC user99 send
<ul style="list-style-type: none"> user99 to receive jobs from user1 on either hostA or hostB 	hostC	user99	hostA user1 recv hostB user1 recv

Windows workgroup account mapping example

The following example describes how Windows workgroup account mapping works when configured in the file lsb.params. In this example, the cluster has a Windows workgroup environment, and only the user account jobuser is valid on all hosts.

To allow ...	In lsb.params, configure ...	Behavior
--------------	------------------------------	----------

To allow ...	In lsb.params, configure ...	Behavior
All hosts within the cluster to run jobs on any other host within the cluster:		
<ul style="list-style-type: none"> Map all local users to user account jobuser 	SYSTEM_MAPPING_ACCOUNT=jobuser	When any local user submits an LSF job, the job runs under the account jobuser, using the permissions that are associated with the jobuser account.

Between-host user account mapping commands

Commands for submission

Command	Description
bsub	<ul style="list-style-type: none"> Submits the job with the user name and password of the user who entered the command. The job runs on the execution host with the submission user name and password, unless you have configured between-host user account mapping. With between-host user account mapping enabled, jobs that execute on a remote host run using the account name configured at the system level for Windows workgroups, or at the user level for local user account mapping.

Commands to monitor

Command	Description
bjobs -l	<ul style="list-style-type: none"> Displays detailed information about jobs, including the user name of the user who submitted the job and the user name with which the job executed.
bhist -l	<ul style="list-style-type: none"> Displays detailed historical information about jobs, including the user name of the user who submitted the job and the user name with which the job executed.

Commands to control

Not applicable.

Commands to display configuration

Command	Description
bparams	<ul style="list-style-type: none"> Displays the value of SYSTEM_MAPPING_ACCOUNT defined in lsb.params.
badmin showconf	<ul style="list-style-type: none"> Displays all configured parameters and their values set in lsf.conf or ego.conf that affect mbatchd and sbatchd. Use a text editor to view other parameters in the lsf.conf or ego.conf configuration files. In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Use a text editor to view the file .lsfhosts.

Cross-cluster user account mapping

The cross-cluster user account mapping feature enables cross-cluster job submission and execution for a multicluster environment that has different user accounts assigned to different hosts. Using this feature, you can map user accounts in a local cluster to user accounts in one or more remote clusters.

- [About cross-cluster user account mapping](#)
For multicluster environments that have different user accounts assigned to different hosts, cross-cluster user account mapping allows you to submit a job from a local host and run the job as a different user on a remote host.
- [Configuration to enable cross-cluster user account mapping](#)
You can enable cross-cluster user account mapping at the system or user level. The enablement is completed in different configuration files, but different users, depending on the configuration level.
- [Cross-cluster user account mapping behavior](#)
- [Cross-cluster user account mapping commands](#)

About cross-cluster user account mapping

For multicluster environments that have different user accounts assigned to different hosts, cross-cluster user account mapping allows you to submit a job from a local host and run the job as a different user on a remote host.

Figure 1. Default behavior (feature not enabled)

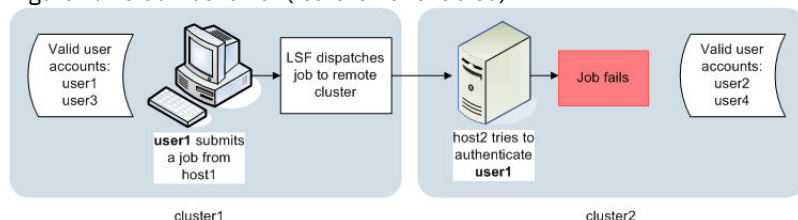
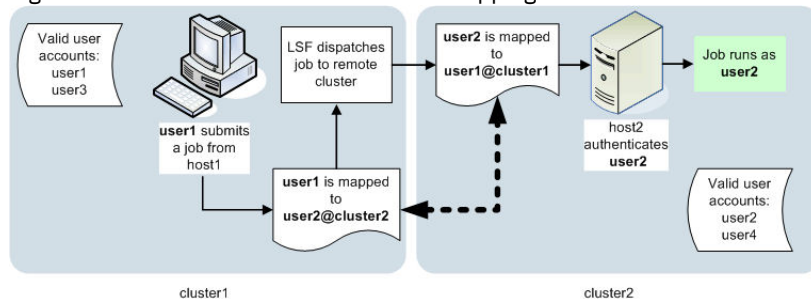


Figure 2. With cross-cluster user account mapping enabled



Scope

Applicability	Details
Operating system	<ul style="list-style-type: none">• UNIX hosts• Windows hosts• A mix of UNIX and Windows hosts within one or more clusters
Not required for	<ul style="list-style-type: none">• Multiple clusters with a uniform user name space
Dependencies	<ul style="list-style-type: none">• UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs.• If users at your site have different user names on UNIX and Windows hosts within a single cluster, you must configure between-host user account mapping at the user level in .lsfhosts.

Applicability	Details
Limitations	<ul style="list-style-type: none"> You cannot configure this feature at both the system-level and the user-level; LSF ignores system-level mapping if user-level mapping is also defined in .lsfhosts. If one or more clusters include both UNIX and Windows hosts, you must also configure UNIX and Windows user account mapping. If one or more clusters have different user accounts assigned to different hosts, you must also configure between-host user account mapping for those clusters, and then configure cross-cluster user account mapping at the system level only.

Configuration to enable cross-cluster user account mapping

You can enable cross-cluster user account mapping at the system or user level. The enablement is completed in different configuration files, but different users, depending on the configuration level.

- LSF administrators can map user accounts at the system level in the **UserMap** section of the lsb.users file. Both the remote and local clusters must have corresponding mappings in their respective lsb.users files.
- Users can map their local accounts at the user level in the .lsfhosts file. This file must reside in the user's home directory with owner read/write permissions for UNIX and owner read-write-execute permissions for Windows. Save the .lsfhosts file without a file extension. Both the remote and local hosts must have corresponding mappings in their respective .lsfhosts files.

Restriction: Define *either* system-level or user-level mapping, but not both. LSF ignores system-level mapping if user-level mapping is also defined in .lsfhosts.

Configuration file	Level	Syntax	Behavior
lsb.users	System	Required fields: LOCAL, REMOTE, and DIRECTION	<ul style="list-style-type: none"> Maps a user name on a local host to a different user name on a remote host Jobs that execute on a remote host run using a mapped user name rather than the job submission user name
.lsfhosts	User	<i>host_name user_name</i> send	<ul style="list-style-type: none"> Jobs sent from the local account run as <i>user_name</i> on <i>host_name</i>
		<i>host_name user_name</i> recv	<ul style="list-style-type: none"> The local account can run jobs received from <i>user_name</i> submitted on <i>host_name</i>
		<i>host_name user_name</i>	<ul style="list-style-type: none"> The local account can send jobs to and receive jobs from <i>user_name</i> on <i>host_name</i>
		<i>cluster_name user_name</i>	<ul style="list-style-type: none"> The local account can send jobs to and receive jobs from <i>user_name</i> on any host in the cluster <i>cluster_name</i>
		+ +	<ul style="list-style-type: none"> The local account can send jobs to and receive jobs from any user on any LSF hosts

Cross-cluster user account mapping behavior

System-level configuration example

The following example illustrates LSF behavior when the LSF administrator sets up cross-cluster user account mapping at the system level. This example shows the UserMap section of the file `lsb.users` on both the local and remote clusters.

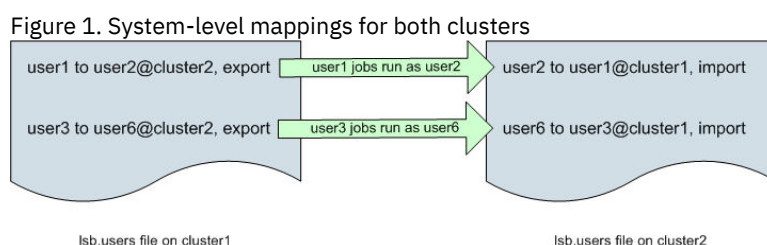
On *cluster1*:

```
Begin UserMap
LOCAL      REMOTE      DIRECTION
user1      user2@cluster2  export
user3      user6@cluster2  export
End UserMap
```

On *cluster2*:

```
Begin UserMap
LOCAL      REMOTE      DIRECTION
user2      user1@cluster1  import
user6      user3@cluster1  import
End UserMap
```

The mappings between users on different clusters are as follows:



Only mappings configured in `lsb.users` on both clusters work. In this example, the common user account mappings are:

- user1@cluster1 to user2@cluster2
- user3@cluster1 to user6@cluster2

User-level configuration examples

The following examples describe how user account mapping works when configured at the user level in the file `.lsfhosts` in the user's home directory. Only mappings that are configured in `.lsfhosts` on hosts in both clusters work.

To allow ...	On ...	In the home directory oflsfhosts must contain the line ...
The accounts user1 and user2 to run jobs on all hosts in both clusters:			
<ul style="list-style-type: none"> • user1 to send jobs to and receive jobs from user2 on cluster2 	All hosts in cluster1	user1	<code>cluster2 user2</code>
<ul style="list-style-type: none"> • user2 to send jobs to and receive jobs from user1 on cluster1 	All hosts in cluster2	user2	<code>cluster1 user1</code>
The account user1 to run jobs on cluster2 using the <code>lsfguest</code> account:			
<ul style="list-style-type: none"> • user1 to send jobs as <code>lsfguest</code> to all hosts in cluster2 	All hosts in cluster1	user1	<code>cluster2 lsfguest send</code>
<ul style="list-style-type: none"> • <code>lsfguest</code> to receive jobs from user1 on cluster1 	All hosts in cluster2	lsfguest	<code>cluster1 user1 recv</code>

Cross-cluster user account mapping commands

Commands for submission

Command	Description
bsub	<ul style="list-style-type: none">Submits the job with the user name and password of the user who entered the command. The job runs on the execution host with the submission user name and password, unless you have configured cross-cluster user account mapping.With cross-cluster user account mapping enabled, jobs that execute on a remote host run using the account name configured at the system or user level.

Commands to monitor

Command	Description
bjobs -l	<ul style="list-style-type: none">Displays detailed information about jobs, including the user name of the user who submitted the job and the user name with which the job executed.
bhist -l	<ul style="list-style-type: none">Displays detailed historical information about jobs, including the user name of the user who submitted the job and the user name with which the job executed.

UNIX and Windows user account mapping

The UNIX and Windows user account mapping feature enables cross-platform job submission and execution in a mixed UNIX and Windows environment. Using this feature, you can map Windows user accounts, which include a domain name, to UNIX user accounts, which do not include a domain name, for user accounts with the same user name on both operating systems.

- [About UNIX and Windows user account mapping](#)
In a mixed UNIX/Windows cluster, LSF treats Windows user names (with domain) and UNIX user names (no domain) as different users. The UNIX/Windows user account mapping feature makes job submission and execution transparent across operating systems by mapping Windows accounts to UNIX accounts. With this feature enabled, LSF sends the user account name in the format that is required by the operating system on the execution host.
- [Configuration to enable UNIX and Windows user account mapping](#)
Enable the UNIX/Windows user account mapping feature by defining one or more LSF user domains using the LSF_USER_DOMAIN parameter in lsfc.conf.
- [UNIX and Windows user account mapping behavior](#)
The following examples describe how UNIX/Windows user account mapping enables job submission and execution across a mixed UNIX/Windows cluster.
- [Configuration to modify UNIX and Windows user account mapping behavior](#)
You can select a preferred execution domain for a particular job. The execution domain must be included in the LSF_USER_DOMAIN list. When you specify an execution domain, LSF ignores the order of the domains listed in LSF_USER_DOMAIN and runs the job using the specified domain. The environment variable LSF_EXECUTE_DOMAIN, defined in the user environment or from the command line, defines the preferred execution domain. Once you submit a job with an execution domain defined, you cannot change the execution domain for that particular job.
- [UNIX and Windows user account mapping commands](#)

About UNIX and Windows user account mapping

In a mixed UNIX/Windows cluster, LSF treats Windows user names (with domain) and UNIX user names (no domain) as different users. The UNIX/Windows user account mapping feature makes job submission and execution transparent across operating systems by mapping Windows accounts to UNIX accounts. With this feature enabled, LSF sends the user account name in the format that is required by the operating system on the execution host.

Figure 1. Default behavior (feature not enabled)

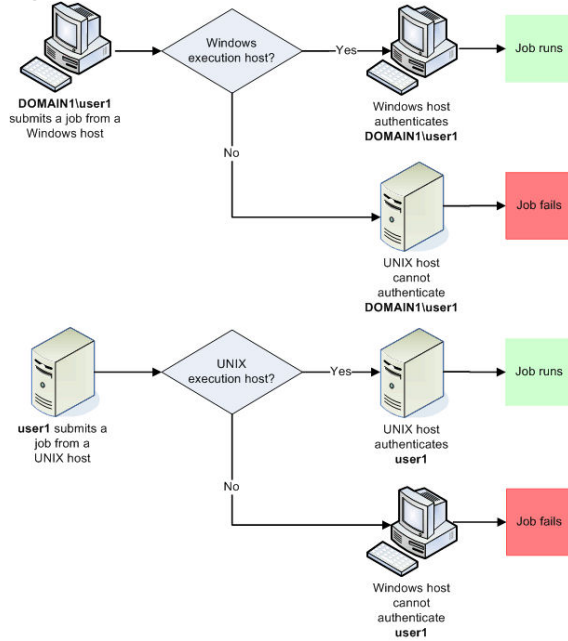
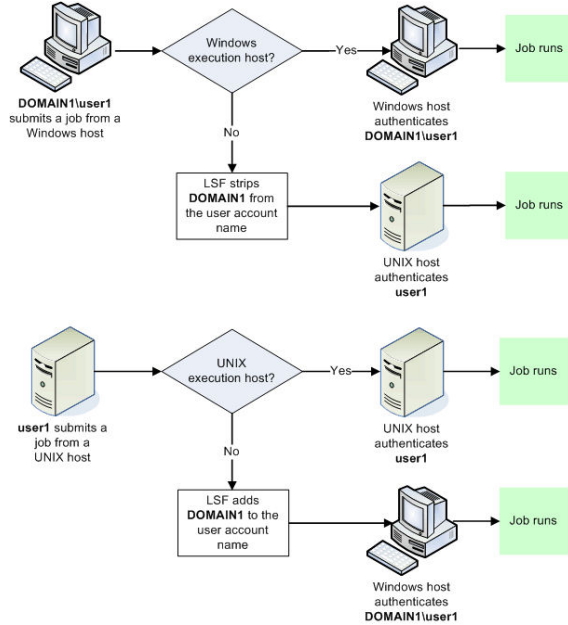


Figure 2. With UNIX/Windows user account mapping enabled



For mixed UNIX/Windows clusters, UNIX/Windows user account mapping allows you to do the following:

- Submit a job from a Windows host and run the job on a UNIX host
- Submit a job from a UNIX host and run the job on a Windows host
- Specify the domain\user combination that is used to run a job on a Windows host
- Schedule and track jobs that are submitted with either a Windows or UNIX account as though the jobs belong to a single user

LSF supports the use of both single and multiple Windows domains. In a multiple domain environment, you can choose one domain as the preferred execution domain for a particular job.

Existing Windows domain trust relationships apply in LSF. If the execution domain trusts the submission domain, the submission account is valid on the execution host.

Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> UNIX and Windows hosts within a single cluster
Not required for	<ul style="list-style-type: none"> Windows-only clusters UNIX-only clusters
Dependencies	<ul style="list-style-type: none"> UNIX and Windows user accounts must be valid on all hosts in the cluster and must have the correct permissions to successfully run jobs.
Limitations	<ul style="list-style-type: none"> This feature works with a uniform user name space. If users at your site have different user names on UNIX and Windows hosts, you must enable between-host user account mapping. This feature does not affect Windows workgroup installations. If you want to map all Windows workgroup users to a single Windows system account, you must configure between-host user account mapping. This feature applies only to job execution. If you issue an LSF command or define an LSF parameter and specify a Windows user, you must use the long form of the user name, including the domain name typed in uppercase letters.

Configuration to enable UNIX and Windows user account mapping

Enable the UNIX/Windows user account mapping feature by defining one or more LSF user domains using the LSF_USER_DOMAIN parameter in lsf.conf.

Important: Configure LSF_USER_DOMAIN immediately after you install LSF—changing this parameter in an existing cluster requires that you verify and possibly reconfigure service accounts, user group memberships, and user passwords.

Configuration file	Parameter and syntax	Behavior
lsf.conf	LSF_USER_DOMAIN= <i>domain_name</i>	<ul style="list-style-type: none"> Enables Windows domain account mapping in a single-domain environment To run jobs on a UNIX host, LSF strips the specified domain name from the user name To run jobs on a Windows host, LSF appends the domain name to the user name
	LSF_USER_DOMAIN= <i>domain_name:domain_name...</i>	<ul style="list-style-type: none"> Enables Windows domain account mapping in a multi-domain environment To run jobs on a UNIX host, LSF strips the specified domain names from the user name To run jobs on a Windows host, LSF appends the first domain name to the user name. If the first domain\user combination does not have permissions to run the job, LSF tries the next domain in the LSF_USER_DOMAIN list.
	LSF_USER_DOMAIN=.	<ul style="list-style-type: none"> Enables Windows domain account mapping To run jobs on a UNIX host, LSF strips the local machine name from the user name To run jobs on a Windows host, LSF appends the local machine name to the user name

UNIX and Windows user account mapping behavior

The following examples describe how UNIX/Windows user account mapping enables job submission and execution across a mixed UNIX/Windows cluster.

When...	In the file ...	And the job is submitted by ...	The job ...
UNIX/Windows user account mapping is not enabled	—	<ul style="list-style-type: none">BUSINESS\user1 on a Windows host	<ul style="list-style-type: none">Runs on a Windows host as BUSINESS\user1Fails on a UNIX host: BUSINESS\user1 is not a valid UNIX user name
UNIX/Windows user account mapping is not enabled	—	<ul style="list-style-type: none">user1 on a UNIX host	<ul style="list-style-type: none">Fails on a Windows host: Windows requires a domain\user combinationRuns on a UNIX host as user1
LSF_USER_DOMAIN=BUSINESS	lsf.conf	<ul style="list-style-type: none">BUSINESS\user1 on a Windows host	<ul style="list-style-type: none">Runs on a Windows host as BUSINESS\user1Runs on a UNIX host as user1
LSF_USER_DOMAIN=BUSINESS	lsf.conf	<ul style="list-style-type: none">user1 on a UNIX host	<ul style="list-style-type: none">Runs on a Windows host as BUSINESS\user1Runs on a UNIX host as user1
LSF_USER_DOMAIN=SUPPORT:ENGINEERING	lsf.conf	<ul style="list-style-type: none">SUPPORT\user1 on a Windows host	<ul style="list-style-type: none">Runs on a Windows host as SUPPORT\user1Runs on a UNIX host as user1
LSF_USER_DOMAIN=SUPPORT:ENGINEERING	lsf.conf	<ul style="list-style-type: none">BUSINESS\user1 on a Windows host	<ul style="list-style-type: none">Runs on a Windows host as BUSINESS\user1Fails on a UNIX host: LSF cannot strip the domain name, and BUSINESS\user1 is not a valid UNIX user name
LSF_USER_DOMAIN=SUPPORT:ENGINEERING	lsf.conf	<ul style="list-style-type: none">user1 on a UNIX host	<ul style="list-style-type: none">Runs on a Windows host as SUPPORT\user1; if the job cannot run with those credentials, the job runs as ENGINEERING\user1Runs on a UNIX host as user1

Configuration to modify UNIX and Windows user account mapping behavior

You can select a preferred execution domain for a particular job. The execution domain must be included in the LSF_USER_DOMAIN list. When you specify an execution domain, LSF ignores the order of the domains listed in LSF_USER_DOMAIN and runs the job using the specified domain. The environment variable LSF_EXECUTE_DOMAIN, defined in the user environment or from the command line, defines the preferred execution domain. Once you submit a job with an execution domain defined, you cannot change the execution domain for that particular job.

Configuration file	Parameter and syntax	Behavior
--------------------	----------------------	----------

Configuration file	Parameter and syntax	Behavior
.cshrc . profile	LSF_EXECUTE_DOMAIN= domain_name	<ul style="list-style-type: none"> Specifies the domain that LSF uses to run jobs on a Windows host If LSF_USER_DOMAIN contains a list of multiple domains, LSF tries the LSF_EXECUTE_DOMAIN first

The following example shows the changed behavior when you define the LSF_EXECUTE_DOMAIN.

When...	In the file ...	And the job is submitted by ...	The job ...
LSF_USER_DOMAIN= SUPPORT:ENGINEERING and LSF_EXECUTE_DOMAIN= ENGINEERING	lsf.conf . profile .cshrc	<ul style="list-style-type: none"> user1 on a UNIX host 	<ul style="list-style-type: none"> Runs on a Windows host as ENGINEERING\user1; if the job cannot run with those credentials, runs as SUPPORT\user1 Runs on a UNIX host as user1

These additional examples are based on the following conditions:

- In lsf.conf, LSF_USER_DOMAIN=SALES:ENGINEERING:BUSINESS
- The user has sufficient permissions to run the job in any of the LSF user domains

UNIX user1 enters ...	And LSF_EXECUTE_DOMAIN is ...	Then LSF runs the job as ...
bsub -m "hostb" myjob	Not defined in the user environment file	SALES\user1
bsub -m "hostb" myjob	Defined as BUSINESS in the user environment file	BUSINESS\user1
setenv LSF_EXECUTE_DOMAIN BUSINESS bsub -m "hostb" myjob	Either defined or not defined in the user environment file	BUSINESS\user1 The command line overrides the user environment file.

UNIX and Windows user account mapping commands

Commands for submission

Command	Description
bsub	<ul style="list-style-type: none"> Submits the job with the user name and password of the user who entered the command. The job runs on the execution host with the same user name and password, unless you have configured UNIX/Windows user account mapping. With UNIX/Windows user account mapping enabled, jobs that execute on a remote host run with the user account name in the format required by the operating system on the execution host.

Commands to monitor

Command	Description
bjobs -w	<ul style="list-style-type: none"> Displays detailed information about jobs. Displays the long form of the Windows user name including the domain name.

Commands to control

Command	Description
lspasswd	<ul style="list-style-type: none">Registers a password for a Windows user account. Windows users must register a password for each domain\user account using this command.

Commands to display configuration

Command	Description
bugroup -w	<ul style="list-style-type: none">Displays information about user groups.If UNIX/Windows user account mapping is enabled, the command bugroup displays user names without domains.If UNIX/Windows user account mapping is not enabled, the command bugroup displays user names with domains.
busers	<ul style="list-style-type: none">Displays information about specific users and user groups.If UNIX/Windows user account mapping is enabled, the command busers displays user names without domains.If UNIX/Windows user account mapping is not enabled, the command busers displays user names with domains.
badmin showconf	<ul style="list-style-type: none">Displays all configured parameters and their values set in lsf.conf or ego.conf that affect mbatchd and sbatchd. <p>Use a text editor to view other parameters in the lsf.conf or ego.conf configuration files.</p> <ul style="list-style-type: none">In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Creating a user group using bconf

Procedure

Run **bconf create usergroup=group_name**

For example:

```
bconf create usergroup=ug12 "GROUP_MEMBER=user1 user2 ; USER_SHARES=[user1, 5]
[user2, 2] ; GROUP_ADMIN=admin1"
bconf: Request for usergroup <ug12> accepted
```

Once accepted by **bconf**, the new user group appears in **bugroup** output:

```
bugroup -l ug12
GROUP_NAME:  ug12
USERS:      user2 user1
GROUP_ADMIN: admin1
SHARES:     [user1, 5] [user2, 2]
```

Removing a user group member using bconf

About this task

You can remove members from a user group using live reconfiguration.

And removing the specified group member, all references to the group member are updated as required.

Procedure

Run **bconf rmmember usergroup=group_name "GROUP_MEMBER=user_name"**

For example:

```
bconf rmmember usergroup=ug12 "GROUP_MEMBER=user1"
bconf: Request for usergroup <ug12> accepted
```

Once accepted by **bconf**, the changed user group appears in **bugroup** output:

```
bugroup -l ug12
GROUP_NAME:  ug12
USERS:      user2
GROUP_ADMIN: admin1
SHARES:     [user2, 2]
```

Notice the SHARES entry for user1 is also removed.

Job groups

A collection of jobs can be organized into job groups for easy management. A job group is a container for jobs in much the same way that a directory in a file system is a container for files. For example, a payroll application may have one group of jobs that calculates weekly payments, another job group for calculating monthly salaries, and a third job group that handles the salaries of part-time or contract employees. Users can submit, view, and control jobs according to their groups rather than looking at individual jobs.

How job groups are created

Job groups can be created *explicitly* or *implicitly*:

- A job group is created *explicitly* with the **bgadd** command.
- A job group is created *implicitly* by the **bsub -g** or **bmod -g** command when the specified group does not exist. Job groups are also created implicitly when a default job group is configured (DEFAULT_JOBGROUP in lsb.params or LSB_DEFAULT_JOBGROUP environment variable).

Job groups that are created when jobs are attached to an SLA service class at submission are implicit job groups (**bsub -sla service_class_name -g job_group_name**). Job groups that are attached to an SLA service class with **bgadd** are explicit job groups (**bgadd -sla service_class_name job_group_name**).

The GRP_ADD event in lsb.events indicates how the job group was created:

- 0x01 - job group was created explicitly
- 0x02 - job group was created implicitly

For example:

```
GRP_ADD" "7.02" 1193032735 1285 1193032735 0 "/z" "" "user1" "" "" 2 0 "" -1 1
```

Means job group /z is an explicitly created job group.

Child groups can be created explicitly or implicitly under any job group. Only an implicitly created job group which has no job group limit (**bgadd -L**) and is not attached to any SLA can be automatically deleted once it becomes empty. An empty job group is a job group that has no jobs that are associated with it (including finished jobs). NJOBS displayed by **bigroup** is 0.

Job group hierarchy

Jobs in job groups are organized into a hierarchical tree similar to the directory structure of a file system. Like a file system, the tree contains groups (which are like directories) and jobs (which are like files). Each group can contain other groups or

individual jobs. Job groups are created independently of jobs, and can have dependency conditions which control when jobs within the group are considered for scheduling.

Job group path

The *job group path* is the name and location of a job group within the job group hierarchy. Multiple levels of job groups can be defined to form a hierarchical tree. A job group can contain jobs and sub-groups.

Root job group

LSF maintains a single tree under which all jobs in the system are organized. The top-most level of the tree is represented by a top-level “root” job group, named “/”. The root group is owned by the primary LSF Administrator and cannot be removed. Users and administrators create new groups under the root group. By default, if you do not specify a job group path name when submitting a job, the job is created under the top-level “root” job group, named “/”.

The root job group is not displayed by job group query commands, and you cannot specify the root job in commands.

Job group owner

Each group is owned by the user who created it. The login name of the user who creates the job group is the job group owner. Users can add job groups into a group that are owned by other users, and they can submit jobs to groups owned by other users. Child job groups are owned by the creator of the job group and the creators of any parent groups.

Job control under job groups

Job owners can control their own jobs that are attached to job groups as usual. Job group owners can also control any job under the groups they own and below.

For example:

- Job group /A is created by `user1`
- Job group /A/B is created by `user2`
- Job group /A/B/C is created by `user3`

All users can submit jobs to any job group, and control the jobs they own in all job groups. For jobs submitted by other users:

- `user1` can control jobs that are submitted by other users in all three job groups: /A, /A/B, and /A/B/C
- `user2` can control jobs that are submitted by other users only in two job groups: /A/B and /A/B/C
- `user3` can control jobs that are submitted by other users only in job group /A/B/C

The LSF administrator can control jobs in any job group.

Default job group

You can specify a default job group for jobs submitted without explicitly specifying a job group. LSF associates the job with the job group specified with `DEFAULT_JOBGROUP` in `lsb.params`. The `LSB_DEFAULT_JOBGROUP` environment variable overrides the setting of `DEFAULT_JOBGROUP`. The **bsub -g job_group_name** option overrides both `LSB_DEFAULT_JOBGROUP` and `DEFAULT_JOBGROUP`.

Default job group specification supports macro substitution for project name (%p) and user name (%u). When you specify **bsub -P project_name**, the value of %p is the specified project name. If you do not specify a project name at job submission, %p is the project name defined by setting the environment variable `LSB_DEFAULTPROJECT`, or the project name specified by `DEFAULT_PROJECT` in `lsb.params`. the default project name is default.

For example, a default job group name specified by `DEFAULT_JOBGROUP=/canada/%p/%u` is expanded to the value for the LSF project name and the user name of the job submission user (for example, `/canada/projects/user1`).

Job group names must follow this format:

- Job group names must start with a slash character (/). For example, `DEFAULT_JOBGROUP=/A/B/C` is correct, but `DEFAULT_JOBGROUP=A/B/C` is not correct.
- Job group names cannot end with a slash character (/). For example, `DEFAULT_JOBGROUP=/A/` is not correct.
- Job group names cannot contain more than one slash character (/) in a row. For example, job group names like `DEFAULT_JOBGROUP=/A//B` or `DEFAULT_JOBGROUP=A///B` are not correct.
- Job group names cannot contain spaces. For example, `DEFAULT_JOBGROUP=/A/B C/D` is not correct.
- Project names and user names used for macro substitution with %p and %u cannot start or end with slash character (/).
- Project names and user names used for macro substitution with %p and %u cannot contain spaces or more than one slash character (/) in a row.
- Project names or user names containing slash character (/) will create separate job groups. For example, if the project name is `canada/projects`, `DEFAULT_JOBGROUP=%p` results in a job group hierarchy `/canada/projects`.
- **[Job group limits](#)**
Job group limits specified with **bgadd -L** apply to the job group hierarchy. The job group limit is a positive number greater than or equal to zero, specifying the maximum number of running and suspended jobs under the job group (including child groups). If limit is zero, no jobs under the job group can run. By default, a job group has no limit. Limits persist across **mbatchd** restart and reconfiguration.
- **[Creating a job group](#)**
- **[Submitting jobs under a job group](#)**
- **[Viewing information about job groups \(bjgroup\)](#)**
- **[Viewing jobs for a specific job group \(bjobs\)](#)**
- **[Job groups and time-based SLAs](#)**
Job groups provide a method for assigning arbitrary labels to groups of jobs. Typically, job groups represent a project hierarchy. You can use -g with -sla at job submission to attach all jobs in a job group to a service class and have them scheduled as SLA jobs, subject to the scheduling policy of the SLA. Within the job group, resources are allocated to jobs on a fair share basis.
- **[Control jobs in job groups](#)**
Suspend and resume jobs in job groups, move jobs to different job groups, terminate jobs in job groups, and delete job groups.
- **[Automatic job group cleanup](#)**
When an implicitly created job group becomes empty, it can be automatically deleted by LSF.

Job group limits

Job group limits specified with **bgadd -L** apply to the job group hierarchy. The job group limit is a positive number greater than or equal to zero, specifying the maximum number of running and suspended jobs under the job group (including child groups). If limit is zero, no jobs under the job group can run. By default, a job group has no limit. Limits persist across **mbatchd** restart and reconfiguration.

You cannot specify a limit for the root job group. The root job group has no job limit. Job groups added with no limits specified inherit any limits of existing parent job groups. The **-L** option only limits the lowest level job group created. The maximum number of running and suspended jobs (including USUSP and SSUSP) in a job group cannot exceed the limit defined on the job group and its parent job group.

The job group limit is based on the number of running and suspended jobs in the job group. If you specify a job group limit as 2, at most 2 jobs can run under the group at any time, regardless of how many jobs or job slots are used. If the currently available job slots is zero, even if the job group job limit is not exceeded, LSF cannot dispatch a job to the job group.

If a parallel job requests 2 CPUs (**bsub -n 2**), the job group limit is per job, not per slots used by the job.

A job array may also be under a job group, so job arrays also support job group limits.

Job group limits are not supported at job submission for job groups that are created automatically with **bsub -g**. Use **bgadd -L** before job submission.

Jobs forwarded to the execution cluster in a multicluster environment are not counted towards the job group limit.

Examples

```
bgadd -L 6 /canada/projects/test
```

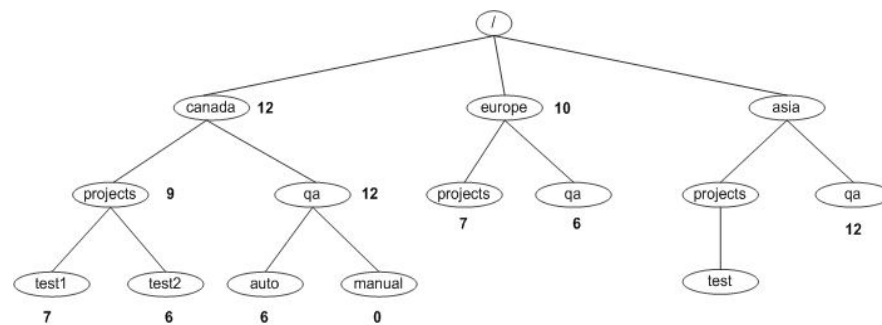
If /canada is existing job group, and /canada/projects and /canada/projects/test are new groups, only the job group /canada/projects/test is limited to 6 running and suspended jobs. Job group /canada/projects will have whatever limit is specified for its parent job group /canada. The limit of /canada does not change.

The limits on child job groups cannot exceed the parent job group limit. For example, if /canada/projects has a limit of 5:

```
bgadd -L 6 /canada/projects/test
```

is rejected because /canada/projects/test attempts to increase the limit of its parent /canada/projects from 5 to 6.

Example job group hierarchy with limits



In this configuration:

- Every node is a job group, including the root (/) job group
- The root (/) job group cannot have any limit definition
- By default, child groups have the same limit definition as their direct parent group, so /asia, /asia/projects, and /asia/projects/test all have no limit
- The number of running and suspended jobs in a job group (including all of its child groups) cannot exceed the defined limit
- If there are 7 running or suspended jobs in job group /canada/projects/test1, even though the job limit of group /canada/qa/auto is 6, /canada/qa/auto can only have a maximum of 5 running and suspended (12-7=5)
- When a job is submitted to a job group, LSF checks the limits for the entire job group. For example, for a job is submitted to job group /canada/qa/auto, LSF checks the limits on groups /canada/qa/auto, /canada/qa and /canada. If any one limit in the branch of the hierarchy is exceeded, the job remains pending
- The zero job limit for job group /canada/qa/manual means that no job in the job group can enter running status

Creating a job group

Procedure

Use the **bgadd** command to create a new job group.

You must provide full group path name for the new job group. The last component of the path is the name of the new group to be created:

```
bgadd /risk_group
```

The preceding example creates a job group named `risk_group` under the root group `/`.

```
bgadd /risk_group/portfolio1
```

The preceding example creates a job group named `portfolio1` under job group `/risk_group`.

```
bgadd /risk_group/portfolio1/current
```

The preceding example creates a job group named `current` under job group `/risk_group/portfolio1`.

If the group hierarchy `/risk_group/portfolio1/current` does not exist, LSF checks its parent recursively, and if no groups in the hierarchy exist, all three job groups are created with the specified hierarchy.

Adding a job group limit (bgadd)

Procedure

Run **bgadd -L limit /job_group_name** to specify a job limit for a job group.

Where *limit* is a positive number greater than or equal to zero, specifying the maximum the number of running and suspended jobs under the job group (including child groups) If limit is zero, no jobs under the job group can run.

For example:

```
bgadd -L 6 /canada/projects/test
```

If `/canada` is existing job group, and `/canada/projects` and `/canada/projects/test` are new groups, only the job group `/canada/projects/test` is limited to 6 running and suspended jobs. Job group `/canada/projects` will have whatever limit is specified for its parent job group `/canada`. The limit of `/canada` does not change.

Submitting jobs under a job group

Procedure

Use the **-g** option of **bsub** to submit a job into a job group.

The job group does not have to exist before submitting the job.

```
bsub -g /risk_group/portfolio1/current myjob
Job <105> is submitted to default queue.
```

Submits `myjob` to the job group `/risk_group/portfolio1/current`.

If group `/risk_group/portfolio1/current` exists, job 105 is attached to the job group.

If group `/risk_group/portfolio1/current` does not exist, LSF checks its parent recursively, and if no groups in the hierarchy exist, all three job groups are created with the specified hierarchy and the job is attached to group.

Example

-g and -sla options

Tip:

Use **-sla** with **-g** to attach all jobs in a job group to a service class and have them scheduled as SLA jobs. Multiple job groups can be created under the same SLA. You can submit more jobs to the job group without specifying the service class name again.

Multicluster

In a multicluster job forwarding mode, job groups only apply on the submission cluster, not on the execution cluster. LSF treats the execution cluster as execution engine, and only enforces job group policies at the submission cluster.

Jobs forwarded to the execution cluster in a multicluster environment are not counted towards job group limits.

Viewing information about job groups (bjgroup)

Procedure

1. Use the **bjgroup** command to see information about jobs in job groups.

bjgroup									
GROUP_NAME	NJOBS	PEND	RUN	SSUSP	USUSP	FINISH	SLA	JLIMIT	OWNER
/A	0	0	0	0	0	0	()	0/10	user1
/X	0	0	0	0	0	0	()	0/-	user2
/A/B	0	0	0	0	0	0	()	0/5	user1
/X/Y	0	0	0	0	0	0	()	0/5	user2

2. Use **bjgroup -s** to sort job groups by group hierarchy.

For example, for job groups named /A, /A/B, /X and /X/Y, **bjgroup -s** displays:

bjgroup -s									
GROUP_NAME	NJOBS	PEND	RUN	SSUSP	USUSP	FINISH	SLA	JLIMIT	OWNER
/A	0	0	0	0	0	0	()	0/10	user1
/A/B	0	0	0	0	0	0	()	0/5	user1
/X	0	0	0	0	0	0	()	0/-	user2
/X/Y	0	0	0	0	0	0	()	0/5	user2

3. Specify a job group name to show the hierarchy of a single job group:

bjgroup -s /X									
GROUP_NAME	NJOBS	PEND	RUN	SSUSP	USUSP	FINISH	SLA	JLIMIT	OWNER
/X	25	0	25	0	0	0	puccini	25/100	user1
/X/Y	20	0	20	0	0	0	puccini	20/30	user1
/X/Z	5	0	5	0	0	0	puccini	5/10	user2

4. Specify a job group name with a trailing slash character (/) to show only the root job group:

bjgroup -s /X/									
GROUP_NAME	NJOBS	PEND	RUN	SSUSP	USUSP	FINISH	SLA	JLIMIT	OWNER
/X	25	0	25	0	0	0	puccini	25/100	user1

5. Use **bjgroup -N** to display job group information by job slots instead of number of jobs. NSLOTS, PEND, RUN, SSUSP, USUSP, RSV are all counted in slots rather than number of jobs:

bjgroup -N								
GROUP_NAME	NSLOTS	PEND	RUN	SSUSP	USUSP	RSV	SLA	OWNER
/X	25	0	25	0	0	0	puccini	user1
/A/B	20	0	20	0	0	0	wagner	batch

-N by itself shows job slot info for all job groups, and can combine with -s to sort the job groups by hierarchy:

bjgroup -N -s								
GROUP_NAME	NSLOTS	PEND	RUN	SSUSP	USUSP	RSV	SLA	OWNER
/A	0	0	0	0	0	0	wagner	batch
/A/B	0	0	0	0	0	0	wagner	user1
/X	25	0	25	0	0	0	puccini	user1
/X/Y	20	0	20	0	0	0	puccini	batch
/X/Z	5	0	5	0	0	0	puccini	batch

Viewing jobs for a specific job group (bjobs)

Procedure

Run **bjobs -g** and specify a job group path to view jobs that are attached to the specified group.

bjobs -g /risk_group							
JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
113	user1	PEND	normal	hostA		myjob	Jun 17 16:15
111	user2	RUN	normal	hostA	hostA	myjob	Jun 14 15:13
110	user1	RUN	normal	hostB	hostA	myjob	Jun 12 05:03
104	user3	RUN	normal	hostA	hostC	myjob	Jun 11 13:18

bjobs -l displays the full path to the group to which a job is attached:

```
bjobs -l -g /risk_group
Job <101>, User <user1>, Project <default>, Job Group </risk_group>, Status <RUN>,
Queue <normal>, Command <myjob>
```

```
Tue Jun 17 16:21:49 2009: Submitted from host <hostA>, CWD </home/user1>;
Tue Jun 17 16:22:01 2009: Started on <hostA>;
...
```

Job groups and time-based SLAs

Job groups provide a method for assigning arbitrary labels to groups of jobs. Typically, job groups represent a project hierarchy. You can use `-g` with `-sla` at job submission to attach all jobs in a job group to a service class and have them scheduled as SLA jobs, subject to the scheduling policy of the SLA. Within the job group, resources are allocated to jobs on a fair share basis.

All jobs submitted to a group under an SLA automatically belong to the SLA itself. You cannot modify a job group of a job that is attached to an SLA.

A job group hierarchy can belong to only one SLA.

It is not possible to have some jobs in a job group not part of the service class. Multiple job groups can be created under the same SLA. You can submit additional jobs to the job group without specifying the service class name again.

If the specified job group does not exist, it is created and attached to the SLA.

You can also use `-sla` to specify a service class when you create a job group with `bgadd`.

- [Viewing job groups attached to a time-based SLA \(bjgroup\)](#)

Viewing job groups attached to a time-based SLA (bjgroup)

Procedure

Run `bjgroup` to display job groups that are attached to a time-based SLA:

```
bjgroup
GROUP_NAME  NJOBS  PEND  RUN  SSUSP  USUSP  FINISH  SLA      JLIMIT  OWNER
/fund1_grp   5      4     0     1      0      0  Venezia  1/5    user1
/fund2_grp  11      2     5     0      0      4  Venezia  5/5    user1
/bond_grp    2      2     0     0      0      0  Venezia  0/-    user2
/risk_grp    2      1     1     0      0      0      ()    1/-    user2
/admi_grp    4      4     0     0      0      0      ()    0/-    user2
```

`bjgroup` displays the name of the service class that the job group is attached to with `bgadd -sla service_class_name`. If the job group is not attached to any service class, empty parentheses `()` are displayed in the SLA name column.

Control jobs in job groups

Suspend and resume jobs in job groups, move jobs to different job groups, terminate jobs in job groups, and delete job groups.

- [Suspending jobs \(bstop\)](#)
- [Resuming suspended jobs \(bresume\)](#)
- [Moving jobs to a different job group \(bmod\)](#)
- [Terminating jobs \(bkill\)](#)
- [Deleting a job group manually \(bgdel\)](#)
- [Modifying a job group limit \(bgmod\)](#)

Suspending jobs (bstop)

Procedure

1. Use the **-g** option of **bstop** and specify a job group path to suspend jobs in a job group

```
bstop -g /risk_group 106
Job <106> is being stopped
```

2. Use job ID 0 (zero) to suspend all jobs in a job group:

```
bstop -g /risk_group/consolidate 0
Job <107> is being stopped
Job <108> is being stopped
Job <109> is being stopped
```

Resuming suspended jobs (bresume)

Procedure

1. Use the **-g** option of **bresume** and specify a job group path to resume suspended jobs in a job group:

```
bresume -g /risk_group 106
Job <106> is being resumed
```

2. Use job ID 0 (zero) to resume all jobs in a job group:

```
bresume -g /risk_group 0
Job <109> is being resumed
Job <110> is being resumed
Job <112> is being resumed
```

Moving jobs to a different job group (bmod)

Procedure

Use the **-g** option of **bmod** and specify a job group path to move a job or a job array from one job group to another.

```
bmod -g /risk_group/portfolio2/monthly 105
```

Moves job 105 to job group `/risk_group/portfolio2/monthly`.

Like **bsub -g**, if the job group does not exist, LSF creates it.

bmod -g cannot be combined with other **bmod** options. It can only operate on pending jobs. It cannot operate on running or finished jobs.

If you define `LSB_MOD_ALL_JOBS=Y` in `lsf.conf`, **bmod -g** can also operate on running jobs.

You can modify your own job groups and job groups that other users create under your job groups. The LSF administrator can modify job groups of all users.

You cannot move job array elements from one job group to another, only entire job arrays. If any job array elements in a job array are running, you cannot move the job array to another group. A job array can only belong to one job group at a time.

You cannot modify the job group of a job that is attached to a service class.

bhist -l shows job group modification information:

```
bhist -l 105
Job <105>, User <user1>, Project <default>, Job Group </risk_group>, Command <myjob>
```

```
Wed May 14 15:24:07 2009: Submitted from host <hostA>, to Queue <normal>, CWD
<$HOME/lfs51/5.1/sparc-sol7-64/bin>;
Wed May 14 15:24:10 2009: Parameters of Job are changed:
Job group changes to: /risk_group/portfolio2/monthly;
Wed May 14 15:24:17 2009: Dispatched to <hostA>;
Wed May 14 15:24:17 2009: Starting (Pid 8602);
...
```

Terminating jobs (bkill)

Procedure

1. Use the **-g** option of **bkill** and specify a job group path to terminate jobs in a job group.

```
bkill -g /risk_group 106
Job <106> is being terminated
```

2. Use job ID 0 (zero) to terminate all jobs in a job group:

```
bkill -g /risk_group 0
Job <1413> is being terminated
Job <1414> is being terminated
Job <1415> is being terminated
Job <1416> is being terminated
```

bkill only kills jobs in the job group you specify. It does not kill jobs in lower-level job groups in the path. For example, jobs are attached to job groups `/risk_group` and `/risk_group/consolidate`:

```
bsub -g /risk_group myjob
Job <115> is submitted to default queue <normal>.
bsub -g /risk_group/consolidate myjob2
Job <116> is submitted to default queue <normal>.
```

The following **bkill** command only kills jobs in `/risk_group`, not the subgroup `/risk_group/consolidate`:

```
bkill -g /risk_group 0
Job <115> is being terminated
```

To kill jobs in `/risk_group/consolidate`, specify the path to the `consolidate` job group explicitly:

```
bkill -g /risk_group/consolidate 0
Job <116> is being terminated
```

Deleting a job group manually (bgdel)

Procedure

1. Use the **bgdel** command to manually remove a job group. The job group cannot contain any jobs.

```
bgdel /risk_group
Job group /risk_group is deleted.
```

Deletes the job group `/risk_group` and all its subgroups.

Normal users can only delete the empty groups that they own that are specified by the requested *job_group_name*. These groups can be explicit or implicit.

2. Run **bgdel 0** to delete all empty job groups you own. These groups can be explicit or implicit.
3. LSF administrators can use **bgdel -u user_name 0** to delete all empty job groups that are created by specific users. These groups can be explicit or implicit.

Run **bgdel -u all 0** to delete all the users' empty job groups and their sub groups. LSF administrators can delete empty job groups that are created by any user. These groups can be explicit or implicit.

4. Run **bgdel -c *job_group_name*** to delete all empty groups below the requested *job_group_name* including *job_group_name* itself.

Modifying a job group limit (bgmod)

Procedure

Run **bgmod** to change a job group limit.

```
bgmod [-L limit | -Ln] /job_group_name
```

where:

- -L *limit* changes the limit of *job_group_name* to the specified value. If the job group has parent job groups, the new limit cannot exceed the limits of any higher level job groups. Similarly, if the job group has child job groups, the new value must be greater than any limits on the lower-level job groups.
- -Ln removes the existing job limit for the job group. If the job group has parent job groups, the job modified group automatically inherits any limits from its direct parent job group.

You must provide full group path name for the modified job group. The last component of the path is the name of the job group to be modified.

Only root, LSF administrators, or the job group creator, or the creator of the parent job groups can use **bgmod** to modify a job group limit.

The following command only modifies the limit of group `/canada/projects/test1`. It does not modify limits of `/canada` or `/canada/projects`.

```
bgmod -L 6 /canada/projects/test1
```

To modify limits of `/canada` or `/canada/projects`, you must specify the exact group name:

```
bgmod -L 6 /canada
```

or

```
bgmod -L 6 /canada/projects
```

Automatic job group cleanup

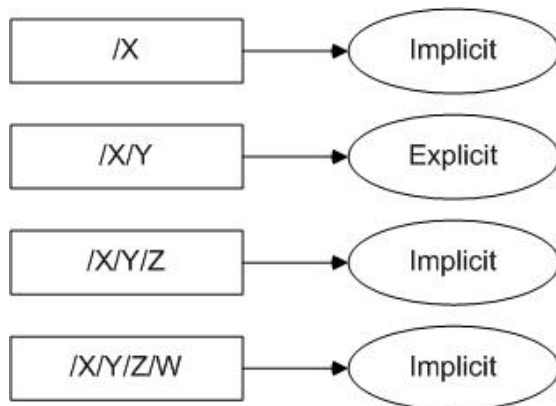
When an implicitly created job group becomes empty, it can be automatically deleted by LSF.

Job groups that can be automatically deleted cannot:

- Have limits that are specified including their child groups
- Have explicitly created child job groups
- Be attached to any SLA

Configure `JOB_GROUP_CLEAN=Y` in `lsb.params` to enable automatic job group deletion.

For example, for the following job groups:



When automatic job group deletion is enabled, LSF only deletes job groups `/X/Y/Z/W` and `/X/Y/Z`. Job group `/X/Y` is not deleted because it is an explicitly created job group. Job group `/X` is also not deleted because it has an explicitly created child job group `/X/Y`.

Automatic job group deletion does not delete job groups that are attached to SLA service classes. Use **bgdel** to manually delete job groups that are attached to SLAs.

Host groups

Host groups gather similar resources to the same group of hosts (for example, all hosts with big memory)- Use host groups to manage dedicated resources for a single organization or to share resources across organizations. You can add limits to host groups, or define host groups in queues to constrain jobs for a scheduling policy that is defined over a specific set of hosts.

You can define a host group within LSF or use an external executable to retrieve host group members.

Use **bhosts** to view a list of existing hosts. Use **bmgroup** to view host group membership.

Where to use host groups

LSF host groups can be used in defining the following parameters in LSF configuration files:

- HOSTS in `lsb.queues` for authorized hosts for the queue
- HOSTS in `lsb.hosts` in the HostPartition section to list host groups that are members of the host partition
- [Configuring host groups](#)
- [Wildcard and special characters to define host names](#)
You can use special characters when defining host group members under the GROUP_MEMBER column to specify hosts. These are useful to define several hosts in a single entry, such as for a range of hosts, or for all host names with a certain text string.
- [Define condensed host groups](#)
You can define condensed host groups to display information for its hosts as a summary for the entire group. This is useful because it allows you to see the total statistics of the host group as a whole instead of having to add up the data yourself. This allows you to better plan the distribution of jobs submitted to the hosts and host groups in your cluster.

Configuring host groups

Procedure

1. Log in as the LSF administrator to any host in the cluster.
2. Open `lsb.hosts`.
3. Add the HostGroup section if it does not exist.

```

Begin HostGroup
GROUP_NAME      GROUP_MEMBER
groupA          (all)
groupB          (groupA ~hostA ~hostB)
groupC          (hostX hostY hostZ)
groupD          (groupC ~hostX)
groupE          (all ~groupC ~hostB)
groupF          (hostF groupC hostK)
desk_tops       (hostD hostE hostF hostG)
Big_servers     (!)
End HostGroup

```

4. Enter a group name under the GROUP_NAME column.
External host groups must be defined in the **egroup** executable.
5. Specify hosts in the GROUP_MEMBER column.
(Optional) To tell LSF that the group members should be retrieved using **egroup**, put an exclamation mark (!) in the GROUP_MEMBER column.
6. Save your changes.
7. Run **badadmin ckconfig** to check the group definition. If any errors are reported, fix the problem and check the configuration again.
8. Run **badadmin mbdrestart** to apply the new configuration.

Wildcard and special characters to define host names

You can use special characters when defining host group members under the GROUP_MEMBER column to specify hosts. These are useful to define several hosts in a single entry, such as for a range of hosts, or for all host names with a certain text string.

If a host matches more than one host group, that host is a member of all groups. If any host group is a condensed host group, the status and other details of the hosts are counted towards all of the matching host groups.

When defining host group members, you can use string literals and the following special characters:

- Tilde (~) excludes specified hosts or host groups from the list. The tilde can be used in conjunction with the other special characters listed below. The following example matches all hosts in the cluster except for `hostA`, `hostB`, and all members of the `groupA` host group:

```
... (all ~hostA ~hostB ~groupA)
```

- Asterisk (*) represent any number of characters. The following example matches all hosts beginning with the text string "hostC" (such as `hostCa`, `hostC1`, or `hostCZ1`):

```
... (hostC*)
```

- Square brackets with a hyphen (`[integer1 - integer2]`) or a colon (`[integer1 : integer2]`) define a range of non-negative integers at the end of a host name. The first integer must be less than the second integer. The following examples match all hosts from `hostD51` to `hostD100`:

```
... (hostD[51-100])
... (hostD[51:100])
```

- Square brackets with commas (`[integer1, integer2 ...]`) define individual non-negative integers at the end of a host name. The following example matches `hostD101`, `hostD123`, and `hostD321`:

```
... (hostD[101,123,321])
```

- Square brackets with commas and hyphens or colons (such as `[integer1 - integer2, integer3, integer4 : integer5]`) define different ranges of non-negative integers at the end of a host name. The following example matches all hosts from `hostD1` to `hostD100`, `hostD102`, all hosts from `hostD201` to `hostD300`, and `hostD320`:

```
... (hostD[1-100,102,201:300,320])
```


Restrictions

You cannot use more than one set of square brackets in a single host group definition.

The following example is *not* correct:

```
... (hostA[1-10]B[1-20] hostC[101-120])
```

The following example is correct:

```
... (hostA[1-20] hostC[101-120])
```

You cannot define subgroups that contain wildcard and special characters. The following definition for `groupB` is not correct because `groupA` defines hosts with a wildcard:

```
Begin HostGroup
GROUP_NAME    GROUP_MEMBER
groupA        (hostA*)
groupB        (groupA)
End HostGroup
```

Define condensed host groups

You can define condensed host groups to display information for its hosts as a summary for the entire group. This is useful because it allows you to see the total statistics of the host group as a whole instead of having to add up the data yourself. This allows you to better plan the distribution of jobs submitted to the hosts and host groups in your cluster.

To define condensed host groups, add a `CONDENSE` column to the `HostGroup` section. Under this column, enter `Y` to define a condensed host group or `N` to define a noncondensed host group, as shown in the following:

```
Begin HostGroup
GROUP_NAME    CONDENSE    GROUP_MEMBER
groupA        Y            (hostA hostB hostD)
groupB        N            (hostC hostE)
End HostGroup
```

The following commands display condensed host group information:

- **bhosts**
- **bhosts -w**
- **bjobs**
- **bjobs -w**

Use **bmgroup -l** to see whether host groups are condensed or not.

Hosts belonging to multiple condensed host groups

If you configure a host to belong to more than one condensed host group using wildcards, `bjobs` can display any of the host groups as execution host name.

For example, host groups `hg1` and `hg2` include the same hosts:

```
Begin HostGroup
GROUP_NAME    CONDENSE    GROUP_MEMBER    # Key words
hg1           Y            (host*)
hg2           Y            (hos*)
End HostGroup
```

Submit jobs using **bsub -m**:

```
bsub -m "hg2" sleep 1001
```

bjobs displays `hg1` as the execution host instead of `hg2`:

bjobs

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
520	user1	RUN	normal	host5	hg1	sleep 1001	Apr 15 13:50
521	user1	RUN	normal	host5	hg1	sleep 1001	Apr 15 13:50
522	user1	PEND	normal	host5		sleep 1001	Apr 15 13:51

Import external host groups (egroup)

When the membership of a host group changes frequently, or when the group contains a large number of members, you can use an external executable called **egroup** to retrieve a list of members rather than having to configure the group membership manually. You can write a site-specific egroup executable that retrieves host group names and the hosts that belong to each group. For information about how to use the external host and user groups feature, see [External host and user groups](#).

Specifying resource requirements

- [About resource requirements](#)
Resource requirements define which hosts a job can run on. Each job has its resource requirements and hosts that match the resource requirements are the candidate hosts. When LSF schedules a job, it uses the load index values of all the candidate hosts. The load values for each host are compared to the scheduling conditions. Jobs are only dispatched to a host if all load values are within the scheduling thresholds.
- [Queue-level resource requirements](#)
Each queue can define resource requirements that apply to all the jobs in the queue.
- [Job-level resource requirements](#)
Each job can specify resource requirements. Job-level resource requirements override any resource requirements specified in the remote task list.
- [Resource requirement strings](#)
Most LSF commands accept a `-R res_req` argument to specify resource requirements. A resource requirement string describes the resources that a job needs. The exact behavior depends on the command. LSF uses resource requirements to select hosts for remote execution and job execution. Resource requirement strings can be *simple* (applying to the entire job) or *compound* (applying to the specified number of slots).
- [Specify GPU resource requirements for your jobs](#)
Specify all GPU resource requirements as part of job submission, or in a queue or application profile, or use the default GPU requirement "num=1:mode=shared:mps=no;j_exclusive=no". Use the option `bsub -gpu` to submit jobs that require GPU resources. Specify how LSF manages GPU mode (exclusive or shared), and whether to enable the NVIDIA Multi-Process Service (MPS) for the GPUs used by the job.

About resource requirements

Resource requirements define which hosts a job can run on. Each job has its resource requirements and hosts that match the resource requirements are the candidate hosts. When LSF schedules a job, it uses the load index values of all the candidate hosts. The load values for each host are compared to the scheduling conditions. Jobs are only dispatched to a host if all load values are within the scheduling thresholds.

By default, if a job has no resource requirements, LSF places it on a host of the same type as the submission host (that is, `type==local`). However, if a job has string or Boolean resource requirements specified and the host type has not been specified, LSF places the job on any host (that is, `type==any`) that satisfies the resource requirements.

To override the LSF defaults, specify resource requirements explicitly. Resource requirements can be set for queues, for application profiles, or for individual jobs.

To best place a job with optimized performance, resource requirements can be specified for each application. This way, you do not have to specify resource requirements every time you submit a job. The LSF administrator may have already configured the resource requirements for your jobs, or you can put your executable name together with its resource requirements into your personal remote task list.

The **bsub** command automatically uses the resource requirements of the job from the remote task lists.

A resource requirement is an expression that contains resource names and operators.

Compound resource requirements

In some cases different resource requirements may apply to different parts of a parallel job. The first execution host, for example, may require more memory or a faster processor for optimal job scheduling. Compound resource requirements allow you to specify different requirements for some slots within a job in the queue-level, application-level, or job-level resource requirement string.

Compound resource requirement strings can be set by the application-level or queue-level `RES_REQ` parameter, or used with **bsub -R** when a job is submitted. **bmod -R** accepts compound resource requirement strings for pending jobs but not running jobs.

Special rules take effect when compound resource requirements are merged with resource requirements defined at more than one level. If a compound resource requirement is used at any level (job, application, or queue) the compound multi-level resource requirement combinations described later in this chapter apply.

The same resource requirement can be used within each component expression (simple resource requirement). For example, suppose static strings resource `res1` and `res2` are defined. We permit a resource requirement such as:

"4*{select[io] same[res1]} + 4*{select[compute] same[res1]}"

With this resource requirement, there are two simple sub-expressions, `R1` and `R2`. For each of these sub-expressions, all slots must come from hosts with equal values of `res1`. However, `R1` may occupy hosts of a different value than those occupied by `R2`.

You can specify a global `same` requirement that takes effect over multiple sub-expressions of a compound resource requirement string. For example,

"{4*{select[io]} + 4*{select[compute]}} same[res1]"

This syntax allows users to express that both sub-expressions must reside on hosts that have a common value for `res1`.

In general, there may be more than two sub-expressions in a compound resource requirement. The global `same` will apply to all of them.

Arbitrary nesting of brackets is not permitted. For example, you cannot have a global `same` apply to only two of three sub-expressions of a compound resource requirement. However, each sub-expression can have its own local `same` as well as a global `same` for the compound expression as a whole. For example, the following is permitted:

"{4*{same[res1]} + 4*{same[res1]}} same[res2]"

In addition, a compound resource requirement expression with a global `same` may be part of a larger alternative resource requirement string.

A compound resource requirement expression with a global `same` can be used in the following instances:

- Submitting a job: **bsub -R "rsrc_req_string" <other_bsub_options> a.out**
- Configuring application profile (lsb.applications): **RES_REQ = "rsrc_req_string"**
- Queue configuration (lsb.queues): **RES_REQ = "rsrc_req_string"**

Syntax:

- A single compound resource requirement:
"{ compound_rsrc_req } same[same_str]"
- A compound resource requirement within an alternative resource requirement:
"{{ compound_rsrc_req } same[same_str]} || {R}"
- A compound resource requirement within an alternative resource requirement with delay:
"{R} || {{ compound_rsrc_req } same[same_str]}@D"

where `D` is a positive integer.

Restriction:

- Compound resource requirements cannot contain the || operator. Compound resource requirements cannot be defined (included) in any multiple -R options.
- Compound resource requirements cannot contain the compute unit (cu) keywords balance or excl, but works normally with other cu keywords (including pref, type, maxcus, and usablecslots).
- Resizable jobs can have compound resource requirements, but only the portion of the job represented by the last term of the compound resource requirement is eligible for automatic resizing. When using **bresize release** to release slots, you can only release slots represented by the last term of the compound resource requirement. To release slots in earlier terms, run **bresize release** repeatedly to release slots in subsequent last terms.
- Compound resource requirements cannot be specified in the definition of a guaranteed resource pool.
- Resource allocation for parallel jobs using compound resources is done for each compound resource term in the order listed instead of considering all possible combinations. A host rejected for not satisfying one resource requirement term will not be reconsidered for subsequent resource requirement terms.
- Compound resource requirements were introduced in LSF Version 7 Update 5, and are not compatible with earlier versions of LSF.

Alternative resource requirements

In some circumstances more than one set of resource requirements may be acceptable for a job to be able to run. LSF provides the ability to specify alternative resource requirements.

An alternative resource requirement consists of two or more individual simple or compound resource requirements. Each separate resource requirement describes an alternative. When a job is submitted with alternative resource requirements, the alternative resource picked must satisfy the mandatory first execution host. If none of the alternatives can satisfy the mandatory first execution host, the job will PEND.

Alternative resource requirement strings can be specified at the application-level or queue-level RES_REQ parameter, or used with **bsub -R** when a job is submitted. **bmod -R** also accepts alternative resource requirement strings for pending jobs.

The rules for merging job, application, and queue alternative resource requirements are the same as for compound resource requirements.

Alternative resource requirements cannot be used with the following features:

- Multiple **bsub -R** commands
- TS jobs, including those with the **tssub** command
- Hosts from HPC integrations that use toplib, including CPUset and Blue Gene.
- Compute unit (cu) sections specified with balance or excl keywords.

If a job with alternative resource requirements specified is re-queued, it will have all alternative resource requirements considered during scheduling. If a @D delay time is specified, it is interpreted as waiting, starting from the original submission time. For a restart job, @D delay time starts from the restart job submission time.

Resource requirements in application profiles

See [Resource requirements](#) for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

Re-sizable jobs and resource requirements

In general, resize allocation requests for re-sizable jobs use the resource requirements of the running job. When the resource requirement string for a job is modified with **bmod -R**, the new string takes effects for a job resize request. The resource requirement of the allocation request is merged from resource requirements specified at the queue, job, and application levels.

Queue-level resource requirements

Each queue can define resource requirements that apply to all the jobs in the queue.

When resource requirements are specified for a queue, and no job-level or application profile resource requirement is specified, the queue-level resource requirements become the default resource requirements for the job.

Resource requirements determined by the queue no longer apply to a running job after running **badmin reconfig**. For example, if you change the RES_REQ parameter in a queue and reconfigure the cluster, the previous queue-level resource requirements for running jobs are lost.

Syntax

The condition for dispatching a job to a host can be specified through the queue-level RES_REQ parameter in the queue definition in `lsb.queues`. Queue-level RES_REQ rusage values must be in the range set by RESRSV_LIMIT (set in `lsb.queues`), or the queue-level RES_REQ is ignored.

Examples

```
RES_REQ=select[ ((type==LINUX2.4 && rlm < 2.0) || (type==AIX && rlm < 1.0)) ]
```

This allows a queue, which contains LINUX2.4 and AIX hosts, to have different thresholds for different types of hosts.

```
RES_REQ=select[ (hname==hostA && mem > 50) || (hname==hostB && mem > 100) ]
```

Using the `hname` resource in the resource requirement string allows you to set up different conditions for different hosts in the same queue.

Load thresholds

Load thresholds can be configured by your LSF administrator to schedule jobs in queues. Load thresholds specify a load index value.

loadSched

The scheduling threshold that determines the load condition for dispatching pending jobs. If a host's load is beyond any defined `loadSched`, a job is not started on the host. This threshold is also used as the condition for resuming suspended jobs.

loadStop

The suspending condition that determines when running jobs should be suspended.

Thresholds can be configured for each queue, for each host, or a combination of both. To schedule a job on a host, the load levels on that host must satisfy both the thresholds configured for that host and the thresholds for the queue from which the job is being dispatched.

The value of a load index may either increase or decrease with load, depending on the meaning of the specific load index. Therefore, when comparing the host load conditions with the threshold values, you need to use either greater than (>) or less than (<), depending on the load index.

Job-level resource requirements

Each job can specify resource requirements. Job-level resource requirements override any resource requirements specified in the remote task list.

In some cases, the queue specification sets an upper or lower bound on a resource. If you attempt to exceed that bound, your job will be rejected.

Syntax

To specify resource requirements for your job, use **bsub -R** and specify the resource requirement string as usual. You can specify multiple -R order, same, rusage, and select sections.

Note:

Within `esub`, you can get resource requirements using the `LSB_SUB_RES_REQ` variable, which merges multiple `-R` from the **`bsub`** command. If you want to modify the `LSB_SUB_RES_REQ` variable, you cannot use multiple `-R` format. Instead, use the `&&` operator to merge them manually.

Merged `RES_REQ` usage values from the job and application levels must be in the range of `RESRSV_LIMIT` (set in `lsb.queues`), or the job is rejected.

Examples

```
bsub -R "swp > 15 && hpux order[ut]" myjob
```

or

```
bsub -R "select[swp > 15]" -R "select[hpux] order[ut]" myjob
```

This runs `myjob` on an HP-UX host that is lightly loaded (CPU utilization) and has at least 15 MB of swap memory available.

```
bsub -R "select[swp > 15]" -R "select[hpux] order[r15m]" -R "order[r15m]" -R rusage[mem=100]"  
-R "order[ut]" -R "same[type]" -R "rusage[tmp=50:duration=60]" -R "same[model]" myjob
```

LSF merges the multiple `-R` options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

Resource requirement strings

Most LSF commands accept a `-R res_req` argument to specify resource requirements. A resource requirement string describes the resources that a job needs. The exact behavior depends on the command. LSF uses resource requirements to select hosts for remote execution and job execution. Resource requirement strings can be *simple* (applying to the entire job) or *compound* (applying to the specified number of slots).

For example, specifying a resource requirement for the **`lsload`** command displays the load levels for all hosts that have the requested resources. Specifying resource requirements for the **`lsrun`** command causes LSF to select the best host out of the set of hosts that have the requested resources.

- **[Selection string](#)**
The selection string specifies the characteristics that a host must have to match the resource requirement. It is a logical expression that is built from a set of resource names. The selection string is evaluated for each host; if the result is non-zero, then that host is selected. When used in conjunction with a `cu` string, hosts not belonging to compute unit are not considered.
- **[Order string](#)**
The order string allows the selected hosts to be sorted according to the values of resources. The values of `r15s`, `r1m`, and `r15m` used for sorting are the normalized load indices that are returned by **`lsload -N`**.
- **[Usage string](#)**
This string defines the expected resource usage of the job. It is used to specify resource reservations for jobs, or for mapping jobs on to hosts and adjusting the load when running interactive jobs.
- **[Span string](#)**
A span string specifies the locality of a parallel job. If span is omitted, LSF allocates the required processors for the job from the available set of processors.
- **[Same string](#)**
- **[Compute unit string](#)**
A `cu` string specifies the network architecture-based requirements of parallel jobs. `cu` sections are accepted by **`bsub -R`**, and by **`bmod -R`** for non-running jobs.
- **[Affinity string](#)**
An *affinity resource requirement* string specifies CPU and memory binding requirements for the tasks of jobs. An `affinity[]` resource requirement section controls CPU and memory resource allocations and specifies the distribution of *processor units* within a host according to the hardware topology information that LSF collects.

Resource requirement string sections

The section names for resource requirement strings are `select`, `order`, `rusage`, `span`, `same`, `cu`, and `affinity`. Sections that do not apply for a command are ignored.

- A selection section (`select`). The selection section specifies the criteria for selecting hosts from the system.
- An ordering section (`order`). The ordering section indicates how the hosts that meet the selection criteria is sorted.
- A resource usage section (`rusage`). The resource usage section specifies the expected resource consumption of the task.
- A job spanning section (`span`). The job spanning section indicates whether a parallel batch job can span across multiple hosts.
- A same resource section (`same`). The same section indicates that all processes of a parallel job must run on the same type of host.
- A compute unit resource section (`cu`). The `cu` section specifies how a job is placed compared to the underlying network architecture.
- An affinity resource section (`affinity`). The affinity section specifies how a job is be placed compared to CPU and memory affinity on NUMA hosts.

Which sections apply

Depending on the command, one or more of the resource requirement string sections might apply.

- The **bsub** command uses all sections.
- The **brsvadd** command uses the information in the `select` and `same` sections to select an appropriate host for an advance reservation.
- The **lshosts** command selects hosts, but does not order them.
- The **lsload** command selects and orders hosts.
- The **lsloadadj** command uses the `rusage` section to determine how the load information is adjusted on a host.
- The **lsplace** command uses the information in the `select`, `order`, and `rusage` sections to select an appropriate host for a task.

Simple syntax

```
select[selection_string] order[order_string] rusage[usage_string [, usage_string]
[|| usage_string] ...] span[span_string] same[same_string] cu[cu_string]
affinity[affinity_string]
```

With the **bsub** and **bmod** commands, and only with these commands, you can specify multiple `-R` `order`, `same`, `rusage`, and `select` sections. The **bmod** command does not support the use of the `||` operator.

The section names are `select`, `order`, `rusage`, `span`, `same`, `cu`, and `affinity`. Sections that do not apply for a command are ignored.

The square brackets must be typed as shown for each section. A blank space must separate each resource requirement section.

You can omit the `select[]` section, but if you include it, the selection section must be the *first* string in the resource requirement string. If you do not use a section keyword (`select`, `order`, `rusage`, `span`, `same`, `cu`, and `affinity`), the first resource requirement string is treated as a selection string (`select[selection_string]`).

Each section has a different syntax.

By default, memory (`mem`) and swap (`swp`) limits in `select[]` and `rusage[]` sections are specified in MB. Use the `LSF_UNIT_FOR_LIMITS` parameter in the `lsf.conf` file to specify a larger unit for these limits.

For the **bsub**, **bmod**, and **brestart** commands, you can use the following units for resource requirements and limits:

- KB or K (kilobytes)
- MB or M (megabytes)
- GB or G (gigabytes)
- TB or T (terabytes)
- PB or P (petabytes)
- EB or E (exabytes)
- ZB or Z (zettabytes)

The specified unit is converted to the appropriate value specified by the LSF_UNIT_FOR_LIMITS parameter. The converted limit values round up to a positive integer. For resource requirements, you can specify unit for `mem`, `swp`, and `tmp` in the `select` and `rusage` sections.

By default, the `tmp` resource is not supported by the LSF_UNIT_FOR_LIMITS parameter. Use the parameter `LSF_ENABLE_TMP_UNIT=Y` to enable the LSF_UNIT_FOR_LIMITS parameter to support limits on the `tmp` resource.

If the `LSF_ENABLE_TMP_UNIT=Y` and `LSF_UNIT_FOR_LIMIT=GB` parameters are set, the following conversion happens.

```
bsub -C 500MB -M 1G -S 1TB -F 1GB -R "rusage[mem=512MB:swp=1GB:tmp=1TB]" sleep 100
```

The units in this job submission are converted to the following units:

```
bsub -C 1 -M 1 -S 1024 -F 1 -R "rusage[mem=0.5:swp=1:tmp=1024]" sleep 100
```

Compound syntax

```
num1*{simple_string1} + num2*{simple_string2} + ...
```

where *numx* is the number of slots that are affected and *simple_stringx* is a simple resource requirement string with the syntax:

```
select[selection_string] order[order_string] rusage[usage_string [, usage_string]...]
span[span_string]
```

Resource requirements that apply to the first execution host (if used) appear in the first compound term *num1**{*simple_string1*}.

Place specific (harder to fill) requirements before general (easier to fill) requirements since compound resource requirement terms are considered in the order they appear. Resource allocation for parallel jobs that use compound resources is done for each compound resource term independently instead of considering all possible combinations.

Note: A host that is rejected for not satisfying one resource requirement term is not reconsidered for subsequent resource requirement terms.

For jobs without the number of total slots that are specified by using the **bsub -n** option, you can omit the final *numx*. The final resource requirement is then applied to the zero or more slots that are not yet accounted for using the default slot setting of the parameter TASKLIMIT as follows:

- (final res_req number of slots) = MAX(0,(default number of job slots from TASKLIMIT)-(num1+num2+...))

For jobs with the total number of slots that are specified with the **bsub -n num_slots** option, the total number of slots must match the number of slots in the resource requirement:

- *num_slots*=(*num1*+*num2*+*num3*+...)

You can omit the final *numx*.

For jobs with compound resource requirements and first execution host candidates that are specified by using the **bsub -m** option, the host that is allocated first must satisfy the simple resource requirement string that appears first in the compound resource requirement. The first execution host must satisfy the requirements in *simple_string1* for the following compound resource requirement:

```
num1*{simple_string1} +
num2*{simple_string2} +
num3*{simple_string3}
```

Compound resource requirements do not support use of the `||` operator within the component `rusage` simple resource requirements, or use of the `cu` section.

How simple multi-level resource requirements are resolved

Simple resource requirements can be specified at the job, application, and queue levels.

When none of the resource requirements are compound, requirements that are defined at different levels are resolved in the following ways:

- In a select string, a host must satisfy *all* queue-level, application-level, and job-level requirements for the job to be dispatched.

- In a same string, all queue-level, application-level, and job-level requirements are combined before the job is dispatched.
- The order, span, and cu sections that are defined at the job level overwrite the sections that are defined at the application level or queue level. The order, span, and cu sections that are defined at the application level overwrite the sections that are defined at the queue level. The default order string is `r15s:pg`.
- For usage strings, the `rusage` section that is defined for the job overrides the `rusage` section that is defined in the application. The two `rusage` definitions are merged. The job-level `rusage` takes precedence. Similarly, `rusage` strings that are defined for the job or application are merged with queue-level strings, with the job and then application definitions taking precedence over the queue if there is any overlap.

Section	Simple resource requirement multilevel behavior
select	All levels are satisfied
same	All levels are combined
order span cu	Job-level section overwrites application-level section, which overwrites queue-level section (if a level is present)
rusage	All levels merge If conflicts occur the job-level section overwrites the application-level section, which overwrites the queue-level section.

For internal load indices and duration, jobs are rejected if the merged job-level and application-level resource reservation requirements exceed the requirements that are specified at the queue level.

Note: If a compound resource requirement is used at one or more levels (job, application, or queue) the compound rules apply.

How compound and multi-level resource requirements are resolved

Compound resource requirements can be specified at the job, application, and queue levels. When one or more of the resource requirements is compound or alternative, requirements at different levels are resolved depending on where the compound resource requirement appears.

During the first stage, LSF decides between the job and application level resource requirement:

1. If a resource requirement is not defined at the job level, LSF takes the application level resource requirement, if any.
2. If any level defines an alternative resource requirement, the job level overrides the application level resource requirement as a whole. There is no merge.
3. If both levels have simple resource requirements, the job level merges with the application level resource requirement.

During the second stage, LSF decides between the job/application merged result and the queue level resource requirement:

1. If the merged result does not define any resource requirement, LSF takes the queue-level resource requirement.
2. If the merged result or queue-level is an alternative resource requirement, LSF takes the merged result.
3. If the queue-level is a simple resource requirement and the merged result is a simple resource requirement, LSF merges the merged result with the queue-level resource requirement.
4. If the queue-level resource requirement is simple and the merged result is an alternative resource requirement, each sub expression in the alternative resource requirement merges with the queue-level resource requirement, following these rules:
 - a. The `select[]` clause must be satisfied for all of them.
 - b. The merged `order[]` clause overrides the queue-level clause.
 - c. The merged `rusage[]` clause merges with the queue-level `rusage`. If the queue-level `rusage` defines a job-level resource, this `rusage` subterm is merged only into the left most atomic resource requirement term.
 - d. The merged `span[]` clause overrides the queue-level `span[]` clause.
 - e. Queue-level `same[]` and `cu[]` clauses are ignored.

For internal load indices and duration, jobs are rejected if they specify resource reservation requirements that exceed the requirements that are specified at the application level or queue level.

Note: If a compound resource requirement is used at one or more levels (job, application, or queue) the compound rules apply.

Compound queue level

When a compound resource requirement is set for a queue, it is ignored unless it is the only resource requirement specified (no resource requirements are set at the job level or application level).

Compound application level

When a compound resource requirement is set at the application level, it is ignored if any job-level resource requirements (simple or compound) are defined.

If no job-level resource requirements are set, the compound application-level requirements interact with queue-level resource requirement strings in the following ways:

- If no queue-level resource requirement is defined or a compound queue-level resource requirement is defined, the compound application-level requirement is used.
- If a simple queue-level requirement is defined, the application-level and queue-level requirements combine as follows:

Section	Compound application and simple queue behavior
select	Both levels are satisfied. Queue requirement applies to all compound terms.
same	Queue level is ignored.
order and span	Application-level section overwrites queue-level section (if a level is present). Queue requirement (if used) applies to all compound terms.
rusage	<ul style="list-style-type: none"> • Both levels merge. • Queue requirement if a job-based resource is applied to the first compound term, otherwise applies to all compound terms. • If conflicts occur, the application-level section overwrites the queue-level section. <p>For example, if the application-level requirement is $\text{num1} * \{\text{rusage}[\text{R1}]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$ and the queue-level requirement is $\text{rusage}[\text{RQ}]$ where RQ is a job-based resource, the merged requirement is $\text{num1} * \{\text{rusage}[\text{merge}(\text{R1}, \text{RQ})]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$</p>

Compound job level

When a compound resource requirement is set at the job level, any simple or compound application-level resource requirements are ignored, and any compound queue-level resource requirements are ignored.

If a simple queue-level requirement appears with a compound job-level requirement, the requirements interact as follows:

Section	Compound job and simple queue behavior
select	Both levels are satisfied; queue requirement applies to all compound terms.
same	Queue level section is ignored.
order and span	Job-level section overwrites queue-level section (if a level is present). Queue requirement (if used) applies to all compound terms.
rusage	<ul style="list-style-type: none"> • Both levels merge. • Queue requirement if a job-based resource is applied to the first compound term, otherwise applies to all compound terms. • If conflicts occur, the job-level section overwrites the queue-level section. <p>For example, if the job-level requirement is $\text{num1} * \{\text{rusage}[\text{R1}]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$ and the queue-level requirement is $\text{rusage}[\text{RQ}]$ where RQ is a job resource, the merged requirement is $\text{num1} * \{\text{rusage}[\text{merge}(\text{R1}, \text{RQ})]\} + \text{num2} * \{\text{rusage}[\text{R2}]\}$</p>

Example 1

A compound job requirement and simple queue requirement are specified.

Job level

```
2*{select[type==X86_64] rusage[licA=1] span[hosts=1]} + 8*{select[type==any]}
```

Application level

Not defined.

Queue level

```
rusage[perslot=1]
```

The final job scheduling resource requirement merges the simple queue-level rusage section into each term of the compound job-level requirement, resulting in the following resource requirement: `2*{select[type==X86_64] rusage[licA=1:perslot=1] span[hosts=1]} + 8*{select[type==any] rusage[perslot=1]}`

Example 2

A compound job requirement and compound queue requirement are specified.

Job level

```
2*{select[type==X86_64 && tmp>10000] rusage[mem=1000] span[hosts=1]} + 8*
{select[type==X86_64]}
```

Application level

Not defined.

Queue level

```
2*{select[type==X86_64] rusage[mem=1000] span[hosts=1]} +8*{select[type==X86_64]}
```

The final job scheduling resource requirement ignores the compound queue-level requirement, resulting in the following resource requirement: `2*{select[type==X86_64 && tmp>10000] rusage[mem=1000] span[hosts=1]} + 8*{select[type==X86_64]}`

Example 3

A compound job requirement and simple queue requirement where the queue requirement is a job-based resource.

Job level

```
2*{select[type==X86_64]} + 2*{select[mem>1000]}
```

Application level

Not defined.

Queue level

```
rusage[licA=1]. The resource licA=1 is job-based.
```

The queue-level requirement is added to the first term of the compound job-level requirement, resulting in the following resource requirement: `2*{select[type==X86_64] rusage[licA=1]} + 2*{select[mem>1000]}`

Example 4

Compound multi-phase job requirements and simple multi-phase queue requirements.

Job level

```
2*{rusage[mem=(400 350):duration=(10 15):decay=(0 1)]} + 2*
{rusage[mem=300:duration=10:decay=1]}
```

Application level

Not defined.

Queue level

```
rusage[mem=(500 300):duration=(20 10):decay=(0 1)]
```

The queue-level requirement is overridden by the first term of the compound job-level requirement, resulting in the following resource requirement: `2*{rusage[mem=(400 350):duration=(10 15):decay=(0 1)]} + 2*{rusage[mem=300:duration=10:decay=1]}`

How alternative resource requirements are resolved

Alternative resource requirements are resolved in two stages. During the first stage, LSF decides between the job and application level resource requirement. During the second stage, LSF decides between the job/application merged result and the queue level resource requirement.

LSF makes the following decisions in the first stage:

1. If a resource requirement is not defined at the job level, LSF takes the application-level resource requirement, if any.
2. If any level defines an alternative resource requirement, the job-level overrides the application level resource requirement as a whole. There is no merge.
3. If both levels have simple resource requirements, the job level merges with the application level resource requirement.

LSF makes the following decisions in the second stage:

1. If the merged result does not define any resource requirement, LSF takes the queue-level resource requirement.
2. If the merged result and queue-level resource requirement is an alternative resource requirement, LSF takes the merged result.
3. If the queue-level is a simple resource requirement and the merged result is a simple resource requirement, LSF merges the merged result with the queue-level resource requirement.
4. If the queue-level resource requirement is simple and the merged result is an alternative resource requirement, each sub expression in the alternative resource requirement merges with the queue-level resource requirement, following these rules:
 - a. The select[] clause must be satisfied for all of them.
 - b. The merged order[] clause overrides the queue-level clause.
 - c. The merged rusage[] clause is merged with the queue-level rusage. When the subterm of the alternative resource requirement is a compound resource requirement, and the queue-level defines a job-level resource, this rusage section is merged only into the left-most atomic resource requirement term of this subterm. Otherwise, it is merged into all the terms for this subterm.
 - d. The merged span[] clause overrides the queue-level span[] clause.
 - e. The queue-level same[] and cu[] clauses are ignored.

After the job is submitted, the pending reason that is given applies only to the first alternative even though LSF is trying the other applicable alternatives.

Combined resource requirements

The combined resource requirement is the result of the **mbatchd** daemon merging job, application, and queue level resource requirements for a job.

Effective resource requirements

The effective resource requirement always represents the job's allocation. The effective resource requirement string for scheduled jobs represents the resource requirement that is used by the scheduler to make a dispatch decision. When a job is dispatched, the **mbschd** daemon generates the effective resource requirement for the job from the combined resource requirement according to the job's real allocation.

After the job starts, you can use the **bmod -R** command to modify the job's effective resource requirement along with the job allocation. The rusage section of the effective resource is updated with the rusage in the newly combined resource requirement. The other sections in the resource requirement string such as select, order, and span are kept the same during job run time because they are still used for the job by the scheduler.

For started jobs, you can modify only simple effective resource requirements with another simple requirement. Any request to change effective resource requirements to compound or alternative resource requirements are rejected. Attempting to modify the resource requirement of a running job to use rusage with or (|) branches are also rejected.

By default, LSF does not modify effective resource requirements and job resource usage when it runs the **bswitch** command. However, you can set the BSWITCH_MODIFY_RUSAGE parameter to **Y** to make the **bswitch** command update job resource usage according to the resource requirements in the new queue.

When a job finishes, the effective resource requirement last used by the job is saved in the JOB_FINISH event record of the lsb.acct file and the JOB_FINISH2 record of the lsb.stream file. The **bjobs -l** command always displays the effective resource requirement that is used by the job in the resource requirement details.

Selection string

The selection string specifies the characteristics that a host must have to match the resource requirement. It is a logical expression that is built from a set of resource names. The selection string is evaluated for each host; if the result is non-zero, then that host is selected. When used in conjunction with a cu string, hosts not belonging to compute unit are not considered.

Syntax

The selection string can combine resource names with logical and arithmetic operators. Non-zero arithmetic values are treated as logical TRUE, and zero as logical FALSE. Boolean resources (for example, server to denote LSF server hosts) have a value of

one if they are defined for a host, and zero if they are not defined for the host.

The resource names swap, idle, login, and cpu are accepted as aliases for swp, it, ls, and r1m respectively.

The ut index measures CPU utilization, which is the percentage of time spent running system and user code. A host with no processes running has a ut value of 0 percent; a host on which the CPU is completely loaded has a ut of 100 percent. You must specify ut as a floating-point number between 0.0 and 1.0.

For the string resources type and model, the special value any selects any value and local selects the same value as that of the local host. For example, type==local selects hosts of the same type as the host submitting the job. If a job can run on any type of host, include type==any in the resource requirements.

If no type is specified, the default depends on the command. For **bhosts**, **bsub**, **lsplace**, **lsrun**, and **lsgrun** the default is type==local unless a string or Boolean resource is specified, in which case it is type==any. For **lshosts** and **lsload**, the default is type==any.

Tip: When PARALLEL_SCHED_BY_SLOT=Y in lsb.params, the resource requirement string keyword ncpus refers to the number of slots instead of the number of CPUs, however **lshosts** output will continue to show ncpus as defined by EGO_DEFINE_NCPUS in lsf.conf.

You can also filter hosts by using **slots** or **maxslots** in the select string of resource requirements. For example:

```
select[slots>4 && maxslots < 10 || mem > 10] order[-slots:maxslots:maxmem:ut]
```

Specify multiple -R options

bsub accepts multiple -R options for the select section in simple resource requirements.

Restriction: Compound resource requirements do not support multiple -R options.

You can specify multiple resource requirement strings instead of using the && operator. For example:

```
bsub -R "select[swp > 15]" -R "select[hpx]"
```

LSF merges the multiple -R options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

You cannot specify more than one select section in the same -R option. Use the logical and (&&) operator to specify multiple selection strings in the same select section. For example, the following command submits a job called `myjob` to run on a host that has more than 15 MB of swap space available, and maximum RAM larger than 100 MB. The job is expected to reserve 100 MB memory on the host:

```
% bsub -R "select [swp > 15 && maxmem > 100] rusage[mem = 100]" myjob
```

The number of -R option sections is unlimited.

Select shared string resources

You must use single quote characters (') around string-type shared resources. For example, use **lsload -s** to see the shared resources that are defined for the cluster:

RESOURCE	VALUE	LOCATION
os_version	4.2	pc36
os_version	4.0	pc34
os_version	4.1	devlinux4
cpu_type	ia	pc36
cpu_type	ia	pc34
cpu_type	unknown	devlinux4

Use a select string in **lsload -R** to specify the shared resources you want to view, enclosing the shared resource values in single quotes. For example:

```
lsload -R "select[os_version=='4.2' || cpu_type=='unknown']"
HOST_NAME      status  r15s  r1m  r15m  ut  pg  ls  it  tmp  swp  mem
```

pc36	ok	0.0	0.2	0.1	1%	3.4	3	0	895M	517M	123M
devlinux4	ok	0.0	0.1	0.0	0%	2.8	4	0	6348M	504M	205M

Note: When reserving resources based on host status (**bsub -R "status==ok"**), the host status must be the one displayed by running **bhosts** not **lsload**.

Operators

These operators can be used in selection strings. The operators are listed in order of decreasing precedence.

Syntax	Meaning
(a)	When LSF_STRICT_RESREQ=Y is configured in lsf.conf, an expression between parentheses has higher priority than other operators.
-a	Negative of a
!a	Logical not: 1 if a==0, 0 otherwise
a * b	Multiply a and b
a / b	Divide a by b
a + b	Add a and b
a - b	Subtract b from a
a > b	1 if a is greater than b, 0 otherwise
a < b	1 if a is less than b, 0 otherwise
a >= b	1 if a is greater than or equal to b, 0 otherwise
a <= b	1 if a is less than or equal to b, 0 otherwise
a == b	1 if a is equal to b, 0 otherwise
a != b	1 if a is not equal to b, 0 otherwise
a && b	Logical AND: 1 if both a and b are non-zero, 0 otherwise
a b	Logical OR: 1 if either a or b is non-zero, 0 otherwise

Examples

```
select[(swp > 50 && type == x86_64) || (swp > 35 && type == LINUX)]
select[((2*r15s + 3*r1m + r15m) / 6 < 1.0) && !fs && (cpuf > 4.0)]
```

Specify shared resources with the keyword “defined”

A shared resource may be used in the resource requirement string of any LSF command. For example, when submitting an LSF job that requires a certain amount of shared scratch space, you might submit the job as follows:

```
bsub -R "avail_scratch > 200 && swap > 50" myjob
```

The above assumes that all hosts in the cluster have access to the shared scratch space. The job is only scheduled if the value of the "avail_scratch" resource is more than 200 MB and goes to a host with at least 50 MB of available swap space.

It is possible for a system to be configured so that only some hosts within the LSF cluster have access to the scratch space. To exclude hosts that cannot access a shared resource, the defined(*resource_name*) function must be specified in the resource requirement string.

For example:

```
bsub -R "defined(avail_scratch) && avail_scratch > 100 && swap > 100" myjob
```

would exclude any hosts that cannot access the scratch resource. The LSF administrator configures which hosts do and do not have access to a particular shared resource.

Supported resource names in the defined function

Only resource names configured in lsf.shared, *except* dynamic NUMERIC resource names with INTERVAL fields defined are accepted as the argument in the defined (*resource_name*) function.

The following resource names are *not* accepted in the defined (*resource_name*) function:

- The following built-in resource names:

```
r15s r1m r15m ut pg io ls it tmp swp mem ncpus ndisks maxmem  
maxswp maxtmp cpuf type model status rexpri server and hname
```

- Dynamic NUMERIC resource names configured in lsf.shared with INTERVAL fields defined. In the default configuration, these are mode, cntrl, it_t.)
- Other non-built-in resource names not configured in lsf.shared.

Specify exclusive resources

An exclusive resource may be used in the resource requirement string of any placement or scheduling command, such as **bsub**, **lsplace**, **lsrun**, or **lsgrun**. An exclusive resource is a special resource that is assignable to a host. This host will not receive a job unless that job explicitly requests the host. For example, use the following command to submit a job requiring the exclusive resource bigmem:

```
bsub -R "bigmem" myjob
```

Jobs will not be dispatched to the host with the **bigmem** resource unless the command uses the **-R** option to explicitly specify "bigmem".

To configure an exclusive resource, first define a static Boolean resource in lsf.shared. For example:

```
Begin Resource  
...  
bigmem Boolean () ()  
End Resource
```

Assign the resource to a host in the Host section of lsf.cluster.cluster_name for static hosts or LSF_LOCAL_RESOURCES for dynamic hosts. Prefix the resource name with an exclamation mark (!) to indicate that the resource is exclusive to the host. For example:

```
Begin Host  
HOSTNAME  model  type  server r1m pg tmp RESOURCES  RUNWINDOW  
...  
hostE     !      !      1      3.5 () ()  (linux !bigmem) ()  
...  
End Host  
  
LSF_LOCAL_RESOURCES="[resource linux] [!bigmem]"
```

Strict syntax for resource requirement selection strings

When LSF_STRICT_RESREQ=Y is configured in lsf.conf, resource requirement strings in select sections must conform to a more strict syntax. The strict resource requirement syntax only applies to the select section. It does not apply to the other resource requirement sections (order, rusage, same, span, or cu). When LSF_STRICT_RESREQ=Y in lsf.conf, LSF rejects resource requirement strings where an rusage section contains a non-consumable resource.

Strict select string syntax usage notes

The strict syntax is case-sensitive.

Boolean variables, such as fs, hpux, cs, can only be computed with the following operators

```
&& || !
```

String variables, such as type, can only be computed with the following operators:

```
= == != < > <= >=
```

For function calls, blanks between the parentheses "()" and the resource name are not valid. For example, the following is not correct:

```
defined( mg )
```

Multiple logical NOT operators (!) are not valid. For example, the following is not correct:

```
!!mg
```

The following resource requirement is valid:

```
!(!mg)
```

At least one blank space must separate each section. For example, the following are correct:

```
type==any rusage[mem=1024]
select[type==any] rusage[mem=1024]
select[type==any] rusage[mem=1024]
```

but the following is not correct:

```
type==anyrusage[mem=1024]
```

Only a single select section is supported by the stricter syntax. The following is not supported in the same resource requirement string:

```
select[mem>0] select[maxmem>0]
```

Escape characters (like '\n') are not supported in string literals.

A colon (:) is not allowed inside the select string. For example, `select[mg:bigmem]` is not correct.

inf and nan can be used as resource names or part of a resource name.

Single or double quotes are only supported around the whole resource requirement string, not within the square brackets containing the selection string. For example, in `lsb.queues`, `RES_REQ='swp>100'` and `RES_REQ="swp>100"` are correct. Neither `RES_REQ=select['swp>100']` nor `RES_REQ=select["swp>100"]` are supported.

The following are correct **bsub** command-level resource requirements:

- `bsub -R "'swp>100'"`
- `bsub -R '"swp>100"'`

The following are not correct:

- `bsub -R "select['swp>100']"`
- `bsub -R 'select["swp>100"]'`

Some incorrect resource requirements are no longer silently ignored. For example, when `LSF_STRICT_RESREQ=Y` is configured in `lsf.conf`, the following are rejected by the resource requirement parser:

- `microcs73` is rejected:

```
linux rusage[mem=16000] microcs73
```

- `select[AMD64]` is rejected:

```
mem < 16384 && select[AMD64]
```

- `linux` is rejected:

```
rusage[mem=2000] linux
```

- Using a colon (:) to separate select conditions, such as `linux:qscw`.
- The restricted syntax of resource requirement select strings that are described in the `lsfintro(1)` man page is not supported.

Explicit and implicit select sections

An explicit select section starts from the section keyword and ends at the begin of next section, for example: the select section is `select[selection_string]`. An implicit select section starts from the first letter of the resource requirement string and ends at the end of the string if there are no other resource requirement sections. If the resource requirement has other sections, the implicit select section ends before the first letter of the first section following the selection string.

All explicit sections must begin with a section keywords (select, order, span rusage, or same). The resource requirement content is contained by square brackets ([) and (]).

An implicit select section must be the first resource requirement string in the whole resource requirement specification. Explicit select sections can appear after other sections. A resource requirement string can have only one select section (either an explicit select section or an implicit select section). A section with an incorrect keyword name is not a valid section.

An implicit select section must have the same format as the content of an explicit select section. For example, the following commands are correct:

- `bsub -R "select[swp>15] rusage[mem=100]" myjob`
- `bsub -R "swp > 15 rusage[mem=100]" myjob`
- `bsub -R "rusage[mem=100] select[swp >15]" myjob`

Examples

The following examples illustrate some correct resource requirement select string syntax.

- `bsub -R "(r15s * 2 + r15m) < 3.0 && !(type == IBMAIX4) || fs" myjob`
- If swap space is equal to 0, the following means TRUE; if swap space is not equal to 0, it means FALSE:

```
bsub -R "!swp" myjob
```

- Select hosts of the same type as the host submitting the job:

```
bsub -R "type == local" myjob
```

- Select hosts that are not the same type as the host submitting the job:

```
bsub -R "type != local" myjob
```

- `bsub -R "r15s < 1.0 || model ==local && swp <= 10" myjob`
Since && has a higher priority than ||, this example means:

```
r15s < 1.0 || (model == local && swp <=10)
```

- This example has different meaning from the previous example:

```
bsub -R "(r15s < 1.0 || model == local) && swp <= 10" myjob
```

This example means:

```
(r15s < 1.0 || model == local) && swp <= 10
```

Check resource requirement syntax

Use the `BSUB_CHK_RESREQ` environment variable to check the compatibility of your existing resource requirement select strings against the stricter syntax enabled by `LSF_STRICT_RESREQ=Y` in `lsf.conf`.

Set the `BSUB_CHK_RESREQ` environment variable to any value enable **bsub** to check the syntax of the resource requirement selection string without actually submitting the job for scheduling and dispatch. `LSF_STRICT_RESREQ` does not need to be set to check the resource requirement selection string syntax.

bsub only checks the select section of the resource requirement. Other sections in the resource requirement string are not checked.

If resource requirement checking detects syntax errors in the selection string, **bsub** returns an error message. For example:

```
bsub -R "select[type==local] select[hname=abc]" sleep 10
Error near "select": duplicate section. Job not submitted.
echo $?
255
```

If no errors are found, **bsub** returns a successful message and exit code zero. For example:

```
env | grep BSUB_CHK_RESREQ
BSUB_CHK_RESREQ=1
bsub -R "select[type==local]" sleep 10
Resource requirement string is valid.
```

```
echo $?  
0
```

If `BSUB_CHK_RESREQ` is set, but you do not specify `-R`, LSF treats it as empty resource requirement. For example:

```
bsub sleep 120  
Resource requirement string is valid.  
echo $?  
0
```

Resizable jobs

Resize allocation requests are scheduled using hosts as determined by the select expression of the merged resource requirement. For example, to run an autoresizable job on 1-100 slots, but only on hosts of type `X86_64`, the following job submission specifies this resource request:

```
bsub -ar -app <applicaion_file> -n "1,100" -R "rusage[swp=100,license=1]" myjob
```

Every time the job grows in slots, slots are requested on hosts of the specified type.

Note: Resizable jobs cannot have compound or alternative resource requirements.

Order string

The order string allows the selected hosts to be sorted according to the values of resources. The values of `r15s`, `r1m`, and `r15m` used for sorting are the normalized load indices that are returned by **lsload -N**.

The order string is used for host sorting and selection. The ordering begins with the rightmost index in the order string and proceeds from right to left. The hosts are sorted into order based on each load index, and if more hosts are available than were requested, the LIM drops the least desirable hosts according to that index. The remaining hosts are then sorted by the next index.

After the hosts are sorted by the leftmost index in the order string, the final phase of sorting orders the hosts according to their status, with hosts that are currently not available for load sharing (that is, not in the `ok` state) listed at the end.

Because the hosts are sorted again for each load index, only the host status and the leftmost index in the order string actually affect the order in which hosts are listed. The other indices are only used to drop undesirable hosts from the list.

When sorting is done on each index, the direction in which the hosts are sorted (increasing versus decreasing values) is determined by the default order returned by **lsinfo** for that index. This direction is chosen such that after sorting, by default, the hosts are ordered from best to worst on that index.

When used with a `cu` string, the preferred compute unit order takes precedence. Within each compute unit hosts are ordered according to the order string requirements.

Syntax

```
[!] [-]resource_name [: [-]resource_name] ...
```

You can specify any built-in or external load index or static resource.

The syntax **!** sorts the candidate hosts. It applies to the entire `order []` section. After candidate hosts are selected and sorted initially, they are sorted again before a job is scheduled by all plug-ins. **!** is the first character in the merged `order []` string if you specify it.

! only works with consumable resources because resources can be specified in the `order []` section and their value may be changed in schedule cycle (for example, slot or memory). For the scheduler, slots in `RUN`, `SSUSP`, `USUP` and `RSV` may become free in different scheduling phases. Therefore, the slot value may change in different scheduling cycles.

Using slots to order candidate hosts may not always improve the utilization of whole cluster. The utilization of the cluster depends on many factors.

When an index name is preceded by a minus sign '-', the sorting order is reversed so that hosts are ordered from worst to best on that index.

In the following example, LSF first tries to pack jobs on to hosts with the least slots. Three serial jobs and one parallel job are submitted.

```
HOST_NAME STATUS JL/U MAX NJOBS RUN SSUSP USUSP RSV
hostA ok - 4 0 0 0 0 0
hostB ok - 4 0 0 0 0 0
```

The three serial jobs are submitted:

- `bsub -R "order[-slots]" job1`
- `bsub -R "order[-slots]" job2`
- `bsub -R "order[-slots]" job3`

The parallel job is submitted:

- `bsub -n 4 -R "order[-slots] span[hosts=1]" sleep 1000`

The serial jobs are dispatched to one host (hostA). The parallel job is dispatched to another host.

Change the global LSF default sorting order

You can change the global LSF system default sorting order of resource requirements so the scheduler can find the right candidate host. This makes it easier to maintain a single global default order instead of having to set a default order in the `lsb.queues` file for every queue defined in the system. You can also specify a default order to replace the default sorting value of `r15s:pg`, which could impact performance in large scale clusters.

To set the default order, you can use the `DEFAULT_RESREQ_ORDER` parameter in `lsb.params`. For example, you can pack jobs onto hosts with the fewest free slots by setting `DEFAULT_RESREQ_ORDER=-slots;-maxslots`. This will dispatch jobs to the host with the fewest free slots and secondly to hosts with the smallest number of jobs slots defined (MXJ). This will leave larger blocks of free slots on the hosts with larger MXJ (if the slot utilization in the cluster is not too high).

Commands with the `-R` parameter (such as **bhosts**, **bmod** and **bsub**) will use the default order defined in `DEFAULT_RESREQ_ORDER` for scheduling if no order is specified in the command.

To change the system default sorting order:

1. Configure the `DEFAULT_RESREQ_ORDER` in `lsb.params`.
2. Run **badmin reconfig** to have the changes take effect.
3. Optional: Run **bparams -a | grep ORDER** to verify that the parameter was set. Output similar to that shown in the following example appears:

```
DEFAULT_RESREQ_ORDER = r15m:it
```

4. Submit your job.
5. When you check the output, you can see the sort order for the resource requirements in the **RESOURCE REQUIREMENT DETAILS** section:

```
bjobs -l 422
Job <422>, User <lsfadmin>, Project <default>
Status <DONE>, Queue <normal>, Command <sleep1>
Fri Jan 18 13:29:35: Submitted from hostA, CWD
                    <home/admin/lsf/conf/lsbatch/LSF/configdir>;
Fri Jan 18 13:29:37: Started on <hostA>, Execution Home </home/lsfadmin>,
Execution CWD </home/admin/lsf/conf/lsbatch/LSF/configdir>;
Fri Jan 18 13:29:44: Done successfully. The CPU time used is 0.0 seconds.

MEMORY USAGE:
MAX MEM: 3 Mbytes;  AVG MEM: 3 Mbytes
```

SCHEDULING PARAMETERS:											
	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

RESOURCE REQUIREMENT DETAILS:
Combined: `select[type == local] order[r15m:it]`
Effective: `select[type == local] order[r15m:it]`

When changing the value for DEFAULT_RESREQ_ORDER, note the following:

- For job scheduling, there are three levels at which you can sort resources from the order section: job-level, application-level and queue-level. The sort order for resource requirements defined at the job level overwrites those defined at the application level or queue level. The sort order for resource requirements defined at the application level overwrites those defined at the queue level. If no sort order is defined at any level, **mbschd** uses the value of DEFAULT_RESREQ_ORDER when scheduling the job.
- You should only sort by one or two resources since it may take longer to sort with more.
- Once the job is running, you cannot redefine the sort order. However, you can still change it while the job is in PEND state.
- For MultiCluster forward and MultiCluster lease modes, the DEFAULT_RESREQ_ORDER value for each local cluster is used.
- If you change DEFAULT_RESREQ_ORDER then requeue a running job, the job will use the new DEFAULT_RESREQ_ORDER value for scheduling.

Specify multiple -R options

bsub accepts multiple -R options for the order section.

Restriction:

Compound resource requirements do not support multiple -R options.

You can specify multiple resource requirement strings instead of using the && operator. For example:

```
bsub -R "order[r15m]" -R "order[ut]"
```

LSF merges the multiple -R options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts. The number of -R option sections is unlimited.

Default

The default sorting order is r15s:pg: **ls:r1m**).

```
swp:r1m:tmp:r15s
```

Resizable jobs

The order in which hosts are considered for resize allocation requests is determined by the order expression of the job. For example, to run an autosizable job on 1-100 slots, preferring hosts with larger memory, the following job submission specifies this resource request:

```
bsub -ar -app <applicaion_file> -n "1,100" -R "rusage[swp=100,license=1]" myjob
```

When slots on multiple hosts become available simultaneously, hosts with larger available memory get preference when the job adds slots.

Note:

Resizable jobs cannot have compound or alternative resource requirements.

Reordering hosts

You can reorder hosts using the `order[!]` syntax.

Suppose host `h1` exists in a cluster and has 110 units of a consumable resource 'res' while host `h2` has 20 of this resource ('res' can be the new batch built-in resource slots, for example). Assume that these two jobs are pending and being considered by scheduler in same scheduling cycle, and job1 will be scheduled first:

```
Job1:bsub -R "maxmem>1000" -R "order[res] rusage[res=100]" -q q1 sleep 10000
```

```
Job2:bsub -R "mem<1000" -R "order[res] rusage[res=10]" -q q2 sleep 10000
```

Early in the scheduling cycle, a candidate host list is built by taking either all hosts in the cluster or the hosts listed in any asked host list (-m) and ordering them by the order section of the resource requirement string. Assume the ordered candidate host lists for the jobs look like this after the ordering:

```
Job1:{h1, h7, h4, h10}
```

```
Job2:{h1,  
h2}
```

This means `h1` ends up being the highest 'res' host the candidate host lists of both jobs. In later scheduling only, one by one each job will be allocated hosts to run on and resources from these hosts.

Suppose Job1 is scheduled to land on host `h1`, and thus will be allocated 100 'res'. Then when Job2 is considered, it too might be scheduled to land on host `h1` because its candidate host list still looks the same. That is, it does not take into account the 100 'res' allocated to Job1 within this same scheduling cycle. To resolve this problem, use `!` at the beginning of the order section to force the scheduler to re-order candidate host lists for jobs in the later scheduling phase:

```
Job1:bsub -R "maxmem>1000" -R "order[!res] rusage[res=100]" -q q1 sleep 10000
```

```
Job2:bsub -R "mem <1000" -R "order[!res] rusage[res=10]" -q q2 sleep 10000
```

The `!` forces a reordering of Job2's candidate host list to Job2: {h2, h1} since after Job1 is allocated 100 'res' on `h1`, `h1` will have 10 'res' (110-100) whereas `h2` will have 20.

You can combine new batch built-in resources slots/maxslots with both reverse ordering and re-ordering to better ensure that large parallel jobs will have a chance to run later (improved packing). For example:

```
bsub -n 2 -R "order[!-slots:maxslots]" ...
```

```
bsub -n 1 -R "order[!-slots:maxslots]" ...
```

Usage string

This string defines the expected resource usage of the job. It is used to specify resource reservations for jobs, or for mapping jobs on to hosts and adjusting the load when running interactive jobs.

By default, no resources are reserved.

LSF rejects resource requirement strings where an `rusage` section contains a non-consumable resource.

Multi-phase resources

Multiple phases within the `rusage` string allow different time periods to have different memory requirements (load index `mem`). The duration of all except the last phase must be specified, while decay rates are all optional and are assumed to be 0 if omitted. If the optional final duration is left blank, the final resource requirement applies until the job is finished.

Multi-phase resource reservations cannot include increasing resources, but can specify constant or decreasing resource reservations over multiple periods of time.

Resource reservation limits

Resource requirement reservation limits can be set using the parameter `RESRSV_LIMIT` in `lsb.queues`. Queue-level `RES_REQ` usage values (set in `lsb.queues`) must be in the range set by `RESRSV_LIMIT`, or the queue-level `RES_REQ` is ignored. Merged `RES_REQ` usage values from the job and application levels must be in the range of `RESRSV_LIMIT`, or the job is rejected.

When both the `RES_REQ` and `RESRSV_LIMIT` are set in `lsb.queues` for a consumable resource, the queue-level `RES_REQ` no longer acts as a hard limit for the merged `RES_REQ` usage values from the job and application levels. In this case only the limits set by `RESRSV_LIMIT` must be satisfied, and the queue-level `RES_REQ` acts as a default value.

Batch jobs

The resource usage (`rusage`) section can be specified at the job level, with the queue configuration parameter `RES_REQ`, or with the application profile parameter `RES_REQ`.

Basic syntax

```
rusage[usage_string [, usage_string] [|| usage_string] ...]
```

where `usage_string` is:

```
load_index=value [:load_index=value]... [:duration=minutes[m]  
| :duration=hoursh | :duration=secondss [:decay=0 | :decay=1]]
```

Note: The default unit for duration is "minutes". To use hours or seconds as the unit, append "h" or "s" to the duration value. For example, `duration=30` means 30 minutes, as does `duration=30m` explicitly. Accordingly, `duration=30h` means 30 hours, and `duration=30s` means 30 seconds.

The keyword `threshold` in the `rusage` section lets you specify a threshold at which the consumed resource must be before an allocation should be made. If the threshold is not satisfied for every host in the cluster, the job becomes pending.

To specify a threshold in the command line, use **`bsub -R`** to attach a threshold to a resource in the `rusage` section. For example:

```
bsub -R "rusage[bwidth=1:threshold=5]" sleep 100
```

You can use **`bmod -R`** to change the content of the `rusage` section. For example:

```
bmod -R "rusage[bwidth=1:threshold=7]" <job ID>
```

To specify a threshold in the configuration file, Use `RES_REQ` to attach a threshold to a resource in `lsb.queues`. For example:

```
RES_REQ = rusage[bwidth=1:threshold=5]
```

You can use `RES_REQ` to attach a threshold to a resource in `lsb.applications`. For example:

```
RES_REQ = rusage[bwidth=1:threshold=5]
```

Multi-phase memory syntax

```
rusage[multi_usage_string [, usage_string] ...]
```

where `multi_usage_string` is:

```
mem=(v1 [v2 ... vn]):[duration=(t1 [t2 ... tm]):[:decay=(d1 [d2... dk])]
```

for $m = n | n-1$. For a single phase ($n=1$), duration is not required.

if $k > m$, d_{m+1} to d_k will be ignored; if $k < m$, $d_{k+1} = .. = d_m = 0$.

`usage_string` is the same as the basic syntax, for any `load_index` other than `mem`.

Multi-phase syntax can be used with a single phase memory resource requirement as well as for multiple phases.

For multi-phase slot-based resource reservation, use with `RESOURCE_RESERVE_PER_TASK=Y` in `lsb.params`.

Multi-phase resource reservations cannot increase over time. A job submission with increasing resource reservations from one phase to the next will be rejected. For example:

```
bsub -R"rusage[mem=(200 300):duration=(2 3)]" myjob
```

specifies an increasing memory reservation from 200 MB to 300 MB. This job will be rejected.

Tip: When a multi-phase mem resource requirement is being used, duration can be specified separately for single-phase resources.

Load index

Internal and external load indices are considered in the resource usage string. The resource value represents the initial reserved amount of the resource.

Duration

The duration is the time period within which the specified resources should be reserved. Specify a duration equal to or greater than the ELIM updating interval.

- If the value is followed by the letter s, m, or h, the specified time is measured in seconds, minutes, or hours respectively.
- By default, duration is specified in minutes.
For example, the following specify a duration of 1 hour for multi-phase syntax:

- `duration=(60)`
- `duration=(1h)`
- `duration=(3600s)`

For example, the following specify a duration of 1 hour for single-phase syntax:

- `duration=60`
- `duration=1h`
- `duration=3600s`

Tip: Duration is not supported for static shared resources. If the shared resource is defined in an `lsb.resources` Limit section, then duration is not applied.

Decay

The decay value indicates how the reserved amount should decrease over the duration.

- A value of 1 indicates that system should linearly decrease the amount reserved over the duration.
- A value of 0 causes the total amount to be reserved for the entire duration.

Values other than 0 or 1 are unsupported, and are taken as the default value of 0. If duration is not specified, decay value is ignored.

Tip: Decay is not supported for static shared resources. If the shared resource is defined in an `lsb.resources` Limit section, then decay is not applied.

Default

If a resource or its value is not specified, the default is not to reserve that resource. If duration is not specified, the default is to reserve the total amount for the lifetime of the job. (The default decay value is 0.)

Example

```
rusage[mem=50:duration=100:decay=1]
```

This example indicates that 50 MB memory should be reserved for the job. As the job runs, the amount reserved will decrease at approximately 0.5 MB per minute until the 100 minutes is up.

Resource reservation method

Specify the resource reservation method in the resource usage string by using the `/job`, `/host`, or `/task` keyword after the numeric value. The resource reservation method specified in the resource string overrides the global setting that is specified in the `ReservationUsage` section of the `lsb.resources` file. You can only specify resource reservation methods for consumable resources. Specify the resource reservation methods as follows:

- `value/task`

Specifies per-task reservation of the specified resource. This is the equivalent of specifying `PER_TASK` for the `METHOD` parameter in the `ReservationUsage` section of the `lsb.resources` file.

- `value/job`
Specifies per-job reservation of the specified resource. This is the equivalent of specifying `PER_JOB` for the `METHOD` parameter in the `ReservationUsage` section of the `lsb.resources` file.
- `value/host`
Specifies per-host reservation of the specified resource. This is the equivalent of specifying `PER_HOST` for the `METHOD` parameter in the `ReservationUsage` section of the `lsb.resources` file.

- Basic syntax:

```
resource_name=value/method:duration=value:decay=value
```

For example,

```
rusage[mem=10/host:duration=10:decay=0]
```

- Multi-phase memory syntax:

```
resource_name=(value ...)/method:duration=(value ...):decay=value
```

For example,

```
rusage[mem=(50 20)/task:duration=(10 5):decay=0]
```

How simple queue-level and job-level rusage sections are resolved

Job-level rusage overrides the queue level specification:

- For internal load indices (`r15s`, `r1m`, `r15m`, `ut`, `pg`, `io`, `ls`, `it`, `tmp`, `swp`, and `mem`), the job-level value cannot be larger than the queue-level value (unless the limit parameter `RESRSV_LIMIT` is being used as a maximum instead of the queue-level value).
- For external load indices, the job-level rusage can be larger than the queue-level requirements.
- For duration, the job-level value of internal and external load indices cannot be larger than the queue-level value.
- For multi-phase simple rusage sections:
 - For internal load indices (`r15s`, `r1m`, `r15m`, `ut`, `pg`, `io`, `ls`, `it`, `tmp`, `swp`, and `mem`), the first phase of the job-level value cannot be larger than the first phase of the queue-level value (unless the limit parameter `RESRSV_LIMIT` is being used as a maximum instead of the queue-level value).
 - For duration and decay, if either job-level or queue-level is multi-phase, the job-level value will take precedence.

How simple queue-level and job-level rusage sections are merged

When both job-level and queue-level rusage sections are defined, the rusage section defined for the job overrides the rusage section defined in the queue. The two rusage definitions are merged, with the job-level rusage taking precedence. For example:

Example 1

Given a `RES_REQ` definition in a queue:

```
RES_REQ = rusage[mem=200:lic=1] ...
```

and job submission:

```
bsub -R "rusage[mem=100]" ...
```

The resulting requirement for the job is

```
rusage[mem=100:lic=1]
```

where `mem=100` specified by the job overrides `mem=200` specified by the queue. However, `lic=1` from queue is kept, since job does not specify it.

Example 2

For the following queue-level RES_REQ (decay and duration defined):

```
RES_REQ = rusage[mem=200:duration=20:decay=1] ...
```

and job submission (no decay or duration):

```
bsub -R "rusage[mem=100]" ...
```

The resulting requirement for the job is:

```
rusage[mem=100:duration=20:decay=1]
```

Queue-level duration and decay are merged with the job-level specification, and `mem=100` for the job overrides `mem=200` specified by the queue. However, `duration=20` and `decay=1` from queue are kept, since job does not specify them.

rusage in application profiles

See [Resource requirements](#) for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

How simple queue-level rusage sections are merged with compound rusage sections

When simple queue-level and compound application-level or job-level rusage sections are defined, the two rusage definitions are merged. If a job-level resource requirement (simple or compound) is defined, the application level is ignored and the job-level and queue-level sections merge. If no job-level resource requirement is defined, the application-level and queue-level merge.

When a compound resource requirement merges with a simple resource requirement from the queue-level, the behavior depends on whether the queue-level requirements are job-based or not.

Example 1

Job-based simple queue-level requirements apply to the first term of the merged compound requirements. For example:

Given a RES_REQ definition for a queue which refers to a job-based resource:

```
RES_REQ = rusage[lic=1] ...
```

and job submission resource requirement:

```
bsub -R "2*{rusage[mem=100] ...} + 4*{[mem=200:duration=20:decay=1] ...}"
```

The resulting requirement for the job is

```
bsub -R "2*{rusage[mem=100:lic=1] ...} + 4*{rusage[mem=200:duration=20:decay=1] ...}"
```

The job-based resource `lic=1` from queue is added to the first term only, since it is job-based and wasn't included the job-level requirement.

Example 2

Host-based or slot-based simple queue-level requirements apply to all terms of the merged compound requirements. For example:

For the following queue-level RES_REQ which does not include job-based resources:

```
RES_REQ = rusage[mem=200:duration=20:decay=1] ...
```

and job submission:

```
bsub -R "2*{rusage[mem=100] ...} + 4*{rusage[lic=1] ...}"
```

The resulting requirement for the job is:

```
2*{rusage[mem=100:duration=20:decay=1] ...} + 4*{rusage[lic=1:mem=200:duration=20:decay=1] ...}
```

Where `duration=20` and `decay=1` from queue are kept, since job does not specify them in any term. In the first term `mem=100` from the job is kept; in the second term `mem=200` from the queue is used since it wasn't specified by the job resource

requirement.

Specify multiple -R options

bsub accepts multiple -R options for the rusage section.

Restriction: Compound resource requirements do not support multiple -R options. Multi-phase rusage strings do not support multiple -R options.

You can specify multiple resource requirement strings instead of using the && operator. For example:

```
bsub -R "rusage[mem=100]" -R "rusage[tmp=50:duration=60]"
```

LSF merges the multiple -R options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

The number of -R option sections is unlimited.

Comma-separated multiple resource requirements within one rusage string is supported. For example:

```
bsub -R "rusage[mem=20]" -R "rusage[mem=10||mem=10]" myjob
```

A given load index cannot appear more than once in the resource usage string.

Specify alternative usage strings

If you use more than one version of an application, you can specify the version you prefer to use together with a legacy version you can use if the preferred version is not available. Use the **OR (| |)** expression to separate the different usage strings that define your alternative resources.

Job-level resource requirement specifications that use the **| |** operator are merged with other rusage requirements defined at the application and queue levels.

Note: Alternative rusage strings cannot be submitted with compound resource requirements.

How LSF merges rusage strings that contain the || operator

The following examples show how LSF merges job-level and queue-level rusage strings that contain the **| |** operator.

Queue level RES_REQ=rusage...	Job level bsub -R "rusage ..."	Resulting rusage string
[mem=200:duration=180]	[w1=1 w2=1 w3=1]"	[w1=1, mem=200:duration=180 w2=1, mem=200:duration=180 w3=1, mem=200:duration=180]
[w1=1 w2=1 w3=1]	[mem=200:duration=180]"	[mem=200:duration=180, w1=1 mem=200:duration=180, w2=1 mem=200:duration=180, w3=1]

Note:

Alternative rusage strings cannot be submitted with compound resource requirements.

Non-batch environments

Resource reservation is only available for batch jobs. If you run jobs using only LSF Base, such as through **lsrun**, LIM uses resource usage to determine the placement of jobs. Resource usage requests are used to temporarily increase the load so that a host is not overloaded. When LIM makes a placement advice, external load indices are not considered in the resource usage string. In this case, the syntax of the resource usage string is

```
res [=value] : res [=value] : ... : res [=value]
```

res is one of the resources whose value is returned by the **lsload** command.

```
rusage[r1m=0.5:mem=20:swp=40]
```

The preceding example indicates that the task is expected to increase the 1-minute run queue length by 0.5, consume 20 MB of memory and 40 MB of swap space.

If no value is specified, the task is assumed to be intensive in using that resource. In this case no more than one task will be assigned to a host regardless of how many CPUs it has.

The default resource usage for a task is `r15s=1.0:r1m=1.0:r15m=1.0`. This indicates a CPU-intensive task which consumes few other resources.

Resizable jobs

Unlike the other components of a resource requirement string that only pertain to adding additional slots to a running job, `rusage` resource requirement strings affect the resource usage when slots are removed from the job as well.

When adding or removing slots from a running job:

- The amount of *slot-based* resources added to or removed from the job allocation is proportional to the change in the number of slots
- The amount of *job-based* resources is not affected by a change in the number of slots
- The amount of each *host-based* resource is proportional to the change in the number of hosts

When using multi-phase resource reservation, the job allocation is based on the phase of the resource reservation.

Note: Resizable jobs cannot have compound resource requirements.

Duration and decay of `rusage`

Duration and decay of resource usage and the `||` operator affect resource allocation.

Duration or decay of a resource in the `rusage` expression is ignored when scheduling the job for the additional slots.

If a job has the following `rusage` string: `rusage[mem=100:duration=300]`, the resize request of one additional slot is scheduled on a host only if there are 100 units of memory available on that host. In this case, `mem` is a slot-based resource (`RESOURCE_RESERVE_PER_TASK=Y` in `lsb.params`).

Once the resize operation is done, if the job has been running less than 300 seconds then additional memory will be reserved only until the job has run for 300 seconds. If the job has been running for more than 300 seconds when the job is resized, no additional memory is reserved. The behavior is similar for decay.

The `||` operator lets you specify multiple alternative `rusage` strings, one of which is used when dispatching the job. You cannot use **`bmod`** to change `rusage` to a new one with a `||` operator after the job has been dispatched

For job resize, when the `||` operator is used, the resize request uses the `rusage` expression that was originally used to dispatch the job. If the `rusage` expression has been modified since the job started, the resize request is scheduled using the new single `rusage` expression.

Example 1

You want to run an autoresizable job such that every slot occupied by the job reserves 100 MB of swap space. In this case, `swp` is a slot-based resource (`RESOURCE_RESERVE_PER_TASK=Y` in `lsb.params`). Each additional slot that is allocated to the job should reserve additional swap space. The following job submission specifies this resource request:

```
bsub -ar -app <applicaion_file> -n "1,100" -R "rusage[swp=100]" myjob
```

Similarly, if you want to release some of the slots from a running job, resources that are reserved by the job are decreased appropriately. For example, for the following job submission:

```
bsub -ar -app <applicaion_file> -n 100 -R "rusage[swp=50]" myjob  
Job <123> is submitted to default queue.
```

you can run **`bresize release`** to release all the slots from the job on one host:

```
bresize release "hostA" 123
```

The swap space used by the job is reduced by the number of slots used on `hostA` times 50 MB.

Example 2

You have a choice between two versions of an application, each version having different memory and swap space requirements on hosts. If you submit an autoresizable job with the `||` operator, once the job is started using one version of an application, slots added to a job during a resize operation reserve resources depending on which version of the application was originally run. For example, for the following job submission:

```
bsub -n "1,100" -ar -R "rusage[mem=20:app_lic_v201=1 || mem=20:swp=50:app_lic_v15=1]" myjob
```

If the job starts with `app_lic_v15`, each additional slot added in a resize operation reserves 20 MB of memory and 50 MB of swap space.

Span string

A span string specifies the locality of a parallel job. If span is omitted, LSF allocates the required processors for the job from the available set of processors.

Syntax

The span string supports the following syntax:

`span[hosts=1]`

Indicates that all the processors allocated to this job must be on the same host.

`span[block=value]`

For parallel jobs, LSF will allocate slots to the job based on block size. LSF tries to pack as many blocks on one host as possible, then goes to next one. Each host is only checked once.

`span[ptile=value]`

Indicates the number of processors on each host that should be allocated to the job, where *value* is one of the following:

- Default ptile value, specified by *n* processors. In the following example, the job requests 4 processors on each available host, regardless of how many processors the host has:

```
span[ptile=4]
```

- Predefined ptile value, specified by `'!'`. The following example uses the predefined maximum job slot limit `lsb.hosts` (MXJ per host type/model) as its value:

```
span[ptile='!']
```

Tip: If the host type/model does not define MXJ, the `span[ptile='!']` value is ignored.

Restriction: Under bash 3.0, the exclamation mark (!) is not interpreted correctly by the shell. To use predefined ptile value (`ptile='!'`), use the `+H` option to disable `'!'` style history substitution in bash (`sh +H`).

- Predefined ptile value with optional multiple ptile values, per host type or host model:
 - For host type, you must specify `same[type]` in the resource requirement. In the following example, the job requests 8 processors on a host of type `HP`, and 2 processors on a host of type `LINUX`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host types:

```
span[ptile='!',HP:8,LINUX:2] same[type]
```

- For host model, you must specify `same[model]` in the resource requirement. In the following example, the job requests 4 processors on hosts of model `PC1133`, and 2 processors on hosts of model `PC233`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host models:

```
span[ptile='!',PC1133:4,PC233:2] same[model]
```

`span[stripe]`

For parallel jobs, LSF stripes the tasks of the job across the free resources of the candidate hosts.

For example, if you submit a job that requests four tasks with the following command:

```
bsub -n 4 -R "span[stripe]" ./a.out
```

The task placement depends on the free resources:

- If there is one candidate host, that host has four tasks (4).
- If there are two candidate hosts, each host has two tasks (2,2).
- If there are three candidate hosts, one host has two tasks, and the two other hosts have one task each (2,1,1).
- If there are four candidate hosts, each host has one task (1,1,1,1).

`span[stripe=max_tasks]`

For parallel jobs, LSF stripes the tasks of the job across the free resources of the candidate hosts up to the specified maximum number of tasks on each host.

`span[hosts=-1]`

Disables span setting in the queue. LSF allocates the required processors for the job from the available set of processors.

Examples

The following examples are for jobs that are submitted in a cluster with the following idle hosts:

bhosts								
HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
host1	ok	-	32	0	0	0	0	0
host2	ok	-	32	0	0	0	0	0
host3	ok	-	32	0	0	0	0	0
host4	ok	-	32	0	0	0	0	0
host5	ok	-	32	0	0	0	0	0

- Submit a job with striping:

```
bsub -n 32 -R "span[stripe]" myjob
```

View the task distribution of a job with striping:

bjobs							
JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
118	userA	RUN	normal	hostA	7*host1	myjob	Nov 29 14:27
					7*host2		
					6*host3		
					6*host4		
					6*host5		

Tasks are distributed evenly to the hosts, and any additional tasks are distributed to the first hosts. There are 7 tasks on host1 and host2, and 6 tasks on host3, host4, and host5.

- Submit a job with ptille:

```
bsub -n 32 -R "span[ptile=8]" myjob
```

View the task distribution of a job with ptille:

bjobs							
JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
119	userA	RUN	normal	hostA	8*host1	myjob	Nov 29 14:29
					8*host2		
					8*host3		
					8*host4		

Tasks are distributed with exactly 8 tasks per host, except the last host might have less tasks than the ptille value if there are fewer total tasks to distribute.

- Submit a job based on block size:

```
bsub -n 32 -R "span[block=8]" myjob
```

View the task distribution of a job based on block size:

bjobs							
JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
120	userA	RUN	normal	hostA	32*host1	myjob	Nov 29 14:32

Tasks are packed in blocks of 8.

- Submit a job to a single host:

```
bsub -n 32 -R "span[hosts=1]" myjob
```

View the task distribution of a job that is submitted to a single host:

```
bjobs
JOBID    USER    STAT    QUEUE    FROM_HOST    EXEC_HOST    JOB_NAME    SUBMIT_TIME
121      userA    RUN     normal    hostA        32*host1     myjob       Nov 29 14:34
```

Tasks are placed into a single host (host1).

Resizable jobs

For resource requirements with `span[hosts=1]`, a resize request is limited to slots on the first-execution host of the job. This behavior eliminates the ambiguities that arise when the span expression is modified from the time that the job was originally dispatched.

For `span[ptile=n]`, the job will be allocated exactly n slots on some number of hosts, and a number between 1 and n slots (inclusive) on one host. This is true even if a range of slots is requested. For example, for the following job submission:

```
bsub -n "1,20" -R "span[ptile=2]" sleep 10000
```

This special span behavior does not only apply to resize requests. It applies to resizable jobs only when the original allocation is made, and in making additional resize allocations.

If every host has only a single slot available, the job is allocated one slot.

Resize requests with partially filled hosts are handled so that LSF does not choose any slots on hosts already occupied by the job. For example, it is common to use the `ptile` feature with `span[ptile=1]` to schedule exclusive jobs.

For a resizable job (auto-resizable or otherwise) with a range of slots requested and `span[ptile=n]`, whenever the job is allocated slots, it will receive either of the following:

- The maximum number of slots requested, comprising n slots on each of a number of hosts, and between 0 and $n-1$ (inclusive) slots on one host
- n slots on each of a number of hosts, summing to some value less than the maximum

For example, if a job requests between 1 and 14 additional slots, and `span[ptile=4]` is part of the job resource requirement string, when additional slots are allocated to the job, the job receives either of the following:

- 14 slots, with 2 slots on one host and 4 slots on each of 3 hosts
- 4, 8 or 12 slots, such that 4 slots are allocated per host of the allocation

Note: Resizable jobs cannot have compound resource requirements.

Example

When running a parallel exclusive job, it is often desirable to specify `span[ptile=1]` so that the job is allocated at most one slot on each host. For an autoresizable job, new slots are allocated on hosts not already used by the job. The following job submission specifies this resource request:

```
bsub -x -ar -app <applicaion_file> -n "1,100" -R "span[ptile=1]" myjob
```

When additional slots are allocated to a running job, the slots will be on new hosts, not already occupied by the job.

Block scheduling

For applications that are not especially sensitive to network latency, or where you prefer to get throughput, you can allocate slots for a parallel job with a specific block size. The applications specified by the job may be running as threaded processes on groups of n cores, but using MPI applications or other socket connections between blocks. LSF will allocate slots to the job based on block size. LSF tries to pack as many blocks on one host as possible, then goes to next one. Each host is only checked once. It does not matter which host contains the slot blocks. The job can start as soon as any previous job is complete.

In the illustration below, for example, each color represents a different job. There are four 16 way jobs:

Node 1	Node 2	Node 3	Node4
4 cores	4 cores	4 cores	4 cores
4 cores	4 cores	4 cores	4 cores
4 cores	4 cores	4 cores	4 cores
4 cores	4 cores	4 cores	4 cores

For **bsub -n 16** and **block=4**, only 4 x 4 slot blocks are necessary. It does not matter which host contains the slot blocks. The job can start as soon as any previous job is complete.

This packing policy is supported by the keyword **block** ("**span[block=value]**") in the **span** section of the resource requirement string. "**span[block=value]**" can also be configured in the **RES_REQ** parameter in **lsb.queues** and **lsb.applications**.

When a block size is specified for a job, LSF allocates only a multiple of the block size for the job. For example, for jobs with block size = 4:

- **bsub -n 2,13**: 4, 8 or 12 slots are allocated to the job (in blocks of size 4).
- **bsub -n 5**: The job is rejected.
- **bsub -n 9,10**: The job is rejected.
- **bsub -n 2,3**: The job is rejected.
- **bsub -n 12**: The job is accept, and allocates 3 blocks of size 4.
- **bsub -n 2**: The job is rejected.
- **bsub -n 3**: The job is rejected.

The minimum value in **-n min,max** is silently changed to a multiple of the block. For example:

bsub -n 2,8 -R span[block=4] sleep 1d

is changed to:

bsub -n 4,8 -R span[block=4] sleep 1d

LSF tries to pack as many blocks in to one host as possible, then goes to the next host. For example, assume **host1** has 8 slots, and **host2** has 8 slots, and **host3** also has 8 slots, where 2 slots of each host are consumed by other jobs. For a job with **-n 9 "span[block=3]"**, the allocation will be:

- **host1**: 6 slots
- **host2**: 3 slots

The following is an example of how you can display hosts with their static and dynamic resource information, specify a block size and resource requirements for a job, and see the output:

bhosts

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
hostA	ok	-	8	0	0	0	0	0
hostB	ok	-	8	0	0	0	0	0
hostC	ok	-	8	0	0	0	0	0
hostD	unavail	-	1	0	0	0	0	0
hostE	ok	-	4	0	0	0	0	0
hostF	ok	-	4	0	0	0	0	0

```
bsub -n 24 -R "order[slots] span[block=4]" sleep 1d
Job <418> is submitted to default queue <normal>.
```

bjobs

JOBID	USER	STAT	QUEUE	FROM HOST	EXEC_HOST	JOB_NAME	SUBMIT TIME
418	user1	RUN	normal	hostE	8*hostC	sleep 1d	Sep 4 21:36
					8*hostB	sleep 1d	Sep 4 21:36
					8*hostA	sleep 1d	Sep 4 21:36

bhosts

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
-----------	--------	------	-----	-------	-----	-------	-------	-----

hostA	closed	-	8	8	8	0	0	0
hostB	closed	-	8	8	8	0	0	0
hostC	closed	-	8	8	8	0	0	0
hostD	unavail	-	1	0	0	0	0	0
hostE	ok	-	4	0	0	0	0	0
hostF	ok	-	4	0	0	0	0	0

The following are some additional examples of how you can use `"span[block=value]"` when submitting a job with resource requirements:

- To specify a predefined block value, per host type or host model, using !:
`bsub -n "2,10" -R "span[block='!'] same[type]" myjob`
- To specify a predefined block value with optional multiple block values, per host type or host model:
`bsub -n "2,10" -R "span[block='!',HP:8,SGI:8,LINUX:2] same[type]" myjob`

If the host type/model does not define MXJ, the default predefined block value is 1.

`"span[block=value]"` can be displayed by **bjobs -l**, **bhist -l**, **bqueues -l**, **bapp -l** and **bacct -l**.

When using the block scheduling feature, note the following:

- For Queue Host Limit (HOSTLIMIT_PER_JOB), **mbatchd** will not reject a job with `block=x` because the exact number of allocated hosts can only be obtained during scheduling.
- `"span[block=value]"` and `"span[ptile=value]"` cannot be specified at the same time. `"span[block=value]"` and `"span[host=value]"` also cannot be specified at the same time because span cannot accept more than one criteria and multiple -R does not support multiple span definitions.
- For the LSF multicluster capability, when using the job forwarding model, job with `block=x` cannot be forwarded to a remote cluster which has a version prior to 9.1.2. When using the leasing model, job with `block=x` cannot be allocated to hosts leased from a remote cluster with a version prior to 9.1.2.

Same string

Tip:

You must have the parallel batch job scheduler plugin installed in order to use the same string.

Parallel jobs run on multiple hosts. If your cluster has heterogeneous hosts, some processes from a parallel job may for example, run on Solaris. However, for performance reasons you may want all processes of a job to run on the same type of host instead of having some processes run on one type of host and others on another type of host.

The *same* string specifies that all processes of a parallel job must run on hosts with the same resource.

You can specify the *same* string:

- At the job level in the resource requirement string of:
 - **bsub**
 - **bmod**
- At the queue level in `lsb.queues` in the `RES_REQ` parameter.

When queue-level, application-level, and job-level *same* sections are defined, LSF combines requirements to allocate processors.

Syntax

```
resource_name[:resource_name]...
```

You can specify any static resource.

For example, if you specify `resource1:resource2`, if hosts always have both resources, the string is interpreted as allocate processors only on hosts that have the same value for `resource1` and the same value for `resource2`.

If hosts do not always have both resources, it is interpreted as allocate processors either on hosts that have the same value for `resource1`, or on hosts that have the same value for `resource2`, or on hosts that have the same value for both `resource1` and `resource2`.

Specify multiple -R options

bsub accepts multiple -R options for the same section.

Restriction: Compound resource requirements do not support multiple -R options.

You can specify multiple resource requirement strings instead of using the && operator. For example:

```
bsub -R "same[type]" -R "same[model]"
```

LSF merges the multiple -R options into one string and dispatches the job if all of the resource requirements can be met. By allowing multiple resource requirement strings and automatically merging them into one string, LSF simplifies the use of multiple layers of wrapper scripts.

Resizable jobs

The same expression ensures that the resize allocation request is dispatched to hosts that have the same resources as the first-execution host. For example, if the first execution host of a job is SOL7 and the resource requirement string contains `same[type]`, additional slots are allocated to the job on hosts of type SOL7.

Taking the same resource as the first-execution host avoids ambiguities that arise when the original job does not have a same expression defined, or has a different same expression when the resize request is scheduled.

For example, a parallel job may be required to have all slots on hosts of the same type or model for performance reasons. For an autoresizable job, any additional slots given to the job will be on hosts of the same type, model, or resource as those slots originally allocated to the job. The following command submits an autoresizable job such that all slots allocated in a resize operation are allocation on hosts with the same model as the original job:

```
bsub -ar -app <application_file> -n "1,100" -R "same[model]" myjob
```

Examples

```
bsub -n 4 -R"select[type==any] same[type:model]" myjob
```

Run all parallel processes on the same host type. Allocate 6 processors on the any host type or model as long as all the processors are on the same host type and model.

```
bsub -n 6 -R"select[type==any] same[type:model]" myjob
```

Run all parallel processes on the same host type and model. Allocate 6 processors on any host type or model as long as all the processors are on the same host type and model.

Same string in application profiles

See [Resource requirements](#) for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

Compute unit string

A cu string specifies the network architecture-based requirements of parallel jobs. cu sections are accepted by **bsub -R**, and by **bmod -R** for non-running jobs.

Compute unit strings are supported in compound and alternative resource requirements except for the `excl` and `balance` keywords.

The cu string supports the following syntax:

cu[type=*cu_type*]

Indicates the type of compute units the job can run on. Types are defined by COMPUTE_UNIT_TYPES in lsb.params. If the type keyword is not specified, the default set by COMPUTE_UNIT_TYPES is assumed.

cu[pref=bestfit | maxavail | minavail | config]

Indicates the compute unit scheduling preference, grouping hosts by compute unit before applying a first-fit algorithm to the sorted hosts. For resource reservation, the default pref=config is always used.

Compute units are ordered as follows:

- **bestfit** attempts to place the job in as few compute units as possible while preferring to use compute units with fewer free slots to reduce fragmentation in the cluster. This scheduling preference does not work with the cu[balance] keyword.
- **config** lists compute units in the order they appear in the ComputeUnit section of lsb.hosts. If pref is not specified, pref=config is assumed.
- **maxavail** lists compute units with more free slots first. Should compute units have equal numbers of free slots, they appear in the order listed in the ComputeUnit section of lsb.hosts.
- **minavail** lists compute units with fewer free slots first. Should compute units have equal numbers of free slots, they appear in the order listed in the ComputeUnit section of lsb.hosts.

Free slots include all available slots that are not occupied by running jobs.

When pref is used with the keyword balance, balance takes precedence.

Hosts accept jobs separated by the time interval set by JOB_ACCEPT_INTERVAL in lsb.params; jobs submitted closer together than this interval will run on different hosts regardless of the pref setting.

cu[maxcus=*number*]

Indicates the maximum number of compute units a job can run over. Jobs may be placed over fewer compute units if possible.

When used with bsub -n *min, max* a job is allocated the first combination satisfying both *min* and maxcus, while without maxcus a job is allocated as close to *max* as possible.

cu[usablecuslots=*number*]

Specifies the minimum number of slots a job must use on each compute unit it occupies. *number* is a non-negative integer value.

When more than one compute unit is used by a job, the final compute unit allocated can provide less than *number* slots if less are needed.

usablecuslots and balance cannot be used together.

cu[balance]

Indicates that a job should be split evenly between compute units, with a difference in compute unit slot allocation of at most 1. A balanced allocation spans the fewest compute units possible.

When used with bsub -n *min, max* the value of *max* is disregarded.

balance and usablecuslots cannot be used together.

When balance and pref are both used, balance takes precedence. The keyword pref is only considered if there are multiple balanced allocations spanning the same number of compute units. In this case pref is considered when choosing the allocation. balance cannot be used with the pref=bestfit scheduling preference.

When balance is used with span[ptile=X] (for X>1) a balanced allocation is one split evenly between compute units, with a difference in compute unit host allocation of at most 1.

balance cannot be used in compound and alternative resource requirements.

cu[excl]

Indicates that jobs must use compute units exclusively. Exclusivity applies to the compute unit granularity that is specified by type.

Compute unit exclusivity must be enabled by `EXCLUSIVE=CU[cu_type]` in `lsb.queues`.

`excl` cannot be used in compound and alternative resource requirements.

Resizable jobs

Autoresizable jobs can be submitted with compute unit resource requirements. The `maxcus` keyword is enforced across the job's entire allocation as it grows, while the `balance` and `usablecuslots` keywords only apply to the initial resource allocation.

Examples

- `bsub -n 11,60 -R "cu[maxcus=2:type=enclosure]" myjob`
Spans the fewest possible compute units for a total allocation of at least 11 slots using at most 2 compute units of type enclosure. In contrast, without `maxcus`:
 - `bsub -n 11,60 myjob`
In this case, the job is allocated as close to 60 slots as possible, with a minimum of 11 slots.
 - `bsub -n 64 -R "cu[balance:maxcus=4:type=enclosure]" myjob`
Spans the fewest possible compute units for a balanced allocation of 64 slots using 4 or less compute units of type enclosure. Possible balanced allocations (in order of preference) are:
 - 64 slots on 1 enclosure
 - 32 slots on 2 enclosures
 - 22 slots on 1 enclosure and 21 slots on 2 enclosures
 - 16 slots on 4 enclosures
 - `bsub -n 64 -R "cu[excl:maxcus=8:usablecuslots=10]" myjob`
Allocates 64 slots over 8 or less compute units in groups of 10 or more slots per compute unit (with one compute unit possibly using less than 10 slots). The default compute unit type set in `COMPUTE_UNIT_TYPES` is used, and are used exclusively by `myjob`.
 - `bsub -n 58 -R "cu[balance:type=rack:usablecuslots=20]" myjob`
Provides a balanced allocation of 58 slots with at least 20 slots in each compute unit of type rack. Possible allocations are 58 slots in 1 rack or 29 slots in 2 racks.
- Jobs submitted with `balance` requirements choose compute units based on the `pref` keyword secondarily, as shown in the following examples where `cu1` has 5 available slots and `cu2` has 19 available slots.
- `bsub -n 5 -R "cu[balance:pref=minavail]"`
Runs the job on compute unit `cu1` where there are the fewest available slots.
 - `bsub -n 5 -R "cu[balance:pref=maxavail]"`
Runs the job on compute unit `cu2` where there are the most available slots. In both cases the job is balanced over the fewest possible compute units.
 - `bsub -n 11,60 -R "cu[maxcus=2:type=enclosure]" -app resizable -ar myjob`
An autoresizable job that spans the fewest possible compute units for a total allocation of at least 11 slots using at most 2 compute units of type enclosure. If the autoresizable job grows, the entire job still uses at most 2 compute units of type enclosure.
 - `bsub -n 64 -R "cu[balance:maxcus=4:type=enclosure]" -app resizable -ar myjob`
An autoresizable job that spans the fewest possible compute units for a balanced allocation of 64 slots using 4 or less compute units of type enclosure. If the autoresizable job grows, each subsequent allocation is a balanced allocation. The entire job (that is, the total of the initial and subsequent job allocations) still uses at most 4 compute units of type enclosure, but the job as a whole might not be a balanced allocation.
 - `bsub -n 64 -R "cu[excl:maxcus=8:usablecuslots=10]" -app resizable -ar myjob`
An autoresizable job that allocates 64 slots over 8 or less compute units in groups of 10 or more slots per compute unit (with one compute unit possibly using less than 10 slots). If the autoresizable job grows, each subsequent allocation

allocates in groups of 10 or more slots per compute unit (with one compute unit possible using less than 10 slots) and the entire job (that is, the total of the initial and subsequent job allocations) still uses at most 8 compute units. Since each subsequent allocation might have one compute unit that uses less than 10 slots, the entire job might have more than one compute unit that uses less than 10 slots. The default compute unit type set in `COMPUTE_UNIT_TYPES` is used, and are used exclusively by `myjob`.

CU string in application profiles

See [Resource requirements](#) for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

Affinity string

An *affinity resource requirement* string specifies CPU and memory binding requirements for the tasks of jobs. An affinity[] resource requirement section controls CPU and memory resource allocations and specifies the distribution of *processor units* within a host according to the hardware topology information that LSF collects.

affinity sections are accepted by **bsub -R**, and by **bmod -R** for non-running jobs, and can be specified in the `RES_REQ` parameter in `lsb.applications` and `lsb.queues`.

Syntax

The affinity string supports the following syntax:

```
affinity[pu_type[*count] | [pu_type(pu_num[,pu_options])[*count]] [:cpubind=numa | socket | core | thread]
[:membind=localonly | localprefer] [:distribute=task_distribution]]
```

```
pu_type[*count] | [pu_type(pu_num[,pu_options])[*count]]
```

Requested processor unit for the job tasks are specified by *pu_type*, which indicates the type and number of processor units the tasks can run on. Processor unit type can be one of `numa`, `socket`, `core`, or `thread`. *pu_num* specifies the number of processor units for each task.

For compatibility with IBM LoadLeveler, options `mcm` and `cpu` are also supported. `mcm` is an alias for the `numa` processor unit type, and `cpu` is an alias for the `thread` processor unit type.

For example, the following affinity requirement requests 5 cores per task:

```
affinity[core(5)]
```

Further processor unit specification is provided by *pu_options*, which have the following syntax:

```
same=level[,exclusive=(level[,scope])]
```

where:

same=level

Controls where processor units are allocated from. Processor unit *level* can be one of `numa`, `socket`, `core`, or `thread`. The *level* for *same* must be higher than the specified processor unit type.

For example, the following requests 2 threads from the same core: `affinity[thread(2,same=core)]`

```
"exclusive=(level[,scope][|scope])"
```

Constrains what level processor units can be allocated exclusively to a job or task. The *level* for *exclusive* can be one of `numa`, `socket`, or `core`. The *scope* for *exclusive* can be one of the following, or a combination separated by a logical OR (`|`):

- `intask` means that the allocated processor unit cannot be shared by different allocations in the same task.
- `injob` means that the allocated processor unit cannot be shared by different tasks in the same job.

- `alljobs` means that the allocated processor unit cannot be shared by different jobs. `alljobs` scope can only be used if `EXCLUSIVE=Y` is configured in the queue.

For example, the following requests 2 threads for each task from the same core, exclusively to the socket. No other tasks in the same job can run on the allocated socket (other jobs or tasks from other jobs can run on that socket): `affinity[thread(2, same=core, exclusive=(socket, injob))]`

Note: `EXCLUSIVE=Y` or `EXCLUSIVE=CU[cu_type]` must be configured in the queue to enable affinity jobs to use CPUs exclusively, when the `alljobs` scope is specified in the `exclusive` option.

**count*

Specifies a multiple of processor unit requests. This is convenient for requesting the same processor unit allocation for a number of tasks.

For example, the following affinity request allocates 4 threads per task from 2 cores, 2 threads in each core. The cores must come from different sockets:

```
affinity[thread(2, same=core, exclusive=(socket, intask))*2]
```

`cpubind=numa | socket | core | thread`

Specifies the CPU binding policy for tasks. If the level of `cpubind` is the same as or lower than the specified processor unit type (*pu_type*), the lowest processor unit is used. If the level of `cpubind` is higher than the requested processor type, the entire processor unit containing the allocation is used for CPU binding.

For example:

- `affinity[core(2):cpubind=thread]`

If the allocated cores are `/0/0/0` and `/0/0/1`, the CPU binding list will contain all threads under `/0/0/0` and `/0/0/1`.

- `affinity[core(2):cpubind=socket]`

If the allocated cores are `/0/0/0` and `/0/0/1`, the CPU binding list will contain all threads under the socket `/0/0`.

`membind=localonly | localprefer`

Specifies the physical NUMA memory binding policy for tasks.

- `localonly` limits the processes within the policy to allocate memory only from the local NUMA node. Memory is allocated if the available memory is greater than or equal to the memory requested by the task.
- `localprefer` specifies that LSF should try to allocate physical memory from the local NUMA node first. If this is not possible, LSF allocates memory from a remote NUMA node. Memory is allocated if the available memory is greater than zero.

`distribute=task_distribution`

Specifies how LSF distributes tasks of a submitted job on a host. Specify *task_distribution* according to the following syntax:

`pack | pack(type=1)`

LSF attempts to pack tasks in the same job on as few processor units as possible, in order to make processor units available for later jobs with the same binding requirements.

`pack (type=1)` forces LSF to pack all tasks for the job into the processor unit specified by *type*, where *type* is one of `numa`, `socket`, `core`, or `thread`. The difference between `pack` and `pack (type=1)` is that LSF will pend the job if `pack (type=1)` cannot be satisfied.

Use `pack` to allow your application to use memory locality.

For example, a job has the following affinity requirements:

```
bsub -n 6 -R "span[hosts=1] affinity[core(1):distribute=pack]"
```

The job asks for 6 slots, running on an single host. Each slot maps to 1 core, and LSF tries to pack all 6 cores as close as possible on a single NUMA or socket.

The following example packs all job tasks on a single NUMA node:

```
affinity[core(1,exclusive=(socket,injob)):distribute=pack(uma=1)]
```

In this allocation, each task needs 1 core and no other tasks from the same job can allocate CPUs from the same socket. All tasks are packed in the same job on one NUMA node.

balance

LSF attempts to distribute job tasks equally across all processor units. Use balance to make as many processor units available to your job as possible.

any

LSF attempts no job task placement optimization. LSF chooses the first available processor units for task placement.

Examples

```
affinity[core(5,same=uma):cpubind=uma:membind=localonly]
```

Each task requests 5 cores in the same NUMA node and binds the tasks on the NUMA node with memory mandatory binding.

The following binds a multithread job on a single NUMA node:

```
affinity[core(3,same=uma):cpubind=uma:membind=localprefer]
```

The following distributes tasks across sockets:

```
affinity[core(2,same=socket,exclusive=(socket,injob|alljobs)):cpubind=socket]
```

Each task needs 2 cores from the same socket and binds each task at the socket level. The allocated socket is exclusive - no other tasks can use it.

Affinity string in application profiles and queues

A job-level affinity string section overwrites an application-level section, which overwrites a queue-level section (if a given level is present).

See [Resource requirements](#) for information about how resource requirements in application profiles are resolved with queue-level and job-level resource requirements.

Specify GPU resource requirements for your jobs

Specify all GPU resource requirements as part of job submission, or in a queue or application profile, or use the default GPU requirement "num=1:mode=shared:mps=no;j_exclusive=no". Use the option `bsub -gpu` to submit jobs that require GPU resources. Specify how LSF manages GPU mode (exclusive or shared), and whether to enable the NVIDIA Multi-Process Service (MPS) for the GPUs used by the job.

See information on specifying GPU resource requirements in [GPU resources](#).

Reserving resources

- [About resource reservation](#)

When a job is dispatched, the system assumes that the resources that the job consumes will be reflected in the load information. However, many jobs do not consume the resources that they require when they first start. Instead, they will typically use the resources over a period of time.

- [Use resource reservation](#)

- [Memory reservation for pending jobs](#)

By default, the `rusage` string reserves resources for running jobs. Because resources are not reserved for pending jobs,

some memory-intensive jobs could be pending indefinitely because smaller jobs take the resources immediately before the larger jobs can start running. The more memory a job requires, the worse the problem is.

- **Time-based slot reservation**

Existing LSF slot reservation works in simple environments, where the host-based MXJ limit is the only constraint to job slot request. I

About resource reservation

When a job is dispatched, the system assumes that the resources that the job consumes will be reflected in the load information. However, many jobs do not consume the resources that they require when they first start. Instead, they will typically use the resources over a period of time.

For example, a job requiring 100 MB of swap is dispatched to a host having 150 MB of available swap. The job starts off initially allocating 5 MB and gradually increases the amount consumed to 100 MB over a period of 30 minutes. During this period, another job requiring more than 50 MB of swap should not be started on the same host to avoid over-committing the resource.

Resources can be reserved to prevent overcommitment by LSF. Resource reservation requirements can be specified as part of the resource requirements when submitting a job, or can be configured into the queue level resource requirements.

Pending job resize allocation requests are not supported in slot reservation policies. Newly added or removed resources are reflected in the pending job predicted start time calculation.

Resource reservation limits

Maximum and minimum values for consumable resource requirements can be set for individual queues, so jobs will only be accepted if they have resource requirements within a specified range. This can be useful when queues are configured to run jobs with specific memory requirements, for example. Jobs requesting more memory than the maximum limit for the queue will not be accepted, and will not take memory resources away from the smaller memory jobs the queue is designed to run.

Resource reservation limits are set at the queue level by the parameter RESRSV_LIMIT in lsb.queues.

How resource reservation works

When deciding whether to schedule a job on a host, LSF considers the reserved resources of jobs that have previously started on that host. For each load index, the amount reserved by all jobs on that host is summed up and subtracted (or added if the index is increasing) from the current value of the resources as reported by the LIM to get amount available for scheduling new jobs:

available amount = current value - reserved amount for all jobs

For example:

```
bsub -R "rusage[tmp=30:duration=30:decay=1]" myjob
```

will reserve 30 MB of temp space for the job. As the job runs, the amount reserved will decrease at approximately 1 MB/minute such that the reserved amount is 0 after 30 minutes.

Queue-level and job-level resource reservation

The queue level resource requirement parameter RES_REQ may also specify the resource reservation. If a queue reserves certain amount of a resource (and the parameter RESRSV_LIMIT is not being used), you cannot reserve a greater amount of that resource at the job level.

For example, if the output of **bqueues -l** command contains:

```
RES_REQ: rusage[mem=40:swp=80:tmp=100]
```

the following submission will be rejected since the requested amount of certain resources exceeds queue's specification:

```
bsub -R "rusage[mem=50:swp=100]" myjob
```

When both RES_REQ and RESRSV_LIMIT are set in lsb.queues for a consumable resource, the queue-level RES_REQ no longer acts as a hard limit for the merged RES_REQ rusage values from the job and application levels. In this case only the limits set by RESRSV_LIMIT must be satisfied, and the queue-level RES_REQ acts as a default value.

Use resource reservation

Queue-level resource reservation

At the queue level, resource reservation allows you to specify the amount of resources to reserve for jobs in the queue. It also serves as the upper limits of resource reservation if a user also specifies it when submitting a job.

Queue-level resource reservation and pending reasons

The use of RES_REQ affects the pending reasons as displayed by **bjobs**. If RES_REQ is specified in the queue and the loadSched thresholds are not specified, then the pending reasons for each individual load index will not be displayed.

- [Configuring resource reservation at the queue level](#)
- [Specifying job-level resource reservation](#)
- [Configuring per-resource reservation](#)

Configuring resource reservation at the queue level

About this task

Queue-level resource reservations and resource reservation limits can be configured as parameters in lsb.queues.

Procedure

Specify the amount of resources a job should reserve after it is started in the resource usage (rusage) section of the resource requirement string of the QUEUE section.

Examples

```
Begin Queue
...
RES_REQ = select[type==any] rusage[swp=100:mem=40:duration=60]
RESRSV_LIMIT = [mem=30,100]
...
End Queue
```

This allows a job to be scheduled on any host that the queue is configured to use and reserves 100 MB of swap and 40 MB of memory for a duration of 60 minutes. The requested memory reservation of 40 MB falls inside the allowed limits set by RESRSV_LIMIT of 30 MB to 100 MB.

```
Begin Queue
...
RES_REQ = select[type==any] rusage[mem=20 | mem=10:swp=20]
...
End Queue
```

This allows a job to be scheduled on any host that the queue is configured to use. The job attempts to reserve 20 MB of memory, or 10 MB of memory and 20 MB of swap if the 20 MB of memory is unavailable. In this case no limits are defined by RESRSV_LIMIT.

Specifying job-level resource reservation

Procedure

To specify resource reservation at the job level, use **bsub -R** and include the resource usage section in the resource requirement string.

Configuring per-resource reservation

Procedure

To enable greater flexibility for reserving numeric resources are reserved by jobs, configure the `ReservationUsage` section in `lsb.resources` to reserve resources as `PER_JOB`, `PER_TASK`, or `PER_HOST`. Only user-defined numeric resources can be reserved. Built-in resources such as `mem`, `cpu`, or `swp` cannot be configured in the `ReservationUsage` section.

The cluster-wide `RESOURCE_RESERVE_PER_TASK` parameter still controls resources that are not configured in `lsb.resources`. Resources not reserved in `lsb.resources` are reserved per job. Configuration in `lsb.resources` overrides `RESOURCE_RESERVE_PER_TASK` if it also exists for the same resource.

`PER_HOST` reservation means that for the parallel job, LSF reserves one instance of a for each host. For example, some applications are charged only once no matter how many applications are running provided those applications are running on the same host under the same user.

Note: Configuration `PER_SLOT` is obsolete as of LSF 9.1.3 and replaced by `PER_TASK`.

Assumptions and limitations

- Per-resource configuration defines resource usage for individual resources, but it does not change any existing resource limit behavior (`PER_JOB`, `PER_TASK`).
- In a MultiCluster environment, you should configure resource usage in the scheduling cluster (submission cluster in lease model or receiving cluster in job forward model).
- The keyword `pref` in the compute unit resource string is ignored, and the default configuration order is used (`pref=config`).

Memory reservation for pending jobs

By default, the `rusage` string reserves resources for running jobs. Because resources are not reserved for pending jobs, some memory-intensive jobs could be pending indefinitely because smaller jobs take the resources immediately before the larger jobs can start running. The more memory a job requires, the worse the problem is.

Memory reservation for pending jobs solves this problem by reserving memory as it becomes available until the total required memory specified on the `rusage` string is accumulated and the job can start. Use memory reservation for pending jobs if memory-intensive jobs often compete for memory with smaller jobs in your cluster.

- [Reserving host memory for pending jobs](#)
- [Enabling memory reservation for sequential jobs](#)
- [Configuring lsb.queues](#)
- [Using memory reservation for pending jobs](#)
- [How memory reservation for pending jobs works](#)

Reserving host memory for pending jobs

Procedure

Use the `RESOURCE_RESERVE` parameter in `lsb.queues` to reserve host memory for pending jobs. The amount of memory reserved is based on the currently available memory when the job is pending. Reserved memory expires at the end of the time period represented by the number of dispatch cycles specified by the value of `MAX_RESERVE_TIME` set on the `RESOURCE_RESERVE` parameter.

Enabling memory reservation for sequential jobs

Procedure

Add the LSF scheduler plug-in module name for resource reservation (`schmod_reserve`) to the `lsb.modules` file:

```
Begin PluginModule
SCH_PLUGIN          RB_PLUGIN          SCH_DISABLE_PHASES
schmod_default      ()                ()
schmod_reserve      ()                ()
schmod_preemption   ()                ()
End PluginModule
```

Configuring lsb.queues

Procedure

Set the `RESOURCE_RESERVE` parameter in a queue defined in `lsb.queues`. If both `RESOURCE_RESERVE` and `SLOT_RESERVE` are defined in the same queue, job slot reservation and memory reservation are both enabled and an error is displayed when the cluster is reconfigured. `SLOT_RESERVE` is ignored.

Example queues

The following queue enables memory reservation for pending jobs:

```
Begin Queue
QUEUE_NAME = reservation
DESCRIPTION = For resource reservation
PRIORITY=40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
End Queue
```

Using memory reservation for pending jobs

Procedure

Use the `rusage` string in the **-R** option to `bsub` or the `RES_REQ` parameter in `lsb.queues` to specify the amount of memory required for the job. Submit the job to a queue with `RESOURCE_RESERVE` configured.

Note:

Compound resource requirements do not support use of the `||` operator within the component `rusage` simple resource requirements, multiple `-R` options, or the `cu` section.

How memory reservation for pending jobs works

Amount of memory reserved

The amount of memory reserved is based on the currently available memory when the job is pending. For example, if LIM reports that a host has 300 MB of memory available, the job submitted by the following command:

```
bsub -R "rusage[mem=400]" -q reservation my_job
```

will be pending and reserve the 300 MB of available memory. As other jobs finish, the memory that becomes available is added to the reserved memory until 400 MB accumulates, and the job starts.

No memory is reserved if no job slots are available for the job because the job could not run anyway, so reserving memory would waste the resource.

Only memory is accumulated while the job is pending; other resources specified on the rusage string are only reserved when the job is running. Duration and decay have no effect on memory reservation while the job is pending.

How long memory is reserved (MAX_RESERVE_TIME)

Reserved memory expires at the end of the time period represented by the number of dispatch cycles specified by the value of MAX_RESERVE_TIME set on the RESOURCE_RESERVE parameter. If a job has not accumulated enough memory to start by the time MAX_RESERVE_TIME expires, it releases all its reserved memory so that other pending jobs can run. After the reservation time expires, the job cannot reserve slots or memory for one scheduling session, so other jobs have a chance to be dispatched. After one scheduling session, the job can reserve available resources again for another period that is specified by MAX_RESERVE_TIME.

Examples

lsb.queues

The following queues are defined in lsb.queues:

```
Begin Queue  
QUEUE_NAME = reservation  
DESCRIPTION = For resource reservation  
PRIORITY=40  
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]  
End Queue
```

Assumptions

Assume one host in the cluster with 10 CPUs and 1 GB of free memory currently available.

Sequential jobs

Each of the following sequential jobs requires 400 MB of memory and runs for 300 minutes.

Job 1:

```
bsub -W 300 -R "rusage[mem=400]" -q reservation myjob1
```

The job starts running, using 400M of memory and one job slot.

Job 2:

Submitting a second job with same requirements yields the same result.

Job 3:

Submitting a third job with same requirements reserves one job slot, and reserves all free memory, if the amount of free memory is between 20 MB and 200 MB (some free memory may be used by the operating system or other software.)

Time-based slot reservation

Existing LSF slot reservation works in simple environments, where the host-based MXJ limit is the only constraint to job slot request. I

n complex environments, where more than one constraint exists (for example job topology or generic slot limit):

- Estimated job start time becomes inaccurate
- The scheduler makes a reservation decision that can postpone estimated job start time or decrease cluster utilization.

Current slot reservation by start time (RESERVE_BY_STARTTIME) resolves several reservation issues in multiple candidate host groups, but it cannot help on other cases:

- Special topology requests, like `span[ptile=n]` and `cu[]` keywords balance, maxcus, and excl.
- Only calculates and displays reservation if host has free slots. Reservations may change or disappear if there are no free CPUs; for example, if a backfill job takes all reserved CPUs.
- For HPC machines containing many internal nodes, host-level number of reserved slots is not enough for administrator and end user to tell which CPUs the job is reserving and waiting for.
- [Configuring time-based slot reservation](#)
- [Time-based slot reservation assumptions and limitations](#)
Time-base slot reservations have assumptions and limitations.
- [Reservation scenarios](#)
- [Reservation examples](#)

Time-based slot reservation versus greedy slot reservation

With time-based reservation, a set of pending jobs gets future allocation and an estimated start time so that the system can reserve a place for each job. Reservations use the estimated start time, which is based on future allocations.

Time-based resource reservation provides a more accurate predicted start time for pending jobs because LSF considers job scheduling constraints and requirements, including job topology and resource limits, for example.

Restriction: Time-based reservation does not work with job chunking.

Start time and future allocation

The estimated start time for a future allocation is the earliest start time when all considered job constraints are satisfied in the future. There may be a small delay of a few minutes between the job finish time on which the estimate was based and the actual start time of the allocated job.

For compound resource requirement strings, the predicted start time is based on the simple resource requirement term (contained in the compound resource requirement) with the latest predicted start time.

If a job cannot be placed in a future allocation, the scheduler uses *greedy* slot reservation to reserve slots. Existing LSF slot reservation is a simple greedy algorithm:

- Only considers current available resources and minimal number of requested job slots to reserve as many slots as it is allowed
- For multiple exclusive candidate host groups, scheduler goes through those groups and makes reservation on the group that has the largest available slots
- For estimated start time, after making reservation, scheduler sorts all running jobs in ascending order based on their finish time and goes through this sorted job list to add up slots used by each running job till it satisfies minimal job slots request. The finish time of last visited job will be job estimated start time.

Reservation decisions made by greedy slot reservation do not have an accurate estimated start time or information about future allocation. The calculated job start time used for backfill scheduling is uncertain, so **bjobs** displays:

Job will start no sooner than indicated time stamp

Time-based reservation and greedy reservation compared

Start time prediction	Time-based reservation	Greedy reservation
Backfill scheduling if free slots are available	Yes	Yes
Correct with no job topology	Yes	Yes
Correct for job topology requests	Yes	No
Correct based on resource allocation limits	Yes (guaranteed if only two limits are defined)	No
Correct for memory requests	Yes	No
When no slots are free for reservation	Yes	No
Future allocation and reservation based on earliest start time	Yes	No
bjobs displays best estimate	Yes	No
bjobs displays predicted future allocation	Yes	No
Absolute predicted start time for all jobs	No	No
Advance reservation considered	No	No

Greedy reservation example

A cluster has four hosts: A, B, C, and D, with 4 CPUs each. Four jobs are running in the cluster: Job1, Job2, Job3 and Job4. According to calculated job estimated start time, the job finish times (FT) have this order: $FT(\text{Job2}) < FT(\text{Job1}) < FT(\text{Job4}) < FT(\text{Job3})$.

A	B	C	D
Job2	Job3	Job4	Job3
Job2	Job3	Job4	Job3
Job2	Job3	Job2	
	Job3	Job1	Job1

Job CPU in use

Available CPU

Now, a user submits a high priority job. It pends because it requests $-n 6 -R \text{"span[ptile=2]"}$. This resource requirement means this pending job needs three hosts with two CPUs on each host. The default greedy slot reservation calculates job start time as the job finish time of Job4 because after Job4 finishes, three hosts with a minimum of two slots are available.

Greedy reservation indicates that the pending job starts no sooner than when Job 2 finishes.

In contrast, time-based reservation can determine that the pending job starts in 2 hours. It is a much more accurate reservation.

Configuring time-based slot reservation

About this task

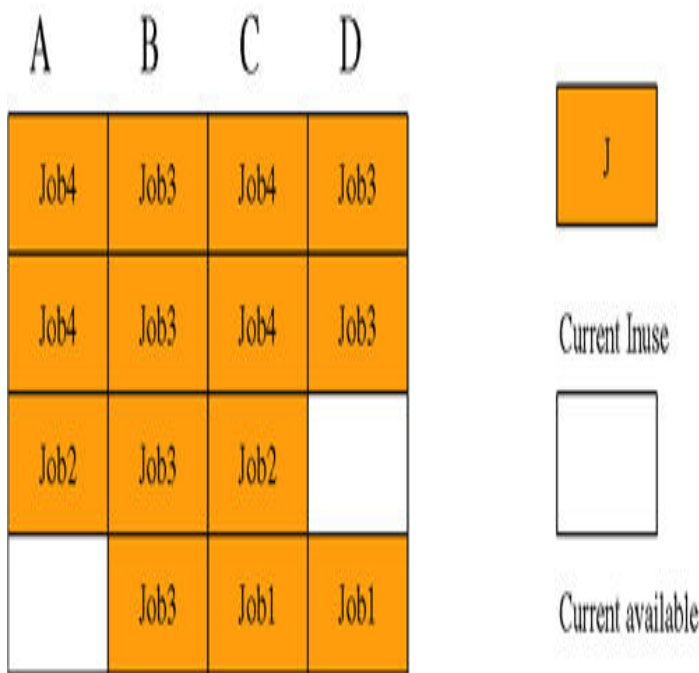
Greedy slot reservation is the default slot reservation mechanism and time-based slot reservation is disabled.

Procedure

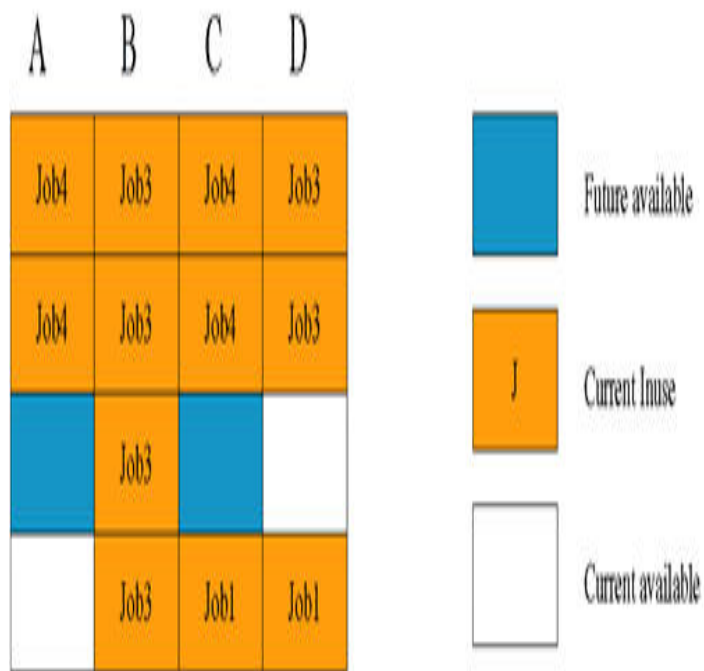
- 1. Use `LSB_TIME_RESERVE_NUMJOBS=maximum_reservation_jobs` in `lsf.conf` to enable time-based slot reservation. The value must be a positive integer.
LSB_TIME_RESERVE_NUMJOBS controls maximum number of jobs using time-based slot reservation. For example, if `LSB_TIME_RESERVE_NUMJOBS=4`, only the top 4 jobs will get their future allocation information.
- 2. Use `LSB_TIME_RESERVE_NUMJOBS=1` to allow only the highest priority job to get accurate start time prediction. Smaller values are better than larger values because after the first pending job starts, the estimated start time of remaining jobs may be changed. For example, you could configure `LSB_TIME_RESERVE_NUMJOBS` based on the number of exclusive host partitions or host groups.

Scheduling examples

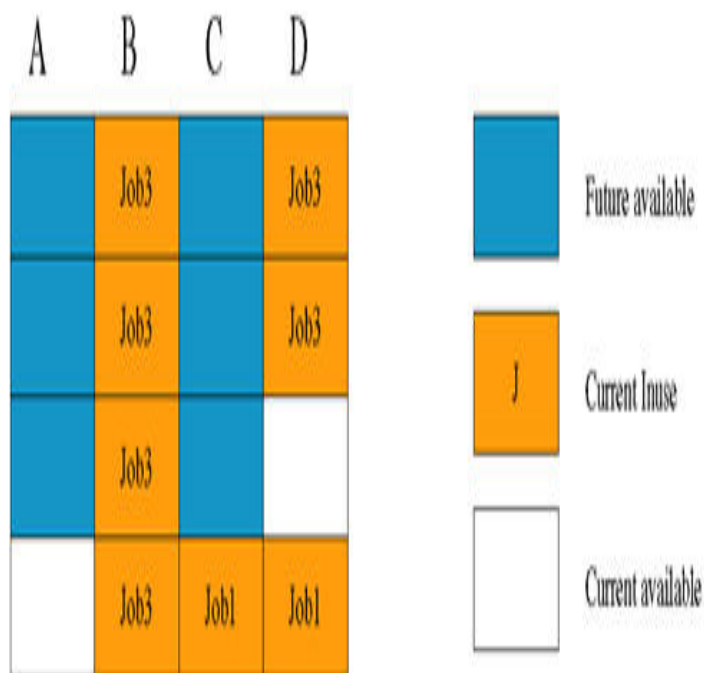
- 1. Job5 requests `-n 6 -R "span[ptile=2]"`, which will require three hosts with 2 CPUs on each host. As in the greedy slot reservation example, four jobs are running in the cluster: Job1, Job2, Job3 and Job4. Two CPUs are available now, 1 on host A, and 1 on host D:



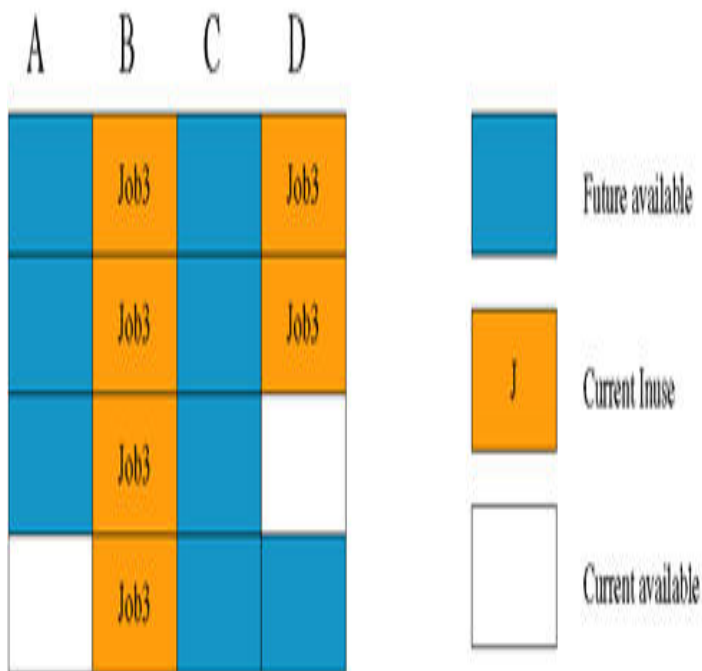
- 2. Job2 finishes, freeing 2 more CPUs for future allocation, 1 on host A, and 1 on host C:



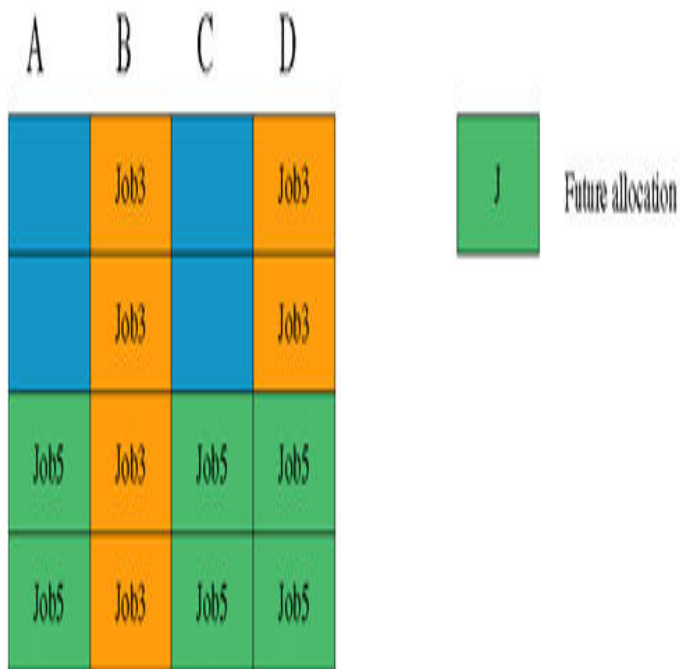
3. Job4 finishes, freeing 4 more CPUs for future allocation, 2 on host A, and 2 on host C:



4. Job1 finishes, freeing 2 more CPUs for future allocation, 1 on host C, and 1 host D:



5. Job5 can now be placed with 2 CPUs on host A, 2 CPUs on host C, and 2 CPUs on host D. The estimated start time is shown as the finish time of Job1:



Time-based slot reservation assumptions and limitations

Time-base slot reservations have assumptions and limitations.

- To get an accurate estimated start time, you must specify a run limit at the job level using the **bsub -W** option, in the queue by configuring RUNLIMIT in lsb.queues, or in the application by configuring RUNLIMIT in lsb.applications, or you must specify a run time estimate by defining the RUNTIME parameter in lsb.applications. If a run limit or a run time estimate is not defined, the scheduler will try to use CPU limit instead.
- Estimated start time is only relatively accurate according to current running job information. If running jobs finish earlier, estimated start time may be moved to earlier time. Only the highest priority job will get accurate predicted start time.

The estimated start time for other jobs could be changed after the first job starts.

- Under time-based slot reservation, only information from currently running jobs is used for making reservation decisions.
- Estimated start time calculation does not consider Deadline scheduling.
- Estimated start time calculation does not consider Advance Reservation.
- Estimated start time calculation does not consider DISPATCH_WINDOW in lsb.hosts and lsb.queue configuration.
- If preemptive scheduling is used, the estimated start time may not be accurate. The scheduler may calculate and estimated time, but actually it may preempt other jobs to start earlier.
- For resizable jobs, time-based slot reservation does not schedule pending resize allocation requests. However, for resized running jobs, the allocation change is used when calculating pending job predicted start time and resource reservation. For example, if a running job uses 4 slots at the beginning, but added another 4 slots, after adding the new resources, LSF expects 8 slots to be available after the running job completes.

Slot limit enforcement

The following slot limits are enforced:

- Slot limits configured in lsb.resources (SLOTS, PER_SLOT)
- MXJ, JL/U in lsb.hosts
- PJOB_LIMIT, HJOB_LIMIT, QJOB_LIMIT, UJOB_LIMIT in lsb.queues

Memory request

To request memory resources, configure RESOURCE_RESERVE in lsb.queues.

When RESOURCE_RESERVE is used, LSF will consider memory and slot requests during time-based reservation calculation. LSF will not reserve slot or memory if any other resources are not satisfied.

If SLOT_RESERVE is configured, time-based reservation will not make a slot reservation if any other type of resource is not satisfied, including memory requests.

When SLOT_RESERVE is used, if job cannot run because of non-slot resources, including memory, time-based reservation will not reserve slots.

Host partition and queue-level scheduling

If host partitions are configured, LSF first schedules jobs on the host partitions and then goes through each queue to schedule jobs. The same job may be scheduled several times, one for each host partition and last one at queue-level. Available candidate hosts may be different for each time.

Because of this difference, the same job may get different estimated start times, future allocation, and reservation in different host partitions and queue-level scheduling. With time-based reservation configured, LSF always keeps the same reservation and future allocation with the earliest estimated start time.

bjobs displays future allocation information

- By default, job future allocation contains LSF host list and number of CPUs per host, for example: `alloc=2*hostA 3*hostB`
- LSF integrations define their own future allocation string to override the default LSF allocation. For example, in cpuset, future allocation is displayed as:

```
alloc=2*mstatx01 2*mstatx00
```

Predicted start time may be postponed for some jobs

If a pending job cannot be placed in a future resource allocation, the scheduler can skip it in the start time reservation calculation and fall back to use greedy slot reservation. There are two possible reasons:

- The job slot request cannot be satisfied in the future allocation

- Other non-slot resources cannot be satisfied.

Either way, the scheduler continues calculating predicted start time for the remaining jobs without considering the skipped job.

Later, once the resource request of skipped job can be satisfied and placed in a future allocation, the scheduler reevaluates the predicted start time for the rest of jobs, which may potentially postpone their start times.

To minimize the overhead in recalculating the predicted start times to include previously skipped jobs, you should configure a small value for `LSB_TIME_RESERVE_NUMJOBS` in `lsf.conf`.

Reservation scenarios

Scenario 1

Even though no running jobs finish and no host status in cluster are changed, a job's future allocation may still change from time to time.

Why this happens

Each scheduling cycle, the scheduler recalculates a job's reservation information, estimated start time, and opportunity for future allocation. The job candidate host list may be reordered according to current load. This reordered candidate host list will be used for the entire scheduling cycle, also including job future allocation calculation. So different order of candidate hosts may lead to different result of job future allocation. However, the job estimated start time should be the same.

For example, there are two hosts in cluster, `hostA` and `hostB`. 4 CPUs per host. Job 1 is running and occupying 2 CPUs on `hostA` and 2 CPUs on `hostB`. Job 2 requests 6 CPUs. If the order of hosts is `hostA` and `hostB`, then the future allocation of job 2 will be 4 CPUs on `hostA` 2 CPUs on `hostB`. If the order of hosts changes in the next scheduling cycle changes to `hostB` and `hostA`, then the future allocation of job 2 will be 4 CPUs on `hostB` 2 CPUs on `hostA`.

Scenario 2:

If you set `JOB_ACCEPT_INTERVAL` to non-zero value, after job is dispatched, within `JOB_ACCEPT_INTERVAL` period, pending job estimated start time and future allocation may momentarily fluctuate.

Why this happens

The scheduler does a time-based reservation calculation each cycle. If `JOB_ACCEPT_INTERVAL` is set to non-zero value. Once a new job has been dispatched to a host, this host will not accept new job within `JOB_ACCEPT_INTERVAL` interval. Because the host will not be considered for the entire scheduling cycle, no time-based reservation calculation is done, which may result in slight change in job estimated start time and future allocation information. After `JOB_ACCEPT_INTERVAL` has passed, host will become available for time-based reservation calculation again, and the pending job estimated start time and future allocation will be accurate again.

Reservation examples

Example 1

Three hosts, 4 CPUs each: `qat24`, `qat25`, and `qat26`. Job 11895 uses 4 slots on `qat24` (10 hours). Job 11896 uses 4 slots on `qat25` (12 hours), and job 11897 uses 2 slots on `qat26` (9 hours).

Job 11898 is submitted and requests `-n 6 -R "span[ptile=2]"`.

```
bjobs -l 11898
```

```
Job <11898>, User <user2>, Project <default>, Status <PEND>, Queue <challenge>,
```

```

Job Priority <50>, Command <sleep 100000000>
...
RUNLIMIT
  840.0 min of hostA
Fri Apr 22 15:18:56 2010: Reserved <2> job slots on host(s) <2*qat26>;
Sat Apr 23 03:28:46 2010: Estimated Job Start Time;
                          alloc=2*qat25 2*qat24 2*qat26.lsf.platform.com
...

```

Example 2

Two cpuset hosts, `mstatx00` and `mstatx01`, 8 CPUs per host. Job 3873 uses 4*mstatx00 and will last for 10 hours. Job 3874 uses 4*mstatx01 and will run for 12 hours. Job 3875 uses 2*mstatx02 and 2*mstatx03, and will run for 13 hours.

Job 3876 is submitted and requests `-n 4 -ext "cpuset[nodes=2]" -R "rusage[mem=100] span[ptile= 2]"`.

```

bjobs -l 3876
Job <3876>, User <user2>, Project <default>, Status <PEND>, Queue <sq32_s>, Command <sleep
33333>
Tue Dec 22 04:56:54: Submitted from host <mstatx00>, CWD <$HOME>, 4 Processors Requested,
Requested Resources <rusage[mem=100] span[ptile= 2]>;
...
RUNLIMIT
60.0 min of mstatx00 Tue Dec 22 06:07:38: Estimated job start time; alloc=2*mstatx01
2*mstatx00 ...

```

Example 3

Rerun example 1, but this time, use greedy slot reservation instead of time-based reservation:

```

bjobs -l 3876
Job <12103>, User <user2>, Project <default>, Status <PEND>, Queue <challenge>,
Job Priority <50>, Command <sleep 1000000>
Fri Apr 22 16:17:59 2010: Submitted from host <qat26>, CWD <$HOME>, 6 Processors Req
uested, Requested Resources <span[ptile=2]>;
...
RUNLIMIT
  720.0 min of qat26
Fri Apr 22 16:18:09 2010: Reserved <2> job slots on host(s) <2*qat26.lsf.platform.com>;
Sat Apr 23 01:39:13 2010: Job will start no sooner than indicated time stamp;
...

```

Limiting job resource allocations

Resource allocation limits configured in the `lsb.resources` file restrict the maximum amount of a resource requested by a job that can be allocated during job scheduling for different classes of jobs to start. Configured limits also specify which resource consumers the limits apply to. Configure all resource allocation limits in one or more `Limit` sections in the `lsb.resources` file.

- [How resource allocation limits work](#)
By default, resource consumers like users, hosts, queues, or projects are not limited in the resources available to them for running jobs.
- [How job limits work](#)
The `JOBS` parameter limits the maximum number of running or suspended jobs available to resource consumers. Limits are enforced depending on the number of jobs in `RUN`, `SSUSP`, and `USUSP` state.
- [Configuring resource allocation limits](#)
Configure all resource allocation limits in one or more `Limit` sections in the `lsb.resources` file. Limit sections set limits for how much of the specified resources must be available for different classes of jobs to start, and which resource consumers the limits apply to.
- [Creating a limit using `bconf`](#)

How resource allocation limits work

By default, resource consumers like users, hosts, queues, or projects are not limited in the resources available to them for running jobs.

Resource allocation limits configured in `lsb.resources` specify the following restrictions:

- The maximum amount of a resource requested by a job that can be allocated during job scheduling for different classes of jobs to start
- Which resource consumers the limits apply to

If all of the resource has been consumed, no more jobs can be started until some of the resource is released.

For example, by limiting maximum amount of memory for each of your hosts, you can make sure that your system operates at optimal performance. By defining a memory limit for some users submitting jobs to a particular queue and a specified set of hosts, you can prevent these users from using up all the memory in the system at one time.

Jobs must specify resource requirements

For limits to apply, the job must specify resource requirements (**`bsub -R`** `rusage` string or the `RES_REQ` parameter in the `lsb.queues` file). For example, the a memory allocation limit of 4 MB is configured in `lsb.resources`:

```
Begin Limit
NAME = mem_limit1
MEM = 4
End Limit
```

A job submitted with an `rusage` resource requirement that exceeds this limit:

```
bsub -R "rusage[mem=5]" " uname
```

and remains pending:

```
bjobs -p 600
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
 600   user1  PEND  normal  suplin02                uname      Aug 12 14:05
Resource (mem) limit defined cluster-wide has been reached;
```

A job is submitted with a resource requirement within the configured limit:

```
bsub -R"rusage[mem=3]" " sleep 100
```

is allowed to run:

```
bjobs
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
 600   user1  PEND  normal  hostA                uname      Aug 12 14:05
 604   user1  RUN   normal  hostA                sleep 100  Aug 12 14:09
```

Resource usage limits and resource allocation limits

Resource allocation limits are not the same as *resource usage limits*, which are enforced during job run time. For example, you set CPU limits, memory limits, and other limits that take effect after a job starts running.

Resource reservation limits and resource allocation limits

Resource allocation limits are not the same as queue-based *resource reservation limits*, which are enforced during job submission. The parameter `RESRSV_LIMIT` in the `lsb.queues` file specifies allowed ranges of resource values, and jobs submitted with resource requests outside of this range are rejected.

How LSF enforces limits

Resource allocation limits are enforced so that they apply to all jobs in the cluster according to the kind of resources, resource consumers, and combinations of consumers.

- All jobs in the cluster
- Several kinds of resources:
 - Job slots by host
 - Job slots per processor
 - Running and suspended jobs
 - Memory (MB or percentage)
 - Swap space (MB or percentage)
 - Tmp space (MB or percentage)
 - Other shared resources
- Several kinds of resource consumers:
 - Users and user groups (all users or per-user)
 - Hosts and host groups (all hosts or per-host)
 - Queues (all queues or per-queue)
 - Projects (all projects or per-project)
- Combinations of consumers:
 - For jobs running on different hosts in the same queue
 - For jobs running from different queues on the same host

How LSF counts resources

Resources on a host are not available if they are taken by jobs that have been started, but have not yet finished. This means running and suspended jobs count against the limits for queues, users, hosts, projects, and processors that they are associated with.

Job slot limits

Job slot limits can correspond to the maximum number of jobs that can run at any point in time. For example, a queue cannot start jobs if it has no job slots available, and jobs cannot run on hosts that have no available job slots.

Limits such as QJOB_LIMIT (lsb.queues), HJOB_LIMIT (lsb.queues), UJOB_LIMIT (lsb.queues), MXJ (lsb.hosts), JL/U (lsb.hosts), MAX_JOBS (lsb.users), and MAX_PEND_SLOTS (lsb.users and lsb.params) limit the number of job slots. When the workload is sequential, job slots are usually equivalent to jobs. For parallel or distributed applications, these are true job slot limits and not job limits.

Job limits

Job limits, specified by JOBS in a Limit section in lsb.resources, correspond to the maximum number of running and suspended jobs that can run at any point in time. MAX_PEND_JOBS (lsb.users and lsb.params) limit the number of pending jobs. If both job limits and job slot limits are configured, the most restrictive limit is applied.

Resource reservation and backfill

When processor or memory reservation occurs, the reserved resources count against the limits for users, queues, hosts, projects, and processors. When backfilling of parallel jobs occurs, the backfill jobs do not count against any limits.

IBM® Spectrum LSF multicluster capability

Limits apply only to the cluster where the lsb.resources file is configured. If the cluster leases hosts from another cluster, limits are enforced on those hosts as if they were local hosts.

Switched jobs can exceed resource allocation limits

If a switched job (the **bswitch** command) has not been dispatched, then the job behaves as if it were submitted to the new queue in the first place, and the JOBS limit is enforced in the target queue.

If a switched job has been dispatched, then resource allocation limits like SWP, TMP, and JOBS can be exceeded in the target queue. For example, given the following JOBS limit configuration:

```
Begin Limit
USERS      QUEUES      SLOTS      TMP      JOBS
```

```
-          normal      -      20      2
-          short       -      20      2
End Limit
```

Submit 3 jobs to the normal queue, and 3 jobs to the short queue:

```
bsub -q normal -R"rusage[tmp=20]" sleep 1000
bsub -q short -R"rusage[tmp=20]" sleep 1000
```

bjobs shows 1 job in RUN state in each queue:

```
bjobs
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
16     user1  RUN   normal hosta        hosta    sleep 1000  Aug 30 16:26
17     user1  PEND  normal hosta        hosta    sleep 1000  Aug 30 16:26
18     user1  PEND  normal hosta        hosta    sleep 1000  Aug 30 16:26
19     user1  RUN   short  hosta        hosta    sleep 1000  Aug 30 16:26
20     user1  PEND  short  hosta        hosta    sleep 1000  Aug 30 16:26
21     user1  PEND  short  hosta        hosta    sleep 1000  Aug 30 16:26
```

blimits shows the TMP limit reached:

```
blimits
INTERNAL RESOURCE LIMITS:
NAME    USERS  QUEUES  SLOTS    TMP    JOBS
NONAME000 -      normal    -      20/20    1/2
NONAME001 -      short     -      20/20    1/2
```

Switch the running job in the normal queue to the short queue:

```
bswitch short 16
```

The **bjobs** command shows 2 jobs running in the short queue, and the second job running in the normal queue:

```
bjobs
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
17     user1  RUN   normal hosta        hosta    sleep 1000  Aug 30 16:26
18     user1  PEND  normal hosta        hosta    sleep 1000  Aug 30 16:26
19     user1  RUN   short  hosta        hosta    sleep 1000  Aug 30 16:26
16     user1  RUN   short  hosta        hosta    sleep 1000  Aug 30 16:26
20     user1  PEND  short  hosta        hosta    sleep 1000  Aug 30 16:26
21     user1  PEND  short  hosta        hosta    sleep 1000  Aug 30 16:26
```

The **blimits** command shows the TMP limit exceeded and the JOBS limit reached in the short queue:

```
blimits
INTERNAL RESOURCE LIMITS:
NAME    USERS  QUEUES  SLOTS    TMP    JOBS
NONAME000 -      normal    -      20/20    1/2
NONAME001 -      short     -      40/20    2/2
```

Switch the running job in the normal queue to the short queue:

```
bswitch short 17
```

The **bjobs** command shows 3 jobs running in the short queue and the third job running in the normal queue:

```
bjobs
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
18     user1  RUN   normal hosta        hosta    sleep 1000  Aug 30 16:26
19     user1  RUN   short  hosta        hosta    sleep 1000  Aug 30 16:26
16     user1  RUN   short  hosta        hosta    sleep 1000  Aug 30 16:26
17     user1  RUN   short  hosta        hosta    sleep 1000  Aug 30 16:26
20     user1  PEND  short  hosta        hosta    sleep 1000  Aug 30 16:26
21     user1  PEND  short  hosta        hosta    sleep 1000  Aug 30 16:26
```

The **blimits** command shows both TMP and JOBS limits exceeded in the short queue:

```
blimits
INTERNAL RESOURCE LIMITS:
NAME    USERS  QUEUES  SLOTS    TMP    JOBS
NONAME000 -      normal    -      20/20    1/2
NONAME001 -      short     -      60/20    3/2
```

Limits for resource consumers

Resource allocation limits are applied according to the kind of resource consumer (host groups, compute units, users, user groups)

Host groups and compute units

If a limit is specified for a host group or compute unit, the total amount of a resource used by all hosts in that group or unit is counted. If a host is a member of more than one group, each job running on that host is counted against the limit for all groups to which the host belongs. Per-user limits are enforced on each user or individually to each user in the user group listed. If a user group contains a subgroup, the limit also applies to each member in the subgroup recursively.

Limits for users and user groups

Jobs are normally queued on a first-come, first-served (FCFS) basis. It is possible for some users to abuse the system by submitting a large number of jobs; jobs from other users must wait until these jobs complete. Limiting resources by user prevents users from monopolizing all the resources.

Users can submit an unlimited number of jobs, but if they have reached their limit for any resource, the rest of their jobs stay pending, until some of their running jobs finish or resources become available.

If a limit is specified for a user group, the total amount of a resource used by all users in that group is counted. If a user is a member of more than one group, each of that user's jobs is counted against the limit for all groups to which that user belongs.

Use the keyword `all` to configure limits that apply to each user or user group in a cluster. This is useful if you have a large cluster but only want to exclude a few users from the limit definition.

You can use `ENFORCE_ONE_UG_LIMITS=Y` combined with `bsub -G` to have better control over limits when user groups have overlapping members. When set to Y, only the specified user group's limits (or those of any parent user group) are enforced. If set to N, the most restrictive job limits of any overlapping user/user group are enforced.

Per-user limits on users and groups

Per-user limits that use the keywords `all` apply to each user in a cluster. If user groups are configured, the limit applies to each member of the user group, not the group as a whole.

Resizable jobs

When a resize allocation request is scheduled for a resizable job, all resource allocation limits (job and slot) are enforced.

Once the new allocation is satisfied, it consumes limits such as SLOTS, MEM, SWAP and TMP for queues, users, projects, hosts, or cluster-wide. However, the new allocation will not consume job limits such as job group limits, job array limits, and non-host level JOBS limit.

Releasing part of an allocation from a resizable job frees general limits that belong to the allocation, but not the actual job limits.

How job limits work

The JOBS parameter limits the maximum number of running or suspended jobs available to resource consumers. Limits are enforced depending on the number of jobs in RUN, SSUSP, and USUSP state.

Stop and resume jobs

Jobs stopped with `bstop`, go into USUSP status. LSF includes USUSP jobs in the count of running jobs, so the usage of JOBS limit will not change when you suspend a job.

Resuming a stopped job (`bresume`) changes job status to SSUSP. The job can enter RUN state, if the JOBS limit has not been exceeded. Lowering the JOBS limit before resuming the job can exceed the JOBS limit, and prevent SSUSP jobs from entering

RUN state.

For example, JOBS=5, and 5 jobs are running in the cluster (JOBS has reached 5/5). Normally, the stopped job (in USUSP state) can later be resumed and begin running, returning to RUN state. If you reconfigure the JOBS limit to 4 before resuming the job, the JOBS usage becomes 5/4, and the job cannot run because the JOBS limit has been exceeded.

Preemption

The JOBS limit does not block preemption based on job slots. For example, if JOBS=2, and a host is already running 2 jobs in a preemptable queue, a new preemptive job can preempt a job on that host as long as the preemptive slots can be satisfied even though the JOBS limit has been reached.

Reservation and backfill

Reservation and backfill are still made at the job slot level, but despite a slot reservation being satisfied, the job may ultimately not run because the JOBS limit has been reached.

Other jobs

- **brun** forces a pending job to run immediately on specified hosts. A job forced to run with **brun** is counted as a running job, which may violate JOBS limits. After the forced job starts, the JOBS limits may be exceeded.
- Re-queued jobs (**brequeue**) are assigned PEND status or PSUSP. Usage of JOBS limit is decreased by the number of re-queued jobs.
- Checkpointed jobs restarted with **brestart** start a new job based on the checkpoint of an existing job. Whether the new job can run depends on the limit policy (including the JOBS limit) that applies to the job. For example, if you checkpoint a job running on a host that has reached its JOBS limit, then restart it, the restarted job cannot run because the JOBS limit has been reached.
- For job arrays, you can define a maximum number of jobs that can run in the array at any given time. The JOBS limit, like other resource allocation limits, works in combination with the array limits. For example, if JOBS=3 and the array limit is 4, at most 3 job elements can run in the array.
- For chunk jobs, only the running job among the jobs that are dispatched together in a chunk is counted against the JOBS limit. Jobs in WAIT state do not affect the JOBS limit usage.

Example limit configurations

Each set of limits is defined in a Limit section enclosed by Begin Limit and End Limit.

Example 1

user1 is limited to 2 job slots on hostA, and user2's jobs on queue normal are limited to 20 MB of memory:

```
Begin Limit
NAME    HOSTS    SLOTS  MEM    SWP    TMP    USERS    QUEUES
Limit1  hostA      2      -      -      -      user1    -
-       -         -      20     -      -      user2    normal
End Limit
```

Example 2

Set a job slot limit of 2 for user user1 submitting jobs to queue normal on host hosta for all projects, but only one job slot for all queues and hosts for project test:

```
Begin Limit
HOSTS   SLOTS  PROJECTS  USERS    QUEUES
hosta   2      -         user1    normal
-       1      test      user1    -
End Limit
```

Example 3

All users in user group ugroup1 except user1 using queue1 and queue2 and running jobs on hosts in host group hgroup1 are limited to 2 job slots per processor on each host:


```

Begin Limit
NAME          = limit1
# Resources:
SLOTS_PER_PROCESSOR = 2
#Consumers:
QUEUES        = queue1 queue2
USERS         = ugroup1 ~user1
PER_HOST      = hgroup1
End Limit

```

Example 4

user1 and user2 can use all queues and all hosts in the cluster with a limit of 20 MB of available memory:

```

Begin Limit
NAME = 20_MB_mem
# Resources:
MEM = 20
# Consumers:
USERS = user1 user2
End Limit

```

Example 5

All users in user group ugroup1 can use queue1 and queue2 and run jobs on any host in host group hgroup1 sharing 10 job slots:

```

Begin Limit
NAME = 10_slot
# Resources:
SLOTS = 10
#Consumers:
QUEUES = queue1 queue2
USERS = ugroup1
HOSTS = hgroup1
End Limit

```

Example 6

All users in user group ugroup1 except user1 can use all queues but queue1 and run jobs with a limit of 10% of available memory on each host in host group hgroup1:

```

Begin Limit
NAME = 10_percent_mem
# Resources:
MEM = 10%
QUEUES = all ~queue1
USERS = ugroup1 ~user1
PER_HOST = hgroup1
End Limit

```

Example 7

Limit users in the develop group to 1 job on each host, and 50% of the memory on the host.

```

Begin Limit
NAME = develop_group_limit
# Resources:
SLOTS = 1
MEM = 50%
#Consumers:
USERS = develop
PER_HOST = all
End Limit

```

Example 8

Limit all hosts to 1 job slot per processor:

```

Begin Limit
NAME = default_limit

```

```

SLOTS_PER_PROCESSOR = 1
PER_HOST             = all
End Limit

```

Example 9

The short queue can have at most 200 running and suspended jobs:

```

Begin Limit
NAME      = shortq_limit
QUEUES    = short
JOBS      = 200
End Limit

```

Configuring resource allocation limits

Configure all resource allocation limits in one or more `Limit` sections in the `lsb.resources` file. Limit sections set limits for how much of the specified resources must be available for different classes of jobs to start, and which resource consumers the limits apply to.

lsb.resources file

You can also specify the duration for which the resource is reserved. When the duration expires, the resource is released, but the limitation is still enforced. This behavior applies for all type of resources, including built-in resources, static, and dynamic shared resources, LSF License Scheduler tokens. The resource requirements that are defined for queue level or job level are the same in this case.

Note: The `Limit` section of the `lsb.resources` file does not support the keywords or format that is used in the `lsb.users`, `lsb.hosts`, and `lsb.queues` files. However, any existing job slot limit configuration in these files continues to apply.

Resource parameters

Limit	Limit section parameter
Total number of running and suspended (RUN, SSUSP, USUSP) jobs.	JOBS
Total number of job slots that can be used by specific jobs.	SLOTS
Jobs slots based on the number of processors on each host that is affected by the limit.	SLOTS_PER_PROCESSOR and PER_HOST
Memory - if the PER_HOST parameter is set for the limit, the amount can be a percentage of memory on each host in the limit.	MEM (in MB or units set in the LSF_UNIT_FOR_LIMITS parameter in the lsf.conf file)
Swap space - if the PER_HOST parameter is set for the limit, the amount can be a percentage of swap space on each host in the limit.	SWP (in MB or units set in the LSF_UNIT_FOR_LIMITS parameter in the lsf.conf file)
Temp space - if the PER_HOST parameter is set for the limit, the amount can be a percentage of temp space on each host in the limit.	TMP (in MB or units set in the LSF_UNIT_FOR_LIMITS parameter in the lsf.conf file)
Any shared resource.	RESOURCE
Number of jobs that are dispatched per scheduling cycle - the USERS, PER_USER, QUEUES, or PER_QUEUE parameter must be set for the limit.	JOB_DISPATCH_LIMIT

Note: By default, the `tmp` resource is not supported by the `LSF_UNIT_FOR_LIMITS` parameter. Use the parameter `LSF_ENABLE_TMP_UNIT=Y` to enable the `LSF_UNIT_FOR_LIMITS` parameter to support limits on the `tmp` resource.

Consumer parameters

Submitted jobs	Limit section parameter
----------------	-------------------------

Submitted jobs	Limit section parameter
By all specified users or user groups.	USERS
To all specified queues.	QUEUES
To all specified hosts, host groups, or compute units.	HOSTS
For all specified projects.	PROJECTS
By each specified user or each member of the specified user groups.	PER_USER
To each specified queue.	PER_QUEUE
To each specified host or each member of specified host groups or compute units.	PER_HOST
For each specified project.	PER_PROJECT

- [Enabling resource allocation limits](#)
- [Configuring cluster-wide limits](#)
- [Limit conflicts](#)
- [How resource allocation limits map to pre-version 7 job slot limits](#)

Enabling resource allocation limits

Procedure

To enable resource allocation limits in your cluster, you configure the resource allocation limits scheduling plugin `schmod_limit` in `lsb.modules`:

```
Begin PluginModule
SCH_PLUGIN          RB_PLUGIN          SCH_DISABLE PHASES
schmod_default      ()                ()
schmod_limit        ()                ()
End PluginModule
```

Configuring cluster-wide limits

Procedure

To configure limits that take effect for your entire cluster, configure limits in `lsb.resources`, but do not specify any consumers.

Limit conflicts

Similar conflicting limits

For similar limits configured in `lsb.resources`, `lsb.users`, `lsb.hosts`, or `lsb.queues`, the most restrictive limit is used. For example, a slot limit of 3 for all users is configured in `lsb.resources`:

```
Begin Limit
NAME = user_limit1
USERS = all
SLOTS = 3
End Limit
```

This is similar, but *not equivalent* to an existing `MAX_JOBS` limit of 2 is configured in `lsb.users`.

```
busers
USER/GROUP  JL/P  MAX  NJOBS  PEND  RUN  SSUSP  USUSP  RSV
user1      -      2    4      2     2     0      0      0
```

user1 submits 4 jobs:

bjobs

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
816	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16:34
817	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16:34
818	user1	PEND	normal	hostA		sleep 1000	Jan 22 16:34
819	user1	PEND	normal	hostA		sleep 1000	Jan 22 16:34

Two jobs (818 and 819) remain pending because the more restrictive limit of 2 from lsb.users is enforced:

bjobs -p

JOBID	USER	STAT	QUEUE	FROM_HOST	JOB_NAME	SUBMIT_TIME
818	user1	PEND	normal	hostA	sleep 1000	Jan 22 16:34
The user has reached his/her job slot limit;						
819	user1	PEND	normal	hostA	sleep 1000	Jan 22 16:34
The user has reached his/her job slot limit;						

If the MAX_JOBS limit in lsb.users is 4:

busers

USER/GROUP	JL/P	MAX	NJOBS	PEND	RUN	SSUSP	USUSP	RSV
user1	-	4	4	1	3	0	0	0

and user1 submits 4 jobs:

bjobs

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
824	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16:38
825	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16:38
826	user1	RUN	normal	hostA	hostA	sleep 1000	Jan 22 16:38
827	user1	PEND	normal	hostA		sleep 1000	Jan 22 16:38

Only one job (827) remains pending because the more restrictive limit of 3 in lsb.resources is enforced:

bjobs -p

JOBID	USER	STAT	QUEUE	FROM_HOST	JOB_NAME	SUBMIT_TIME
827	user1	PEND	normal	hostA	sleep 1000	Jan 22 16:38
Resource (slot) limit defined cluster-wide has been reached;						

Equivalent conflicting limits

New limits in lsb.resources that are equivalent to existing limits in lsb.users, lsb.hosts, or lsb.queues, but with a different value override the existing limits. The equivalent limits in lsb.users, lsb.hosts, or lsb.queues are ignored, and the value of the new limit in lsb.resources is used.

For example, a *per-user* job slot limit in lsb.resources is equivalent to a MAX_JOBS limit in lsb.users, so only the lsb.resources limit is enforced, the limit in lsb.users is ignored:

Begin Limit

NAME = slot_limit

PER_USER =all

SLOTS = 3

End Limit

How resource allocation limits map to pre-version 7 job slot limits

Job slot limits are the only type of limit you can configure in lsb.users, lsb.hosts, and lsb.queues. You cannot configure limits for user groups, host groups, and projects in lsb.users, lsb.hosts, and lsb.queues. You should not configure any new resource allocation limits in lsb.users, lsb.hosts, and lsb.queues. Use lsb.resources to configure all new resource allocation limits, including job slot limits.

Job slot resources	Resource consumers (lsb.resources)					Equivalent existing limit (file)
(lsb.resources)	USERS	PER_USER	QUEUES	HOSTS	PER_HOST	
SLOTS	—	all	—	host_name	—	JL/U (lsb.hosts)
SLOTS_PER_PROCESSOR	user_name	—	—	—	all	JL/P (lsb.users)
SLOTS	—	all	queue_name	—	—	UJOB_LIMIT (lsb.queues)
SLOTS	—	all	—	—	—	MAX_JOBS (lsb.users)
SLOTS	—	—	queue_name	—	all	HJOB_LIMIT (lsb.queues)
SLOTS	—	—	—	host_name	—	MXJ (lsb.hosts)
SLOTS_PER_PROCESSOR	—	—	queue_name	—	all	PJOB_LIMIT (lsb.queues)
SLOTS	—	—	queue_name	—	—	QJOB_LIMIT (lsb.queues)

Limits for the following resources have no corresponding limit in lsb.users, lsb.hosts, and lsb.queues:

- JOBS
- RESOURCE
- SWP
- TMP

Creating a limit using bconf

About this task

You can create new limits using live reconfiguration.

Procedure

Run `bconf create limit=limit_name`

For example, to create the limit X1 with a job limit of 23 per host:

```
bconf create limit=X1 "JOBS=23;PER_HOST=host12"
bconf: Request for limit <X1> accepted
```

Once accepted by **bconf**, the new limit appears in **blimits** output:

```
blimits -cn X1
Begin Limit
  NAME      = X1
  PER_HOST  = host12
  JOBS      = 23
End Limit
```

Results

Limits that are created using **bconf create** are written to the changed lsb.resources configuration file in horizontal format.

- [Updating a limit using bconf](#)

Updating a limit using bconf

Procedure

Run **bconf update limit=limit_name**. For example:

```
bconf update limit=Lim3 "JOBS=20; SLOTS=100"
```

Examples of changing a limit in two steps

Changing a limit using bconf might require two **bconf** calls if you have a dependent value or want to change from an integer to a percentage setting.

For example, given the limit L1 configured in lsb.resources, MEM is dependent on PER_HOST:

```
Begin Limit
NAME          = L1
PER_USER      = all
PER_QUEUE     = normal priority
PER_HOST      = all
MEM           = 40%
End Limit
```

One **bconf update** call cannot reset both the PER_HOST value and dependent MEM percentage value:

```
bconf update limit=L1 "MEM=-;PER_HOST=-"
bconf: Request for limit <L1> rejected
Error(s): PER_HOST cannot be replaced due to the dependent resource MEM
```

Instead, reset MEM and PER_HOST in two steps:

```
bconf update limit=L1 "MEM=-;"
bconf: Request for limit <L1> accepted
bconf update limit=L1 "PER_HOST=-"
bconf: Request for limit <L1> accepted
```

Similarly, changing the value of SWP from a percentage to an integer requires two steps:

```
Begin Limit
NAME          = L1
...
SWP = 40%
End Limit

bconf update limit=L1 "SWP=20"
bconf: Request for limit <L1> rejected
Error(s): Cannot change between integer and percentage directly; reset the resource first
```

First reset SWP and then set as an integer, calling **bconf** twice:

```
bconf update limit=L1 "SWP=-;"
bconf: Request for limit <L1> accepted
bconf update limit=L1 "SWP=20"
bconf: Request for limit <L1> accepted
```

Make sure LSF resources are distributed fairly

Use runtime resource usage limits to control how much resource can be consumed by running jobs. Configure load thresholds so that Jobs running under LSF can be automatically suspended and resumed based on the load conditions on the execution hosts. Learn about dispatch and run windows and deadline constraint scheduling.

- [Runtime resource usage limits](#)

- [Load thresholds](#)
- [Time configuration](#)

Learn about dispatch and run windows and deadline constraint scheduling.

Runtime resource usage limits

- [About resource usage limits](#)
Resource usage limits control how much resource can be consumed by running jobs. Jobs that use more than the specified amount of a resource are signaled or have their priority lowered.
- [Changing the units for resource usage limits](#)
Use the LSF_UNIT_FOR_LIMITS parameter in the lsf.conf file to specify larger units for resource usage limits.
- [Specifying resource usage limits](#)
- [Supported resource usage limits and syntax](#)
Set runtime resource usage limits in the lsb.queues file or with the **bsub** command. Each limit has a default value and a specific format.
- [Linmit examples](#)
- [CPU time and run time normalization](#)
To set the CPU time limit and run time limit for jobs in a platform-independent way, LSF scales the limits by the CPU factor of the hosts involved. When a job is dispatched to a host for execution, the limits are then normalized according to the CPU factor of the execution host.
- [Memory and swap limit enforcement based on Linux cgroup memory subsystem](#)
All LSF job processes are controlled by the Linux cgroup system so that cgroup memory and swap limits cannot be exceeded. These limits are enforced on a per job and per host basis, not per task, on Red Hat Enterprise Linux (RHEL) 6.2 or above and SuSe Linux Enterprise Linux 11 SP2 or above. LSF enforces memory and swap limits for jobs by periodically collecting job usage and comparing it with limits set by users. LSF can impose strict host-level memory and swap limits on systems that support Linux **cgroup v1** or **cgroup v2** cgroups.
- [PAM resource limits](#)
PAM limits are system resource limits defined in limits.conf.

About resource usage limits

Resource usage limits control how much resource can be consumed by running jobs. Jobs that use more than the specified amount of a resource are signaled or have their priority lowered.

Limits can be specified by the LSF administrator:

- At the queue level in lsb.queues
- In an application profile in lsb.applications
- At the job level when you submit a job

For example, by defining a high-priority short queue, you can allow short jobs to be scheduled earlier than long jobs. To prevent some users from submitting long jobs to this short queue, you can set CPU limit for the queue so that no jobs submitted from the queue can run for longer than that limit.

Limits specified at the queue level are *hard* limits, while those specified with job submission or in an application profile are *soft* limits. The hard limit acts as a ceiling for the soft limit. See `setrlimit(2)` man page for concepts of hard and soft limits.

Note: This chapter describes queue-level and job-level resource usage limits. Priority of limits is different if limits are also configured in an application profile.

Resource usage limits and resource allocation limits

Resource usage limits are not the same as *resource allocation limits*, which are enforced during job scheduling and before jobs are dispatched. You set resource allocation limits to restrict the amount of a given resource that must be available during job

scheduling for different classes of jobs to start, and to which resource consumers the limits apply.

Resource usage limits and resource reservation limits

Resource usage limits are not the same as queue-based *resource reservation limits*, which are enforced during job submission. The parameter RESRSV_LIMIT (in lsb.queues) specifies allowed ranges of resource values, and jobs submitted with resource requests outside of this range are rejected.

Summary of resource usage limits

Limit	Job syntax (bsub)	Syntax (lsb.queues and lsb.applications)	Format/Default Units
Core file size limit	-C <i>core_limit</i>	CORELIMIT= <i>limit</i>	<i>integer</i> KB
CPU time limit	-c <i>cpu_limit</i>	CPULIMIT=[<i>default</i>] <i>maximum</i>	[<i>hours</i> :] <i>minutes</i> [/ <i>host_name</i> / <i>host_model</i>]
Data segment size limit	-D <i>data_limit</i>	DATALIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB
File size limit	-F <i>file_limit</i>	FILELIMIT= <i>limit</i>	<i>integer</i> KB
Memory limit	-M <i>mem_limit</i>	MEMLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB
Process limit	-p <i>process_limit</i>	PROCESSLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>
Run time limit	-W <i>run_limit</i>	RUNLIMIT=[<i>default</i>] <i>maximum</i>	[<i>hours</i> :] <i>minutes</i> [/ <i>host_name</i> / <i>host_model</i>]
Stack segment size limit	-S <i>stack_limit</i>	STACKLIMIT= <i>limit</i>	<i>integer</i> KB
Virtual memory limit	-v <i>swap_limit</i>	SWAPLIMIT= <i>limit</i>	<i>integer</i> KB
Thread limit	-T <i>thread_limit</i>	THREADLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>

Priority of resource usage limits

If no limit is specified at job submission, then the following apply to all jobs submitted to the queue:

If ...	Then ...
Both default and maximum limits are defined	The default is enforced
Only a maximum is defined	The maximum is enforced
No limit is specified in the queue or at job submission	No limits are enforced

Incorrect resource usage limits

Incorrect limits are ignored, and a warning message is displayed when the cluster is reconfigured or restarted. A warning message is also logged to the mbatchd log file when LSF is started.

If no limit is specified at job submission, then the following apply to all jobs submitted to the queue:

If ...	Then ...
The default limit is not correct	The default is ignored and the maximum limit is enforced
Both default and maximum limits are specified, and the maximum is not correct	The maximum is ignored and the resource has no maximum limit, only a default limit
Both default and maximum limits are not correct	The default and maximum are ignored and no limit is enforced

Resource usage limits specified at job submission must be less than the maximum specified in lsb.queues. The job submission is rejected if the user-specified limit is greater than the queue-level maximum, and the following message is issued:

```
Cannot exceed queue's hard limit(s). Job not submitted.
```


Changing the units for resource usage limits

Use the LSF_UNIT_FOR_LIMITS parameter in the lsf.conf file to specify larger units for resource usage limits.

The default unit for the following resource usage limits is KB:

- Core limit (**-C** and CORELIMIT)
- Memory limit (**-M** and MEMLIMIT)
- Stack limit (**-S** and STACKLIMIT)
- Swap limit (**-v** and SWAPLIMIT)

This default may be too small for environments that make use of very large resource usage limits, for example, GB, TB, or larger.

The unit for the resource usage limit can be one of:

- KB or K (kilobytes)
- MB or M (megabytes)
- GB or G (gigabytes)
- TB or T (terabytes)
- PB or P (petabytes)
- EB or E (exabytes)
- ZB or Z (zettabytes)

The LSF_UNIT_FOR_LIMITS parameter in the lsf.conf file applies cluster-wide to limits at the job-level (**bsub** command), queue-level (lsb.queues file), and application level (lsb.applications file).

The limit unit specified by the LSF_UNIT_FOR_LIMITS parameter also applies to limits modified with **bmod**, and the display of resource usage limits in query commands (**bacct**, **bapp**, **bhist**, **bhosts**, **bjobs**, **bqueues**, **lsload**, and **lshosts**).

By default, the `tmp` resource is not supported by the LSF_UNIT_FOR_LIMITS parameter. Use the parameter LSF_ENABLE_TMP_UNIT=Y to enable the LSF_UNIT_FOR_LIMITS parameter to support limits on the `tmp` resource.

Important: Before you change the units of your resource usage limits, completely drain the cluster of all workload, so that no running, pending, or finished jobs are in the system.

In the LSF multicluster capability environment, configure the same unit for all clusters.

After you change the LSF_UNIT_FOR_LIMITS parameter, you must restart your cluster.

How changing the limit unit affects command options and output

When the LSF_UNIT_FOR_LIMITS parameter is specified, the defined unit is used for the following commands. In command output, the larger unit appears as T, G, P, E, or Z, depending on the job `rusage` and the unit defined.

Command	Option or output	Default unit
bsub/bmod	-C (core limit)	KB
	-M (memory limit)	KB
	-S (stack limit)	KB
	-v (swap limit)	KB
bjobs	rusage CORELIMIT, MEMLIMIT, STACKLIMIT, SWAPLIMIT	KB (might show MB depending on job <code>rusage</code>)
bqueues	CORELIMIT, MEMLIMIT, STACKLIMIT, SWAPLIMIT	KB (might show MB depending on job <code>rusage</code>)
	loadSched, loadStop	MB
bacct	Summary <code>rusage</code>	KB (might show MB depending on job <code>rusage</code>)
bapp	CORELIMIT, MEMLIMIT, STACKLIMIT, SWAPLIMIT	KB
bhist	History of limit change by bmod	KB
	MEM, SWAP	KB (might show MB depending on job <code>rusage</code>)
bhosts	loadSched, loadStop	MB

Command	Option or output	Default unit
lsload	mem, swp	KB (might show MB depending on job <i>rusage</i>)
lshosts	maxmem, maxswp	KB (might show MB depending on job <i>rusage</i>)

Example

A job is submitted with **bsub -M 100** and the LSF_UNIT_FOR_LIMITS=GB parameter is set. The memory limit for the job is 100 GB rather than the default 100 MB.

Specifying resource usage limits

About this task

Queues can enforce resource usage limits on running jobs. LSF supports most of the limits that the underlying operating system supports. In addition, LSF also supports a few limits that the underlying operating system does not support.

Procedure

Specify queue-level resource usage limits using parameters in `lsb.queues`.

- [Default run limits for backfill scheduling](#)
- [Specifying job-level resource usage limits](#)

Specify queue-level resource usage limits

About this task

Limits configured in `lsb.queues` apply to all jobs submitted to the queue. Job-level resource usage limits specified at job submission override the queue definitions.

Procedure

Specify only a maximum value for the resource.

For example, to specify a maximum run limit, use one value for the RUNLIMIT parameter in `lsb.queues`:

```
RUNLIMIT = 10
```

The maximum run limit for the queue is 10 minutes. Jobs cannot run for more than 10 minutes. Jobs in the RUN state for longer than 10 minutes are killed by LSF.

If only one run limit is specified, jobs that are submitted with **bsub -W** with a run limit that exceeds the maximum run limit is not allowed to run. Jobs submitted without **bsub -W** are allowed to run but are killed when they are in the RUN state for longer than the specified maximum run limit.

For example, in `lsb.queues`:

```
RUNLIMIT = 10
```

Default and maximum values

If you specify two limits, the first one is the default limit for jobs in the queue and the second one is the maximum (hard) limit. Both the default and the maximum limits must be positive integers. The default limit must be less than the maximum limit. The default limit is ignored if it is greater than the maximum limit.

Use the default limit to avoid having to specify resource usage limits in the **bsub** command.

For example, to specify a default and a maximum run limit, use two values for the RUNLIMIT parameter in `lsb.queues`:

RUNLIMIT = 10 15

- The first number is the default run limit applied to all jobs in the queue that are submitted without a job-specific run limit (without **bsub -W**).
- The second number is the maximum run limit applied to all jobs in the queue that are submitted with a job-specific run limit (with **bsub -W**). The default run limit must be less than the maximum run limit.

You can specify both default and maximum values for the following resource usage limits in `lsb.queues`:

- CPULIMIT
- DATALIMIT
- MEMLIMIT
- PROCESSLIMIT
- RUNLIMIT
- THREADLIMIT

Host specification with two limits

If default and maximum limits are specified for CPU time limits or run time limits, only one host specification is permitted. For example, the following CPU limits are correct (and have an identical effect):

```
CPULIMIT = 400/hostA 600
CPULIMIT = 400 600/hostA
```

The following CPU limit is not correct:

```
CPULIMIT = 400/hostA 600/hostB
```

The following run limits are correct (and have an identical effect):

```
RUNLIMIT = 10/hostA 15
RUNLIMIT = 10 15/hostA
```

The following run limit is not correct:

```
RUNLIMIT = 10/hostA 15/hostB
```

Default run limits for backfill scheduling

Default run limits are used for backfill scheduling of parallel jobs.

For example, in `lsb.queues`, you enter: **RUNLIMIT = 10 15**

- The first number is the default run limit applied to all jobs in the queue that are submitted without a job-specific run limit (without **bsub -W**).
- The second number is the maximum run limit applied to all jobs in the queue that are submitted with a job-specific run limit (with **bsub -W**). The default run limit cannot exceed the maximum run limit.

Automatically assigning a default run limit to all jobs in the queue means that backfill scheduling works efficiently.

If you submit a job to the queue with the **-W** option, (**bsub-W 12 myjob**) the maximum run limit is used. The job `myjob` is allowed to run on the queue because the specified run limit (12) is less than the maximum run limit for the queue (15).

However, if the specified run limit is greater than the run limit for the queue (15) (for example, **bsub-W 20 myjob**), then the job will be rejected from the queue.

Specifying job-level resource usage limits

Procedure

To specify resource usage limits at the job level, use one of the following **bsub** options:

- **-C *core_limit***
- **-c *cpu_limit***
- **-D *data_limit***
- **-F *file_limit***
- **-M *mem_limit***
- **-p *process_limit***
- **-W *run_limit***
- **-S *stack_limit***
- **-T *thread_limit***
- **-v *swap_limit***

Job-level resource usage limits specified at job submission override the queue definitions.

Supported resource usage limits and syntax

Set runtime resource usage limits in the `lsb.queues` file or with the **bsub** command. Each limit has a default value and a specific format.

- **[CPU time limit](#)**
Sets the soft CPU time limit to *cpu_limit* for this batch job. The default is no limit.
- **[Data segment size limit](#)**
Sets a per-process (soft) data segment size limit in KB for each process that belongs to this batch job (see **getrlimit**).
- **[File size limit](#)**
Sets a file size limit in KB for each process that belongs to this batch job.
- **[Memory limit](#)**
Sets a per-process physical memory limit for all of the processes belonging to a job.
- **[Process limit](#)**
Sets the limit of the number of processes to *process_limit* for the whole job. The default is no limit. Exceeding the limit causes the job to terminate.
- **[Runtime limit](#)**
A *runtime limit* is the maximum amount of time a job can run before it is terminated.
- **[Thread limit](#)**
Sets the limit of the number of concurrent threads for the whole job. The default is no limit.
- **[Stack segment size limit](#)**
Sets a per-process (hard) stack segment size limit for all of the processes belonging to a job. Application-level and job-level stack segment size limits overwrite this value as the soft limit, but cannot exceed the hard limit set in the `lsb.queues` file.
- **[Swap usage limit](#)**
Sets a total process limit (*memory* and *swap* usage) for the whole job. The default is no limit. Exceeding the limit causes the job to terminate.

CPU time limit

Sets the soft CPU time limit to *cpu_limit* for this batch job. The default is no limit.

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
<code>-c <i>cpu_limit</i></code>	<code>CPULIMIT=[<i>default</i>] <i>maximum</i></code>	<code>[<i>hours</i>:]<i>minutes</i>[/<i>host_name</i> /<i>host_model</i>]</code>

This option is useful for avoiding runaway jobs that use up too many resources. LSF keeps track of the CPU time used by all processes of the job.

When the job accumulates the specified amount of CPU time, a `SIGXCPU` signal is sent to all processes belonging to the job. If the job has no signal handler for `SIGXCPU`, the job is killed immediately. If the `SIGXCPU` signal is handled, blocked, or ignored

by the application, then after the grace period expires, LSF sends `SIGINT`, `SIGTERM`, and `SIGKILL` signals to the job to kill it.

You can define whether the CPU limit is a per-process limit enforced by the OS or a per-job limit enforced by LSF with the `LSB_JOB_CPULIMIT` parameter in the `lsf.conf` file.

Jobs submitted to a chunk job queue are not chunked if the CPU limit is greater than 30 minutes.

`cpu_limit` is in the form `[hour:]minute`, where *minute* can be greater than 59. 3.5 hours can either be specified as 3:30 or 210.

- **Normalized CPU time**

The CPU time limit is *normalized* according to the CPU factor of the submission host and execution host. The CPU limit is scaled so that the job does approximately the same amount of processing for a given CPU limit, even if it is sent to a host with a faster or slower CPU.

Normalized CPU time

The CPU time limit is *normalized* according to the CPU factor of the submission host and execution host. The CPU limit is scaled so that the job does approximately the same amount of processing for a given CPU limit, even if it is sent to a host with a faster or slower CPU.

For example, if a job is submitted from a host with a CPU factor of 2 and executed on a host with a CPU factor of 3, the CPU time limit is multiplied by 2/3 because the execution host can do the same amount of work as the submission host in 2/3 of the time.

If the optional host name or host model is not given, the CPU limit is scaled based on the `DEFAULT_HOST_SPEC` value specified in the `lsb.params` file. If the `DEFAULT_HOST_SPEC` parameter is not defined, the fastest batch host in the cluster is used as the default. If host or host model is given, its CPU scaling factor is used to adjust the actual CPU time limit at the execution host.

The following example specifies that `myjob` can run for 10 minutes on a DEC3000 host, or the corresponding time on any other host:

```
bsub -c 10/DEC3000 myjob
```

Data segment size limit

Sets a per-process (soft) data segment size limit in KB for each process that belongs to this batch job (see **getrlimit**).

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
<code>-D data_limit</code>	<code>DATALIMIT=[default] maximum</code>	<i>integer</i> KB

This option affects calls to `sbrk()` and `brk()`. An `sbrk()` or `malloc()` call to extend the data segment beyond the data limit returns an error.

Note: Linux does not use `sbrk()` and `brk()` within its `calloc()` and `malloc()`. Instead, it uses `(mmap())` to create memory. The `DATALIMIT` value cannot be enforced on Linux applications that call `sbrk()` and `malloc()`.

On AIX, if the `XPG_SUS_ENV=ON` environment variable is set in the user's environment before the process is executed and a process attempts to set the limit lower than current usage, the operation fails with **errno** set to `EINVAL`. If the `XPG_SUS_ENV` environment variable is not set, the operation fails with **errno** set to `EFAULT`.

The default is no soft limit.

File size limit

Sets a file size limit in KB for each process that belongs to this batch job.

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-F <i>file_limit</i>	FILELIMIT= <i>limit</i>	<i>integer</i> KB

If a per-process (soft) process of this job attempts to write to a file such that the file size would increase beyond the file limit, the kernel sends that process a `SIGXFSZ` signal. This condition normally terminates the process, but may be caught. The default is no soft limit.

Memory limit

Sets a per-process physical memory limit for all of the processes belonging to a job.

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-M <i>mem_limit</i>	MEMLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i> KB

By default, the limit is specified in KB. Use the `LSF_UNIT_FOR_LIMITS` parameter in the `lsf.conf` file to specify a larger unit for the limit (MB, GB, TB, PB, or EB).

If the `LSB_MEMLIMIT_ENFORCE=Y` or the `LSB_JOB_MEMLIMIT=Y` parameter is set in the `lsf.conf` file, LSF kills the job when it exceeds the memory limit. Otherwise, LSF passes the memory limit to the operating system. Some operating systems apply the memory limit to each process, and some do not enforce the memory limit at all.

- [LSF memory limit enforcement](#)
To enable LSF memory limit enforcement, set the `LSB_MEMLIMIT_ENFORCE` parameter in the `lsf.conf` file to `y`.
- [Smart memory limit enforcement](#)
The parameter `LSB_MEMLIMIT_ENF_CONTROL` in the `lsf.conf` file further refines the behavior of enforcing a job memory limit.
- [OS memory limit enforcement](#)
OS enforcement usually allows the process to eventually run to completion. LSF passes *mem_limit* to the OS, which uses it as a guide for the system scheduler and memory allocator.

LSF memory limit enforcement

To enable LSF memory limit enforcement, set the `LSB_MEMLIMIT_ENFORCE` parameter in the `lsf.conf` file to `y`.

LSF memory limit enforcement explicitly sends a signal to kill a running process once it has allocated memory past *mem_limit*.

You can also enable LSF memory limit enforcement by setting the `LSB_JOB_MEMLIMIT` parameter in the `lsf.conf` file to `y`. The difference between `LSB_JOB_MEMLIMIT` set to `y` and `LSB_MEMLIMIT_ENFORCE` set to `y` is that with `LSB_JOB_MEMLIMIT`, only the per-job memory limit enforced by LSF is enabled. The per-process memory limit enforced by the OS is disabled. With the `LSB_MEMLIMIT_ENFORCE` parameter set to `y`, both the per-job memory limit enforced by LSF and the per-process memory limit enforced by the OS are enabled.

The `LSB_JOB_MEMLIMIT` parameter disables the per-process memory limit enforced by the OS and enables per-job memory limit enforced by LSF. When the total memory allocated to all processes in the job exceeds the memory limit, LSF sends the following signals to kill the job: `SIGINT` first, then `SIGTERM`, then `SIGKILL`.

On UNIX, the time interval between `SIGINT`, `SIGKILL`, `SIGTERM` can be configured with the parameter `JOB_TERMINATE_INTERVAL` in the `lsb.params` file.

Smart memory limit enforcement

The parameter `LSB_MEMLIMIT_ENF_CONTROL` in the `lsf.conf` file further refines the behavior of enforcing a job memory limit.

In the case that one or more jobs reach a specified memory limit (that is, both the host memory and swap utilization has reached a configurable threshold) at execution time, the worst offending job will be killed. A job is selected as the worst offending job on that host if it has the most overuse of memory (actual memory usage minus memory limit of the job).

You also have the choice of killing all jobs exceeding the thresholds (not just the worst).

OS memory limit enforcement

OS enforcement usually allows the process to eventually run to completion. LSF passes *mem_limit* to the OS, which uses it as a guide for the system scheduler and memory allocator.

The system might allocate more memory to a process if there is a surplus. When memory is low, the system takes memory from and lowers the scheduling priority (re-nice) of a process that has exceeded its declared *mem_limit*.

OS memory limit enforcement is only available on systems that support `RLIMIT_RSS` for the `setrlimit()` function.

The following operating systems do not support the memory limit at the OS level:

- Microsoft Windows
- Sun Solaris 2.x

Process limit

Sets the limit of the number of processes to *process_limit* for the whole job. The default is no limit. Exceeding the limit causes the job to terminate.

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-p <i>process_limit</i>	PROCESSLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>

Limits the number of concurrent processes that can be part of a job.

If a default process limit is specified, jobs submitted to the queue without a job-level process limit are killed when the default process limit is reached.

If you specify only one limit, it is the maximum, or hard, process limit. If you specify two limits, the first one is the default, or soft, process limit, and the second one is the maximum process limit.

Runtime limit

A *runtime limit* is the maximum amount of time a job can run before it is terminated.

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-W <i>run_limit</i>	RUNLIMIT=[<i>default</i>] <i>maximum</i>	[<i>hours:</i>] <i>minutes</i> [/ <i>host_name</i> / <i>host_model</i>]

It sets the run time limit of a job. The default is no limit. If the accumulated time the job has spent in the RUN state exceeds this limit, the job is sent a `USR2` signal. If the job does not terminate within 10 minutes after being sent this signal, it is killed.

With deadline constraint scheduling configured, a run limit also specifies the amount of time a job is expected to take, and the minimum amount of time that must be available before a job can be started.

run_limit is in the form [*hour:*]*minute*, where *minute* can be greater than 59. 3.5 hours can either be specified as 3:30 or 210.

Jobs submitted to a chunk job queue are not chunked if the run limit is greater than 30 minutes.

- **Normalized run time**

The runtime limit is *normalized* according to the CPU factor of the submission host and execution host. The run limit is scaled so that the job has approximately the same run time for a given run limit, even if it is sent to a host with a faster or slower CPU.

- **Runtime limits with the LSF multicluster capability**

The runtime limit is *normalized* according to the CPU factor of the submission host and execution host. The runtime limit is scaled so that the job has approximately the same run time for a given run limit, even if it is sent to a host with a faster or slower CPU.

Normalized run time

The runtime limit is *normalized* according to the CPU factor of the submission host and execution host. The run limit is scaled so that the job has approximately the same run time for a given run limit, even if it is sent to a host with a faster or slower CPU.

For example, if a job is submitted from a host with a CPU factor of 2 and executed on a host with a CPU factor of 3, the run limit is multiplied by 2/3 because the execution host can do the same amount of work as the submission host in 2/3 of the time.

If the optional host name or host model is not given, the run limit is scaled based on the DEFAULT_HOST_SPEC parameter specified in the lsb.params file. (If the DEFAULT_HOST_SPEC parameter is not defined, the fastest batch host in the cluster is used as the default.) If host or host model is given, its CPU scaling factor is used to adjust the actual run limit at the execution host.

The following example specifies that `myjob` can run for 10 minutes on a DEC3000 host, or the corresponding time on any other host:

```
bsub -W 10/DEC3000 myjob
```

If ABS_RUNLIMIT=Y is defined in the lsb.params file, the runtime limit is *not* normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run limit.

Runtime limits with the LSF multicluster capability

The runtime limit is *normalized* according to the CPU factor of the submission host and execution host. The runtime limit is scaled so that the job has approximately the same run time for a given run limit, even if it is sent to a host with a faster or slower CPU.

For LSF multicluster capability jobs, if no other CPU time normalization host is defined and information about the submission host is not available, LSF uses the host with the largest CPU factor (the fastest host in the cluster). The ABS_RUNLIMIT parameter in the lsb.params file is not supported in either LSF multicluster capability model; run time limit is normalized by the CPU factor of the execution host.

Thread limit

Sets the limit of the number of concurrent threads for the whole job. The default is no limit.

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
-T <i>thread_limit</i>	THREADLIMIT=[<i>default</i>] <i>maximum</i>	<i>integer</i>

Exceeding the limit causes the job to terminate. The system sends the following signals in sequence to all processes belongs to the job: SIGINT, SIGTERM, and SIGKILL.

If a default thread limit is specified, jobs submitted to the queue without a job-level thread limit are killed when the default thread limit is reached.

If you specify only one limit, it is the maximum, or hard, thread limit. If you specify two limits, the first one is the default, or soft, thread limit, and the second one is the maximum thread limit.

Stack segment size limit

Sets a per-process (hard) stack segment size limit for all of the processes belonging to a job. Application-level and job-level stack segment size limits overwrite this value as the soft limit, but cannot exceed the hard limit set in the `lsb.queues` file.

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
<code>-S stack_limit</code>	<code>STACKLIMIT=limit</code>	<i>integer</i> KB

By default, the limit is specified in KB. Use the `LSF_UNIT_FOR_LIMITS` parameter in the `lsf.conf` file to specify a larger unit for the limit (MB, GB, TB, PB, or EB).

An `sbrk()` call to extend the stack segment beyond the stack limit causes the process to be terminated. The default is no limit.

Swap usage limit

Sets a total process limit (*memory* and *swap* usage) for the whole job. The default is no limit. Exceeding the limit causes the job to terminate.

Job syntax (bsub)	Queue syntax (lsb.queues)	Format/Default Units
<code>-v swap_limit</code>	<code>SWAPLIMIT=limit</code>	<i>integer</i> KB

By default, the limit is specified in KB. Use the `LSF_UNIT_FOR_LIMITS` parameter in the `lsf.conf` file to specify a larger unit for the limit (MB, GB, TB, PB, or EB).

This limit applies to the whole job, no matter how many processes the job may contain.

By default, LSF collects both memory and swap usage through PIM:

- If the `EGO_PIM_SWAP_REPORT=n` parameter is set in the `lsf.conf` file (this is the default), swap usage is virtual memory (VSZ) of the entire job process.
- If the `EGO_PIM_SWAP_REPORT=y` parameter is set in the `lsf.conf` file, the resident set size (RSS) is subtracted from the virtual memory usage. RSS is the portion of memory occupied by a process that is held in main memory. Swap usage is collected as the $VSZ - RSS$.

For example, if a job process allocates memory as follows,

```
malloc(p, 200MB);  
memset(p, 150MB);
```

then `bjobs -l` shows SWAP as either 200 MB if `EGO_PIM_SWAP_REPORT=n`, or 50 MB if `EGO_PIM_SWAP_REPORT=y`. From `ps` or `top` output, 200 MB is the virtual memory (VSZ) and 150 MB is the resident set size (RSS).

On Linux, if you check the output of Linux utility **free** or monitor the output of **vmstat**, there is actually no swap usage at all. In fact, the meaning of swap usage in Linux is the physical space usage of the underlining swap device, which is usually a block device.

If memory enforcement through the Linux cgroup memory subsystem is enabled with the `LSF_LINUX_CGROUP_ACCT=y` parameter in the `lsf.conf` file, LSF uses the cgroup memory subsystem to collect memory and swap usage of all processes in a job.

For example, if you enable the `LSF_LINUX_CGROUP_ACCT=y` parameter, and run the same program,

```
malloc(p, 200MB);  
memset(p, 150MB);
```

The **bjobs -l** command shows SWAP as the swap usage of all currently running processes in a job collected by the cgroup memory subsystem.

Linmit examples

Queue-level limits

```
CPULIMIT = 20/hostA 15
```

The first number is the default CPU limit. The second number is the maximum CPU limit.

However, the default CPU limit is ignored because it is a higher value than the maximum CPU limit.

```
CPULIMIT = 10/hostA
```

In this example, the lack of a second number specifies that there is no default CPU limit. The specified number is considered as the default and maximum CPU limit.

```
RUNLIMIT = 10/hostA 15
```

The first number is the default run limit. The second number is the maximum run limit.

The first number specifies that the default run limit is to be used for jobs that are submitted without a specified run limit (without the **-W** option of **bsub**).

```
RUNLIMIT = 10/hostA
```

No default run limit is specified. The specified number is considered as the default and maximum run limit.

```
THREADLIMIT=6
```

No default thread limit is specified. The value 6 is the default and maximum thread limit.

```
THREADLIMIT=6 8
```

The first value (6) is the default thread limit. The second value (8) is the maximum thread limit.

Job-level limits

```
bsub -M 5000 myjob
```

Submits `myjob` with a memory limit of 5000 KB.

```
bsub -W 14 myjob
```

`myjob` is expected to run for 14 minutes. If the run limit specified with **bsub -W** exceeds the value for the queue, the job is rejected.

```
bsub -T 4 myjob
```

Submits `myjob` with a maximum number of concurrent threads of 4.

CPU time and run time normalization

To set the CPU time limit and run time limit for jobs in a platform-independent way, LSF scales the limits by the CPU factor of the hosts involved. When a job is dispatched to a host for execution, the limits are then normalized according to the CPU factor of the execution host.

Whenever a normalized CPU time or run time is given, the actual time on the execution host is the specified time multiplied by the CPU factor of the normalization host then divided by the CPU factor of the execution host.

If `ABS_RUNLIMIT=Y` is defined in `lsb.params` or in `lsb.applications` for the application associated with your job, the run time limit and run time estimate are not normalized by the host CPU factor. Absolute wall-clock run time is used for all jobs submitted with a run time limit or a run time estimate.

Normalization host

If no host or host model is given with the CPU time or run time, LSF uses the default CPU time normalization host defined at the queue level (`DEFAULT_HOST_SPEC` in `lsb.queues`) if it has been configured, otherwise uses the default CPU time normalization host defined at the cluster level (`DEFAULT_HOST_SPEC` in `lsb.params`) if it has been configured, otherwise uses the submission host.

Example

CPULIMIT=10/hostA

If `hostA` has a CPU factor of 2, and `hostB` has a CPU factor of 1 (`hostB` is slower than `hostA`), this specifies an actual time limit of 10 minutes on `hostA`, or on any other host that has a CPU factor of 2. However, if `hostB` is the execution host, the actual time limit on `hostB` is 20 minutes ($10 * 2 / 1$).

Normalization hosts for default CPU and run time limits

The first valid CPU factor encountered is used for both CPU limit and run time limit. To be valid, a host specification must be a valid host name that is a member of the LSF cluster. The CPU factor is used even if the specified limit is not valid.

If the CPU and run limit have different host specifications, the CPU limit host specification is enforced.

If no host or host model is given with the CPU or run time limits, LSF determines the default normalization host according to the following priority:

1. `DEFAULT_HOST_SPEC` is configured in `lsb.queues`
2. `DEFAULT_HOST_SPEC` is configured in `lsb.params`
3. If `DEFAULT_HOST_SPEC` is not configured in `lsb.queues` or `lsb.params`, host with the largest CPU factor is used.

CPU time display (bacct, bhist, bqueues)

Normalized CPU time is displayed in the output of **bqueues**. CPU time is *not* normalized in the output of **bacct** and **bhist**.

Memory and swap limit enforcement based on Linux cgroup memory subsystem

All LSF job processes are controlled by the Linux cgroup system so that cgroup memory and swap limits cannot be exceeded. These limits are enforced on a per job and per host basis, not per task, on Red Hat Enterprise Linux (RHEL) 6.2 or above and SuSe Linux Enterprise Linux 11 SP2 or above. LSF enforces memory and swap limits for jobs by periodically collecting job usage and comparing it with limits set by users. LSF can impose strict host-level memory and swap limits on systems that support Linux **cgroup v1** or **cgroup v2** cgroups.

Different LSF hosts in the cluster can use different versions of cgroup as long as each individual LSF host is only running one version of cgroup. If you have both versions of cgroup enabled in a host, you must disable one of the versions. For example, `hostA` can use **cgroup v1** and `hostB` can use **cgroup v2** as long as each host is only running one version of **cgroup**.

To enable memory enforcement through the Linux cgroup memory subsystem, configure the `LSB_RESOURCE_ENFORCE="memory"` parameter in the `lsf.conf` file.

If the host OS is Red Hat Enterprise Linux 6.3 or above, cgroup memory limits are enforced, and LSF is notified to terminate the job. More notification is provided to users through specific termination reasons that are displayed by **bhist -l**.

To change (such as reduce) the cgroup memory limits for running jobs, use **bmod -M** to change a job's memory limit, and use **bmod -v** to change the swap limit for a job. Note that when you request to change a job's memory or swap limits, LSF modifies the job's requirement to accommodate your request, and modifies the job's cgroup limits setting. If the OS rejects the cgroup memory or swap limit modifications, LSF posts a message to the job to indicate that the cgroup is not changed. After the cgroup limit changes, the OS can adjust the job's memory or swap allocation. As a best practice, do not decrease the cgroup memory or swap limit to less than your application use.

If you enable memory or swap enforcement through the Linux cgroup memory subsystem after you upgrade an existing LSF cluster, make sure that the following parameters are set in the `lsf.conf` file:

- `LSF_PROCESS_TRACKING=Y`
- `LSF_LINUX_CGROUP_ACCT=Y`

Setting `LSB_RESOURCE_ENFORCE="memory"` automatically turns on cgroup accounting (`LSF_LINUX_CGROUP_ACCT=Y`) to provide more accurate memory and swap consumption data for memory and swap enforcement checking. Setting `LSF_PROCESS_TRACKING=Y` enables LSF to kill jobs cleanly after memory and swap limits are exceeded.

Note: If `LSB_RESOURCE_ENFORCE="memory"` is configured, all existing LSF memory limit related parameters such as `LSF_HPC_EXTENSIONS="TASK_MEMLIMIT"`, `LSF_HPC_EXTENSIONS="TASK_SWAPLIMIT"`, `"LSB_JOB_MEMLIMIT"`, and `"LSB_MEMLIMIT_ENFORCE"` are ignored.

Example

Submit a parallel job with 3 tasks and a memory limit of 100 MB, with `span[ptile=2]` so that 2 tasks can run on one host and 1 task can run on another host:

```
bsub -n 3 -M 100 -R "span[ptile=2]" blaunch ./mem_eater
```

The application `mem_eater` keeps increasing the memory usage.

LSF kills the job at any point in time that it consumes more than 200 MB total memory on `hosta` or more than 100 MB total memory on `hostb`. For example, if at any time 2 tasks run on `hosta` and 1 task runs on `hostb`, the job is killed only if total memory consumed by the 2 tasks on `hosta` exceeds 200 MB on `hosta` or 100 MB in `hostb`.

LSF does not support per task memory enforcement for cgroups. For example, if one of the tasks on `hosta` consumes 150 MB memory and the other task consumes only 10 MB, the job is not killed because, at that point in time, the total memory that is consumed by the job on `hosta` is only 160 MB.

Memory enforcement does not apply to accumulated memory usage. For example, two tasks consume a maximum 250 MB on `hosta` in total. The maximum memory rusage of task1 on `hosta` is 150 MB and the maximum memory rusage of task2 on `hosta` is 100 MB, but this never happens at the same time, so at any given time, the two tasks consumes less than 200 MB and this job is not killed. The job would be killed only if at a specific point in time, the two tasks consume more than 200 MB on `hosta`.

Note: The cgroup memory subsystem does not separate enforcement of memory usage and swap usage. If a swap limit is specified, limit enforcement differs from previous LSF behavior. **bjobs -l** shows SWAP as 0. This is correct since swap device usage is not collected separately from memory usage.

For example, for the following job submission:

```
bsub -M 100 -v 50 ./mem_eater
```

After the application uses more than 100 MB of memory, the cgroup will start to use swap for the job process. The job is not killed until the application reaches 150 MB memory usage (100 MB memory + 50 MB swap).

The following job specifies only a swap limit:

```
bsub -v 50 ./mem_eater
```

Because no memory limit is specified, LSF considers the memory limit to be same as a swap limit. The job is killed when it reaches 50 MB combined memory and swap usage.

Host-based memory and swap limit enforcement by Linux cgroups

When the `LSB_RESOURCE_ENFORCE="memory"` parameter is configured in the `lsf.conf` file, memory and swap limits are calculated and enforced as a multiple of the number of tasks running on the execution host when memory and swap limits are specified for the job (at the job-level with `-M` and `-v`, or in `lsb.queues` or `lsb.applications` with `MEMLIMIT` and `SWAPLIMIT`).

The `bsub -hl` option enables job-level host-based memory and swap limit enforcement regardless of the number of tasks running on the execution host. The `LSB_RESOURCE_ENFORCE="memory"` parameter must be specified in `lsf.conf` for host-based memory and swap limit enforcement with the `-hl` option to take effect.

If no memory or swap limit is specified for the job (the merged limit for the job, queue, and application profile, if specified), or the `LSB_RESOURCE_ENFORCE="memory"` parameter is not specified, a host-based memory limit is not set for the job. The `-hl` option only applies only to memory and swap limits; it does not apply to any other resource usage limits.

Limitations and known issues

- For parallel jobs, cgroup limits are only enforced for jobs that are launched through the LSF **blaunch** framework. Parallel jobs that are launched through LSF PAM/**taskstarter** are not supported.
- On RHEL 6.2, LSF cannot receive notification from the cgroup that memory and swap limits are exceeded. When job memory and swap limits are exceeded, LSF cannot guarantee that the job is killed. On RHEL 6.3, LSF does receive notification and kills the job.
- On RHEL 6.2, a multithreaded application becomes a zombie process if the application is killed by cgroup due to memory enforcement. As a result, LSF cannot wait for the user application exited status and LSF processes are hung. LSF recognizes the job does not exit and the job always runs.

PAM resource limits

PAM limits are system resource limits defined in `limits.conf`.

- Windows: Not applicable
- Linux: `/etc/pam.d/lsf`

When `USE_PAM_CREDS` is set to `y` or `limits` in the `lsb.queues` or `lsb.applications` file, applies PAM limits to an application or queue when its job is dispatched to a Linux host using PAM. The job will fail if the execution host does not have PAM configured.

- [Configuring a PAM file](#)

Configuring a PAM file

About this task

When `USE_PAM_CREDS` is set to `y` or `limits` in the `lsb.queues` or `lsb.applications` file, LSF can apply PAM limits to an application when its job is dispatched to a Linux host using PAM LSF. The LSF job does not run within the PAM session. For these parameter settings, LSF assumes that the Linux PAM service "**lsf**" is created.

When `USE_PAM_CREDS` is set to `session` in the `lsb.queues` or `lsb.applications` file, LSF opens a PAM session for the user and executes a RES process into that session. The RES process executes the job task, then LSF disowns the process. This means that other LSF integrations are automatically handled in the PAM session. For this parameter setting, LSF assumes that the Linux PAM service "**lsf**-<clustername>" is created.

You can also specify `limits` and `sessions` together (`USE_PAM_CREDS=limits session`). For this parameter setting, LSF assumes that the Linux PAM services "**lsf**" and "**lsf**-<clustername>" are created.

If LSF limits are more restrictive than PAM limits, LSF limits are used, otherwise PAM limits are used. PAM limits are system resource limits defined in the `limits.conf` file.

The job **sbatchd** daemon checks the **lsf** service, and the job or task RES daemon checks the **lsf-*<clustername>*** service. The job will fail if the execution host does not have PAM configured.

Procedure

1. Create a PAM configuration file on each execution host you want.
`/etc/pam.d/lsf`
2. In the first two lines, specify the authentication and authorization you need to successfully run PAM limits. For example:

```
auth required pam_localuser.so  
  
account required pam_unix.so
```
3. Specify any resource limits. For example:

```
session required pam_limits.so
```

Results

For more information about configuring a PAM file, check Linux documentation.

Load thresholds

- **Automatic job suspension**
Jobs running under LSF can be suspended based on the load conditions on the execution hosts. Each host and each queue can be configured with a set of suspending conditions. If the load conditions on an execution host exceed either the corresponding host or queue suspending conditions, one or more jobs running on that host are suspended to reduce the load.
- **Suspending conditions**
LSF provides different alternatives for configuring suspending conditions. Suspending conditions are configured at the host level as load thresholds, whereas suspending conditions are configured at the queue level as either load thresholds, or by using the `STOP_COND` parameter in the `lsb.queues` file, or both.

Automatic job suspension

Jobs running under LSF can be suspended based on the load conditions on the execution hosts. Each host and each queue can be configured with a set of suspending conditions. If the load conditions on an execution host exceed either the corresponding host or queue suspending conditions, one or more jobs running on that host are suspended to reduce the load.

When LSF suspends a job, it invokes the `SUSPEND` action. The default `SUSPEND` action is to send the signal `SIGSTOP`.

By default, jobs are resumed when load levels fall below the suspending conditions. Each host and queue can be configured so that suspended checkpointable or rerunnable jobs are automatically migrated to another host instead.

If no suspending threshold is configured for a load index, LSF does not check the value of that load index when deciding whether to suspend jobs.

Suspending thresholds can also be used to enforce inter-queue priorities. For example, if you configure a low-priority queue with an `r1m` (1 minute CPU run queue length) scheduling threshold of 0.25 and an `r1m` suspending threshold of 1.75, this queue starts one job when the machine is idle. If the job is CPU intensive, it increases the run queue length from 0.25 to roughly 1.25. A high-priority queue configured with a scheduling threshold of 1.5 and an unlimited suspending threshold sends a second job to the same host, increasing the run queue to 2.25. This exceeds the suspending threshold for the low priority job, so it is stopped. The run queue length stays above 0.25 until the high priority job exits. After the high priority job exits the run queue index drops back to the idle level, so the low priority job is resumed.

When jobs are running on a host, LSF periodically checks the load levels on that host. If any load index exceeds the corresponding per-host or per-queue suspending threshold for a job, LSF suspends the job. The job remains suspended until the load levels satisfy the scheduling thresholds.

At regular intervals, LSF gets the load levels for that host. The period is defined by the `SBD_SLEEP_TIME` parameter in the `lsb.params` file. Then, for each job running on the host, LSF compares the load levels against the host suspending conditions and the queue suspending conditions. If any suspending condition at either the corresponding host or queue level is satisfied as a result of increased load, the job is suspended. A job is only suspended if the load levels are too high for that particular job's suspending thresholds.

There is a time delay between when LSF suspends a job and when the changes to host load are seen by the LIM. To allow time for load changes to take effect, LSF suspends no more than one job at a time on each host.

Jobs from the lowest priority queue are checked first. If two jobs are running on a host and the host is too busy, the lower priority job is suspended and the higher priority job is allowed to continue. If the load levels are still too high on the next turn, the higher priority job is also suspended.

If a job is suspended because of its own load, the load drops as soon as the job is suspended. When the load goes back within the thresholds, the job is resumed until it causes itself to be suspended again.

Exceptions

In some special cases, LSF does not automatically suspend jobs because of load levels. LSF does not suspend a job:

- Forced to run with **`brun -f`**.
- If it is the only job running on a host, unless the host is being used interactively. When only one job is running on a host, it is not suspended for any reason except that the host is not interactively idle (the interactive idle time load index is less than one minute). This means that once a job is started on a host, at least one job continues to run unless there is an interactive user on the host. Once the job is suspended, it is not resumed until all the scheduling conditions are met, so it should not interfere with the interactive user.
- Because of the paging rate, unless the host is being used interactively. When a host has interactive users, LSF suspends jobs with high paging rates, to improve the response time on the host for interactive users. When a host is idle, the pg (paging rate) load index is ignored. The `PG_SUSP_IT` parameter in `lsb.params` controls this behavior. If the host has been idle for more than `PG_SUSP_IT` minutes, the pg load index is not checked against the suspending threshold.

Suspending conditions

LSF provides different alternatives for configuring suspending conditions. Suspending conditions are configured at the host level as load thresholds, whereas suspending conditions are configured at the queue level as either load thresholds, or by using the `STOP_COND` parameter in the `lsb.queues` file, or both.

The load indices most commonly used for suspending conditions are the CPU run queue lengths (`r15s`, `r1m`, and `r15m`), paging rate (`pg`), and idle time (`it`). The (`swp`) and (`tmp`) indices are also considered for suspending jobs.

To give priority to interactive users, set the suspending threshold on the `it` (idle time) load index to a non-zero value. Jobs are stopped when any user is active, and resumed when the host has been idle for the time given in the `it` scheduling condition.

To tune the suspending threshold for paging rate, it is desirable to know the behavior of your application. On an otherwise idle machine, check the paging rate using **`lsload`**, and then start your application. Watch the paging rate as the application runs. By subtracting the active paging rate from the idle paging rate, you get a number for the paging rate of your application. The suspending threshold should allow at least 1.5 times that amount. A job can be scheduled at any paging rate up to the scheduling threshold, so the suspending threshold should be at least the scheduling threshold plus 1.5 times the application paging rate. This prevents the system from scheduling a job and then immediately suspending it because of its own paging.

The effective CPU run queue length condition should be configured like the paging rate. For CPU-intensive sequential jobs, the effective run queue length indices increase by approximately one for each job. For jobs that use more than one process, you should make some test runs to determine your job's effect on the run queue length indices. Again, the suspending threshold should be equal to at least the scheduling threshold plus 1.5 times the load for one job.

Re-sizable jobs

If new hosts are added for re-sizable jobs, LSF considers load threshold scheduling on those new hosts. If hosts are removed from allocation, LSF does not apply load threshold scheduling for resizing the jobs.

- [Configuring suspending conditions at queue level](#)
- [About resuming suspended jobs](#)
Jobs are suspended to prevent overloading hosts, to prevent batch jobs from interfering with interactive use, or to allow a more urgent job to run. When the host is no longer overloaded, suspended jobs should continue running.
- [Specifying resume condition](#)

Configuring load thresholds at queue level

The queue definition (lsb.queues) can contain thresholds for 0 or more of the load indices. Any load index that does not have a configured threshold has no effect on job scheduling.

Syntax

Each load index is configured on a separate line with the format:

```
load_index = loadSched/loadStop
```

Specify the name of the load index, for example r1m for the 1-minute CPU run queue length or pg for the paging rate. loadSched is the scheduling threshold for this load index. loadStop is the suspending threshold. The loadSched condition must be satisfied by a host before a job is dispatched to it and also before a job suspended on a host can be resumed. If the loadStop condition is satisfied, a job is suspended.

The loadSched and loadStop thresholds permit the specification of conditions using simple AND/OR logic. For example, the specification:

```
MEM=100/10 SWAP=200/30
```

translates into a loadSched condition of `mem>=100 && swap>=200` and a loadStop condition of `mem < 10 || swap < 30`.

Theory

- The r15s, r1m, and r15m CPU run queue length conditions are compared to the effective queue length as reported by **lsload -E**, which is normalized for multiprocessor hosts. Thresholds for these parameters should be set at appropriate levels for single processor hosts.
- Configure load thresholds consistently across queues. If a low priority queue has higher suspension thresholds than a high priority queue, then jobs in the higher priority queue are suspended before jobs in the low priority queue.

Load thresholds at host level

A shared resource cannot be used as a load threshold in the Hosts section of the `lsf.cluster.cluster_name` file.

Configuring suspending conditions at queue level

About this task

The condition for suspending a job can be specified using the queue-level STOP_COND parameter. It is defined by a resource requirement string. Only the select section of the resource requirement string is considered when stopping a job. All other sections are ignored.

This parameter provides similar but more flexible functionality for loadStop.

If loadStop thresholds have been specified, then a job is suspended if either the STOP_COND is TRUE or the loadStop thresholds are exceeded.

Procedure

Modify a queue to suspend a job based on a condition.

For example, suspend a job based on the idle time for desktop machines and availability of swap and memory on compute servers.

Assume `cs` is a Boolean resource defined in the `lsf.shared` file and configured in the `lsf.cluster.cluster_name` file to indicate that a host is a compute server

Begin Queue

```
STOP_COND= select[(!cs && it < 5) || (cs && mem < 15 && swap < 50)]
```

End Queue

About resuming suspended jobs

Jobs are suspended to prevent overloading hosts, to prevent batch jobs from interfering with interactive use, or to allow a more urgent job to run. When the host is no longer overloaded, suspended jobs should continue running.

When LSF automatically resumes a job, it invokes the RESUME action. The default action for RESUME is to send the signal SIGCONT.

If there are any suspended jobs on a host, LSF checks the load levels in each dispatch turn.

If the load levels are within the scheduling thresholds for the queue and the host, and all the resume conditions for the queue (RESUME_COND in `lsb.queue`s) are satisfied, the job is resumed.

If RESUME_COND is not defined, then the `loadSched` thresholds are used to control resuming of jobs: all the `loadSched` thresholds must be satisfied for the job to be resumed. The `loadSched` thresholds are ignored if RESUME_COND is defined.

Jobs from higher priority queues are checked first. To prevent overloading the host again, only one job is resumed in each dispatch turn.

Specifying resume condition

Procedure

Use RESUME_COND in `lsb.queue`s to specify the condition that must be satisfied on a host if a suspended job is to be resumed. Only the select section of the resource requirement string is considered when resuming a job. All other sections are ignored.

Time configuration

Learn about dispatch and run windows and deadline constraint scheduling.

Dispatch and run windows are time windows that control when LSF jobs start and run. Deadline constraints suspend or terminate running jobs at a certain time.

Note: Not all time windows take effect immediately. LSF might take some time to apply all time windows.

- [Time windows](#)
To specify a time window, specify two time values separated by a hyphen (-), with no space in between.
- [Time expressions](#)
Time expressions use time windows to specify when to change configurations.

- [Automatic time-based configuration](#)

Variable configuration is used to automatically change LSF configuration based on time windows. It is supported in the following files:

- [Dispatch and run windows](#)

Both dispatch and run windows are time windows that control when LSF jobs start and run.

- [Deadline constraint scheduling](#)

Deadline constraints suspend or terminate running jobs at a certain time.

Time windows

To specify a time window, specify two time values separated by a hyphen (-), with no space in between.

```
time_window = begin_time-end_time
```

Time format

Times are specified in the format:

```
[day:]hour[:minute]
```

where all fields are numbers with the following ranges:

- *day of the week*: 0-7 (0 and 7 are both Sunday)
- *hour*: 0-23
- *minute*: 0-59

Specify a time window one of the following ways:

- *hour-hour*
- *hour:minute-hour:minute*
- *day:hour:minute-day:hour:minute*

The default value for minute is 0 (on the hour); the default value for day is every day of the week.

You must specify at least the hour. Day of the week and minute are optional. Both the start time and end time values must use the same syntax. If you do not specify a minute, LSF assumes the first minute of the hour (:00). If you do not specify a day, LSF assumes every day of the week. If you do specify the day, you must also specify the minute.

Examples of time windows

Daily window

To specify a daily window omit the day field from the time window. Use either the hour-hour or hour:minute-hour:minute format. For example, to specify a daily 8:30AM to 6:30 PM window:

```
8:30-18:30
```

Overnight window

To specify an overnight window make time1 greater than time2. For example, to specify 6:30 PM to 8:30 AM the following day:

```
18:30-8:30
```

Weekend window

To specify a weekend window use the day field. For example, to specify Friday at 6:30 PM to Monday at 8:30 AM:

```
5:18:30-1:8:30
```

Time expressions

Time expressions use time windows to specify when to change configurations.

Time expression syntax

A time expression is made up of the time keyword followed by one or more space-separated time windows enclosed in parentheses. Time expressions can be combined using the &&, ||, and ! logical operators.

The syntax for a time expression is:

```
expression = time(time_window[ time_window ...])
            | expression && expression
            | expression || expression
            | !expression
```

Example

Both of the following expressions specify weekends (Friday evening at 6:30 PM until Monday morning at 8:30 AM) and nights (8:00 PM to 8:30 A daily).

```
time(5:18:30-1:8:30 20:00-8:30)
time(5:18:30-1:8:30) || time(20:00-8:30)
```

Automatic time-based configuration

Variable configuration is used to automatically change LSF configuration based on time windows. It is supported in the following files:

- lsb.hosts
- lsb.params
- lsb.queues
- lsb.resources
- lsb.users
- lsf.licensescheduler
- lsb.applications

You define automatic configuration changes in configuration files by using if-else constructs and time expressions. After you change the files, reconfigure the cluster with the **badadmin reconfig** command.

The expressions are evaluated by LSF every 10 minutes based on **mbatchd** start time. When an expression evaluates true, LSF dynamically changes the configuration based on the associated configuration statements. Reconfiguration is done in real time without restarting **mbatchd**, providing continuous system availability.

In the following examples, the #if, #else, #endif are not interpreted as comments by LSF but as if-else constructs.

lsb.hosts example

```
Begin Host
HOST_NAME    r15s    r1m    pg
host1        3/5     3/5     12/20
#if time(5:16:30-1:8:30 EDT 20:00-8:30 EDT)
host2        3/5     3/5     12/20
#else
host2        2/3     2/3     10/12
#endif
host3        3/5     3/5     12/20
End Host
```

lsb.params example

```
# if 18:30-19:30 is your short job express period, but
# you want all jobs going to the short queue by default
# and be subject to the thresholds of that queue
# for all other hours, normal is the default queue
#if time(18:30-19:30 EDT)
DEFAULT_QUEUE=short
#else
DEFAULT_QUEUE=normal
#endif
```

lsb.queues example

```
Begin Queue
...
#if time(8:30-18:30 EDT)
INTERACTIVE = ONLY # interactive only during day shift #endif
#endif
...
End Queue
```

lsb.users example

From 12 - 1 p.m. daily, user `smith` has 10 job slots, but during other hours, user has only five job slots.

```
Begin User
USER_NAME  MAX_JOBS  JL/P
#if time(12-13 EDT)
smith      10        -
#else
smith      5         -
default    1         -
#endif
End User
```

Create if-else constructs

The if-else construct can express single decisions and multi-way decisions by including `elif` statements in the construct.

If-else

The syntax for constructing if-else expressions is:

```
#if time(expression) statement#elsetatement#endif
```

The `#endif` part is mandatory and the `#else` part is optional.

elif

The `#elif` expressions are evaluated in order. If any expression is true, the associated statement is used, and this terminates the whole chain.

The `#else` part handles the default case where none of the other conditions are satisfied.

When you use `#elif`, the `#else` and `#endif` parts are mandatory.

```
#if time(expression)
statement
#elif time(expression)
statement
#elif time(expression)
statement
#else
statement
#endif
```

Dispatch and run windows

Both dispatch and run windows are time windows that control when LSF jobs start and run.

- Dispatch windows can be defined in `lsb.hosts`. Dispatch and run windows can be defined in `lsb.queues`.
- Hosts can only have dispatch windows. Queues can have dispatch windows and run windows.
- Both windows affect job starting; only run windows affect the stopping of jobs.
- Dispatch windows define when hosts and queues are active and inactive. It does not control job submission.
- Run windows define when jobs can and cannot run. While a run window is closed, LSF cannot start any of the jobs placed in the queue, or finish any of the jobs already running.
- When a dispatch window closes, running jobs continue and finish, and no new jobs can be dispatched to the host or from the queue. When a run window closes, LSF suspends running jobs, but new jobs can still be submitted to the queue.
- [Run windows](#)
Queues can be configured with a run window, which specifies one or more time periods during which jobs in the queue are allowed to run. Once a run window is configured, jobs in the queue cannot run outside of the run window.
- [Dispatch windows](#)
Queues can be configured with a dispatch window, which specifies one or more time periods during which jobs are accepted. Hosts can be configured with a dispatch window, which specifies one or more time periods during which jobs are allowed to start.

Run windows

Queues can be configured with a run window, which specifies one or more time periods during which jobs in the queue are allowed to run. Once a run window is configured, jobs in the queue cannot run outside of the run window.

Jobs can be submitted to a queue at any time; if the run window is closed, the jobs remain pending until it opens again. If the run window is open, jobs are placed and dispatched as usual. When an open run window closes, running jobs are suspended, and pending jobs remain pending. The suspended jobs are resumed when the window opens again.

- [Configuring run windows](#)
- [Viewing information about run windows](#)

Configuring run windows

Procedure

To configure a run window, set `RUN_WINDOW` in `lsb.queues`.

To use time zones, specify a supported time zone after the time window. If you do not specify a time zone, LSF uses the local system time zone. LSF supports all standard time zone abbreviations.

To specify that the run window will be open from 4:30 a.m. to noon, specify the following:

```
RUN_WINDOW = 4:30-12:00
```

To specify that the run window will be open from 4:30 a.m. to noon in Eastern Daylight Time, specify the following:

```
RUN_WINDOW = 4:30-12:00 EDT
```

You can specify multiple time windows, but all time window entries must be consistent in whether they set the time zones. That is, either all entries must set a time zone, or all entries must not set a time zone.

Viewing information about run windows

Procedure

Use **bqueues -l** to display information about queue run windows.

Dispatch windows

Queues can be configured with a dispatch window, which specifies one or more time periods during which jobs are accepted. Hosts can be configured with a dispatch window, which specifies one or more time periods during which jobs are allowed to start.

Once a dispatch window is configured, LSF cannot dispatch jobs outside of the window. By default, no dispatch windows are configured (the windows are always open).

Dispatch windows have no effect on jobs that have already been dispatched to the execution host; jobs are allowed to run outside the dispatch windows, as long as the queue run window is open.

Queue-level

Each queue can have a dispatch window. A queue can only dispatch jobs when the window is open.

You can submit jobs to a queue at any time; if the queue dispatch window is closed, the jobs remain pending in the queue until the dispatch window opens again.

Host-level

Each host can have dispatch windows. A host is not eligible to accept jobs when its dispatch windows are closed.

- [Configuring host dispatch windows](#)
- [Configuring queue dispatch windows](#)
- [Displaying host dispatch windows](#)
- [Displaying queue dispatch windows](#)

Configuring host dispatch windows

Procedure

To configure dispatch windows for a host, set `DISPATCH_WINDOW` in `lsb.hosts` and specify one or more time windows. If no host dispatch window is configured, the window is always open.

Configuring queue dispatch windows

Procedure

To configure dispatch windows for queues, set `DISPATCH_WINDOW` in `lsb.queues` and specify one or more time windows. If no queue dispatch window is configured, the window is always open.

Displaying host dispatch windows

Procedure

Use **bhosts -l** to display host dispatch windows.

Displaying queue dispatch windows

Procedure

Use **bqueues -l** to display queue dispatch windows.

Deadline constraint scheduling

Deadline constraints suspend or terminate running jobs at a certain time.

There are two kinds of deadline constraints:

- A run window, specified at the queue level, suspends a running job
- A termination time, specified at the job level (**bsub -t**), terminates a running job

Time-based resource usage limits

- A CPU limit, specified at job or queue level, terminates a running job when it has used up a certain amount of CPU time.
- A run limit, specified at the job or queue level, terminates a running job after it has spent a certain amount of time in the RUN state.

How deadline constraint scheduling works

If deadline constraint scheduling is enabled, LSF does not place a job that will be interrupted by a deadline constraint before its run limit expires, or before its CPU limit expires, if the job has no run limit. In this case, deadline constraint scheduling could prevent a job from ever starting. If a job has neither a run limit nor a CPU limit, deadline constraint scheduling has no effect.

A job that cannot start because of a deadline constraint causes an email to be sent to the job owner.

Deadline constraint scheduling only affects the placement of jobs. Once a job starts, if it is still running at the time of the deadline, it will be suspended or terminated because of the deadline constraint or resource usage limit.

Re-sizable jobs

LSF considers both job termination time and queue run windows as part of deadline constraints. Since the job has already started, LSF does not apply deadline constraint scheduling to job resize allocation requests.

- [Disabling deadline constraint scheduling](#)

Disabling deadline constraint scheduling

About this task

Deadline constraint scheduling is enabled by default.

Procedure

To disable deadline constraint scheduling for a queue, set IGNORE_DEADLINE=y in `lsb.queues`.

Example

LSF schedules jobs in the `liberal` queue without observing the deadline constraints.

```
Begin Queue
QUEUE_NAME = liberal
IGNORE_DEADLINE=y
End Queue
```

Preemptive scheduling

The preemptive scheduling feature allows a pending high-priority job to preempt a running job of lower priority. The lower-priority job is suspended and is resumed as soon as possible. Use preemptive scheduling if you have long-running, low-priority jobs causing high-priority jobs to wait an unacceptably long time.

- [Resource preemption](#)
- [About preemptive scheduling](#)
- [Configuration to enable preemptive scheduling](#)
The preemptive scheduling feature is enabled by defining at least one queue as preemptive or pre-emptable, using the PREEMPTION parameter in the `lsb.queues` file. Preemption does not actually occur until at least one queue is assigned a higher relative priority than another queue, using the PRIORITY parameter, which is also set in the `lsb.queues` file.
- [Preemptive scheduling behavior](#)
Preemptive scheduling is based primarily on parameters specified at the queue level: some queues are eligible for preemption, others are not. Once a hierarchy of queues has been established, other factors, such as queue priority and preferred preemption order, determine which jobs from a queue should be preempted.
- [Configuration to modify preemptive scheduling behavior](#)
There are configuration parameters that modify various aspects of preemptive scheduling behavior.
- [Preemptive scheduling commands](#)

Resource preemption

- [About resource preemption](#)
- [Requirements for resource preemption](#)
Resource preemption depends on many conditions:
- [Custom job controls for resource preemption](#)
- [Preempting resources](#)
- [Configuring resource preemption](#)
- [Memory preemption](#)

About resource preemption

Preemptive scheduling and resource preemption

Resource preemption is a special type of preemptive scheduling. It is similar to job slot preemption.

Job slot preemption and resource preemption

If you enable preemptive scheduling, job slot preemption is always enabled. Resource preemption is optional. With resource preemption, you can configure preemptive scheduling that is based on other resources in addition to job slots.

Other Resources

Resource preemption works for any custom shared numeric resource (except increasing dynamic resources). To preempt on a host-based resource, such as memory, you could configure a custom resource "shared" on only one host.

Multiple resource preemption

If multiple resources are required, LSF can preempt multiple jobs until sufficient resources are available. For example, one or more jobs might be preempted for a job that needs:

- Multiple job slots
- Multiple resources, such as a job slots and memory
- More of a resource than can be obtained by preempting just one job

Use resource preemption

To allow your job to participate in resource preemption, you must use resource reservation to reserve the preemption resource (the cluster might be configured so that this occurs automatically). For dynamic resources, you must specify a duration also.

Resource reservation is part of resource requirement, which can be specified at the job level or at the queue level or application level.

You can use a task file to associate specific resource requirements with specific applications.

Dynamic resources

Specify duration

If the preemption resource is dynamic, you must specify the duration part of the resource reservation string when you submit a preempting or preemptable job.

Resources outside the control of LSF

If an ELIM is needed to determine the value of a dynamic resource, LSF preempts jobs as necessary, then waits for ELIM to report that the resources are available before starting the high-priority job. By default, LSF waits 300 seconds (5 minutes) for resources to become available. This time can be increased (PREEMPTION_WAIT_TIME in lsb.params).

If the preempted jobs do not release the resources, or the resources have been intercepted by a non-LSF user, the ELIM does not report any more of the resource becoming available, and LSF might preempt more jobs to get the resources.

Requirements for resource preemption

Resource preemption depends on many conditions:

- The preemption resources must be configured (PREEMPTABLE_RESOURCES in lsb.params).
- Jobs must reserve the correct amount of the preemption resource, using resource reservation (the rusage part of the resource requirement string).
- For dynamic preemption resources, jobs must specify the duration part of the resource reservation string.
- Jobs that use the preemption resource must be spread out among multiple queues of different priority, and preemptive scheduling must be configured so that preemption can occur among these queues (preemption can only occur if jobs are in different queues).
- Only a releasable resource can be a preemption resource. LSF must be configured to release the preemption resource when the job is suspended (RELEASE=Y in lsf.shared, which is the default). You must configure this no matter what your

preemption action is.

- LSF's preemption behavior must be modified. By default, LSF's default preemption action does not allow an application to release any resources, except for job slots and static shared resources.

Custom job controls for resource preemption

Why you have to customize LSF

By default, LSF's preemption action is to send a suspend signal (SIGSTOP) to stop the application. Some applications do not release resources when they get SIGSTOP. If this happens, the preemption resource does not become available, and the preempting job is not successful.

You modify LSF's default preemption behavior to make the application release the preemption resource when a job is preempted.

Customize the SUSPEND action

Ask your application vendor what job control signals or actions cause your application to suspend a job and release the preemption resources. You need to replace the default SUSPEND action (the SIGSTOP signal) with another signal or script that works properly with your application when it suspends the job. For example, your application might be able to catch SIGTSTP instead of SIGSTOP.

By default, LSF sends SIGCONT to resume suspended jobs. You should find out if this causes your application to take back the resources when it resumes the job. If not, you need to modify the RESUME action also.

Whatever changes you make to the SUSPEND job control affects all suspended jobs in the queue, including preempted jobs, jobs that are suspended because of load thresholds, and jobs that you suspend using LSF commands. Similarly, changes made to the RESUME job control also affect the whole queue.

Kill preempted jobs

If you want to use resource preemption, but cannot get your application to release or take back the resource, you can configure LSF to kill the low-priority job instead of suspending it. This method is less efficient because when you kill a job, you lose all the work, and you have to restart the job from the beginning.

- You can configure LSF to kill and requeue suspended jobs (use brequeue as the SUSPEND job control in lsb.queueues). This kills all jobs that are suspended in the queue, not just preempted jobs.
- You can configure LSF to kill preempted jobs instead of suspending them (TERMINATE_WHEN=PREEMPT in lsb.queueues). In this case, LSF does not restart the preempted job, you have to resubmit it manually.

Preempting resources

About this task

To make resource preemption useful, you may need to work through all of these steps.

Procedure

1. Read.

Before you set up resource preemption, you should understand the following:

- Preemptive Scheduling
- Resource Preemption

- Resource Reservation
 - Customizing Resources
 - Customizing Job Controls
2. Plan.
When you plan how to set up resource preemption, consider:
 - Custom job controls: Find out what signals or actions you can use with your application to control the preemption resource when you suspend and resume jobs.
 - Existing cluster configuration: Your design might be based on preemptive queues or custom resources that are already configured in your cluster.
 - Requirements for resource preemption: Your design must be able to work. If a host-based resource such as memory is the preemption resource, you cannot set up only one queue for each host because preemption occurs when 2 jobs are competing for the same resource.
 3. Write the ELIM.
 4. Configure LSF.
 - a. `lsb.queues`
 - Set `PREEMPTION` in at least one queue (to `PREEMPTIVE` in a high-priority queue, or to `PREEMPTABLE` in a low-priority queue).
 - Set `JOB_CONTROLS` (or `TERMINATE_WHEN`) in the low-priority queues. Optional. Set `RES_REQ` to automatically reserve the custom resource.
 - b. `lsf.shared`
Define the custom resource in the Resource section.
 - c. `lsb.params`
 - Set `PREEMPTABLE_RESOURCES` and specify the custom resource.
 - Optional. Set `PREEMPTION_WAIT_TIME` to specify how many seconds to wait for dynamic resources to become available.
 - Optional. Set `PREEMPT_JOBTYPE` to enable preemption of exclusive and backfill jobs. Specify one or both of the keywords `EXCLUSIVE` and `BACKFILL`. By default, exclusive and backfill jobs are only preempted if the exclusive low priority job is running on a host that is different than the one used by the preemptive high priority job.
 - d. `lsf.cluster.cluster_name`
Define how the custom resource is shared in the ResourceMap section.
 5. Reconfigure LSF to make your changes take effect.
 6. Operate.
 - Use resource reservation to reserve the preemption resource (this might be configured to occur automatically). For dynamic resources, you must specify a duration as well as a quantity.
 - Distribute jobs that use the preemption resource in way that allows preemption to occur between queues (this should happen as a result of the cluster design).
 7. Track.
Use **bparams -l** to view information about preemption configuration in your cluster.

Configuring resource preemption

Procedure

1. Configure preemptive scheduling (`PREEMPTION` in `lsb.queues`).
2. Configure the preemption resources (`PREEMPTABLE_RESOURCES` in `lsb.params`).
Job slots are the default preemption resource. To define additional resources to use with preemptive scheduling, set `PREEMPTABLE_RESOURCES` in `lsb.params`, and specify the names of the custom resources as a space-separated list.
3. Customize the preemption action.
Preemptive scheduling uses the `SUSPEND` and `RESUME` job control actions to suspend and resume preempted jobs. For resource preemption, it is critical that the preempted job releases the resource. You must modify LSF default job controls to make resource preemption work.
 - Suspend using a custom job control.

To modify the default suspend action, set `JOB_CONTROLS` in `lsb.queues` and use replace the `SUSPEND` job control with a script or a signal that your application can catch. Do this for all queues where there could be preemptable jobs using the preemption resources.

For example, if your application vendor tells you to use the `SIGTSTP` signal, set `JOB_CONTROLS` in `lsb.queues` and use `SIGTSTP` as the `SUSPEND` job control:

```
JOB_CONTROLS = SUSPEND [SIGTSTP]
```

- Kill jobs with **brequeue**.

To kill and requeue preempted jobs instead of suspending them, set `JOB_CONTROLS` in `lsb.queues` and use `brequeue` as the `SUSPEND` job control:

```
JOB_CONTROLS = SUSPEND [brequeue $LSB_JOBID]
```

Do this for all queues where there could be preemptable jobs using the preemption resources. This kills a preempted job, and then requeues it so that it has a chance to run and finish successfully.

- Kill jobs with `TERMINATE_WHEN`.

To kill preempted jobs instead of suspending them, set `TERMINATE_WHEN` in `lsb.queues` to `PREEMPT`. Do this for all queues where there could be preemptable jobs using the preemption resources.

If you do this, the preempted job does not get to run unless you resubmit it.

4. Optional. Configure the preemption wait time.

To specify how long LSF waits for the `ELIM` to report that the resources are available, set `PREEMPTION_WAIT_TIME` in `lsb.params` and specify the number of seconds to wait. You cannot specify any less than the default time (300 seconds).

For example, to make LSF wait for 8 minutes, specify

```
PREEMPTION_WAIT_TIME=480
```

Memory preemption

Configure memory preemption

By default, memory is not be preemptable. To enable memory preemption, specify the `mem` keyword in the value of the `PREEMPTABLE_RESOURCES` parameter in the `lsb.params` file. LSF preempts on both slots and memory.

Jobs with rusage duration

Users are permitted to submit jobs with `rusage` duration on memory. However, `rusage` duration does not take effect on memory when memory preemption is enabled. LSF continues to reserve memory for a job while it resides on a host.

OS memory behavior

When a job is suspended, it might continue to occupy physical memory. Unless there is another process on the host that can use the memory, the job might not release memory. If LSF launches another job on the host that can use the memory, the OS can start swapping pages of the suspended job out to disk. LSF does not look at swap space as a criterion for preemption.

When jobs exceed their memory requests

If a low priority job exceeds memory allocation on a host and a high priority job needs that memory allocation, you cannot get the memory allocation back through preemption.

For example, suppose that a host has a total of 8 GB of memory. A low priority job is submitted, requesting 4 GB of memory. However, once the job starts it uses all 8 GB of memory.

A high priority job is submitted that requests 8 GB of memory. LSF sees that there is no memory free on the host. The preemption module calculates that 4 GB of memory can be obtained by preempting the low priority job. This is not sufficient for the high priority job, so no preemption occurs.

About preemptive scheduling

Preemptive scheduling takes effect when two jobs compete for the same job slots. If a high-priority job is pending, LSF can suspend a lower-priority job that is running, and then start the high-priority job instead. For this to happen, the high-priority job must be pending in a *preemptive queue* (a queue that can preempt other queues), or the low-priority job must belong to a *preemptable queue* (a queue that can be preempted by other queues).

If multiple slots are required, LSF can preempt multiple jobs until sufficient slots are available. For example, one or more jobs can be preempted for a job that needs multiple job slots.

A preempted job is resumed as soon as more job slots become available; it does not necessarily have to wait for the preempting job to finish.

Preemptive queue

Jobs in a preemptive queue can preempt jobs in any queue of lower priority, even if the lower-priority queues are not specified as preemptable.

Preemptive queues are more aggressive at scheduling jobs because a slot that is not available to a low-priority queue may be available by preemption to a high-priority queue.

Preemptable queue

Jobs in a preemptable queue can be preempted by jobs from any queue of a higher priority, even if the higher-priority queues are not specified as preemptive.

When multiple preemptable jobs exist (low-priority jobs holding the required slots), and preemption occurs, LSF preempts a job from the least-loaded host.

Re-sizable jobs

Resize allocation requests are not able to take advantage of the queue-based preemption mechanism to preempt other jobs. However, regular pending jobs are still able to preempt running re-sizable jobs, even while they have a resize request pending. When a re-sizable job is preempted and goes to the SSUSP state, its resize request remains pending and LSF stops scheduling it until it returns back to RUN state.

- New pending allocation requests cannot make use of preemption policy to get slots from other running or suspended jobs.
- Once a resize decision has been made, LSF updates its job counters to be reflected in future preemption calculations. For instance, resizing a running pre-emptable job from 2 slots to 4 slots, makes 4 pre-emptable slots for high priority pending jobs.
- If a job is suspended, LSF stops allocating resources to a pending resize request.
- When a preemption decision is made, if job has pending resize request and scheduler already has made an allocation decision for this request, LSF cancels the allocation decision.
- If a preemption decision is made while a job resize notification command is running, LSF prevents the suspend signal from reaching the job.

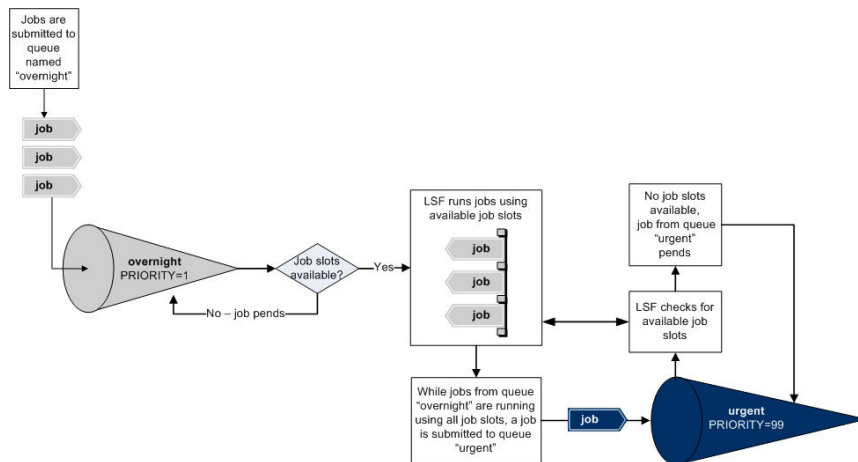
Scope

By default, pre-emptive scheduling does not apply to jobs that have been forced to run (using **brun**) or backfill and exclusive jobs.

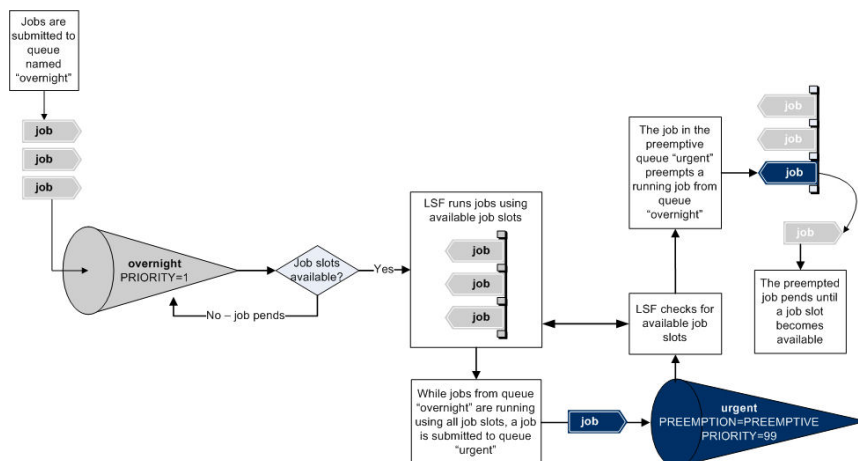
Limitations	Description
Exclusive jobs	Jobs requesting exclusive use of resources cannot preempt other jobs.
	Jobs using resources exclusively cannot be preempted.

Limitations	Description
Backfill jobs	Jobs backfilling future advance reservations cannot be preempted.
brun	Jobs forced to run with the command brun cannot be preempted.

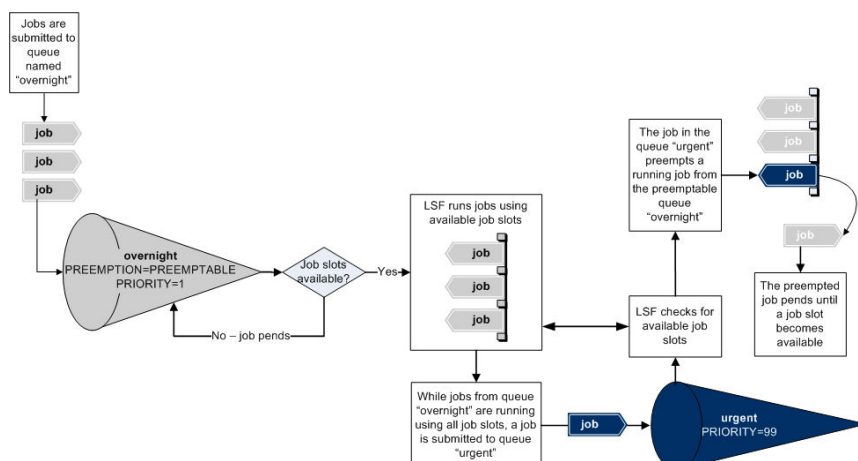
Default behavior (preemptive scheduling not enabled)



With preemptive scheduling enabled (preemptive queue)



With preemptive scheduling enabled (preemptable queue)



Configuration to enable preemptive scheduling

The preemptive scheduling feature is enabled by defining at least one queue as preemptive or pre-emptable, using the PREEMPTION parameter in the **lsb.queues** file. Preemption does not actually occur until at least one queue is assigned a higher relative priority than another queue, using the PRIORITY parameter, which is also set in the **lsb.queues** file.

Both PREEMPTION and PRIORITY are used to determine which queues can preempt other queues, either by establishing relative priority of queues or by specifically defining preemptive properties for a queue.

Configuration file	Parameter and syntax	Default behavior
lsb.queues	PREEMPTION=P REEMPTIVE	<ul style="list-style-type: none">Enables preemptive schedulingJobs in this queue can preempt jobs in any queue of lower priority, even if the lower-priority queue is not specified as preemptable
	PREEMPTION=P REEMPTABLE	<ul style="list-style-type: none">Enables preemptive schedulingJobs in this queue can be preempted by jobs from any queue of higher priority, even if the higher-priority queue is not specified as preemptive
	PRIORITY= <i>integer</i>	<ul style="list-style-type: none">Sets the priority for this queue relative to all other queuesThe larger the number, the higher the priority—a queue with PRIORITY=99 has a higher priority than a queue with PRIORITY=1

Preemptive scheduling behavior

Preemptive scheduling is based primarily on parameters specified at the queue level: some queues are eligible for preemption, others are not. Once a hierarchy of queues has been established, other factors, such as queue priority and preferred preemption order, determine which jobs from a queue should be preempted.

There are three ways to establish which queues should be preempted:

- Based on queue priority: the PREEMPTION parameter defines a queue as preemptive or pre-emptable and preemption is based on queue priority, where jobs from higher-priority queues can preempt jobs from lower-priority queues
- Based on a preferred order: the PREEMPTION parameter defines queues that can preempt other queues, in a preferred order
- Explicitly, by specific queues: the PREEMPTION parameter defines queues that can be preempted, and by which queues

Preemption configuration	Behavior
Preemption is not enabled—no queue is defined as preemptable, and no queue is defined as preemptive	High-priority jobs do not preempt jobs that are already running
A queue is defined as preemptable, but no specific queues are listed that can preempt it	Jobs from this queue can be preempted by jobs from any queue with a higher value for priority
A queue is defined as preemptable, and one or more queues are specified that can preempt it	Jobs from this queue can be preempted only by jobs from the specified queues
A queue is defined as preemptive, but no specific queues are listed that it can preempt	<ul style="list-style-type: none">Jobs from this queue preempt jobs from all queues with a lower value for priorityJobs are preempted from the least-loaded host
A queue is defined as preemptive, and one or more specific queues are listed that it can preempt, but no queue preference is specified	<ul style="list-style-type: none">Jobs from this queue preempt jobs from any queue in the specified listJobs are preempted on the least-loaded host first

Preemption configuration	Behavior
A queue is defined as preemptive, and one or more queues have a preference number specified, indicating a preferred order of preemption	<ul style="list-style-type: none"> Queues with a preference number are preferred for preemption over queues without a preference number Queues with a higher preference number are preferred for preemption over queues with a lower preference number For queues that have the same preference number, the queue with lowest priority is preferred for preemption over queues with higher priority For queues without a preference number, the queue with lower priority is preferred for preemption over the queue with higher priority
A queue is defined as preemptive, or a queue is defined as preemptable, and preemption of jobs with the shortest run time is configured	<ul style="list-style-type: none"> A queue from which to preempt a job is determined based on other parameters as shown above The job that has been running for the shortest period of time is preempted
A queue is defined as preemptive, or a queue is defined as preemptable, and preemption of jobs that will finish within a certain time period is prevented	<ul style="list-style-type: none"> A queue from which to preempt a job is determined based on other parameters as shown above A job that has a run limit or a run time specified and that will not finish within the specified time period is preempted
A queue is defined as preemptive, or a queue is defined as preemptable, and preemption of jobs with the specified run time is prevented	<ul style="list-style-type: none"> A queue from which to preempt a job is determined based on other parameters as shown above The job that has been running for less than the specified period of time is preempted

Case study: Three queues with varying priority

Consider the case where three queues are defined as follows:

- Queue A has the highest relative priority, with a value of 99
- Queue B is both preemptive and preemptable, and has a relative priority of 10
- Queue C has the lowest relative priority, with the default value of 1

The queues can preempt as follows:

- A can preempt B because B is preemptable and B has a lower priority than A
- B can preempt C because B is preemptive and C has a lower priority than B
- A cannot preempt C, even though A has a higher priority than C, because A is not preemptive, nor is C preemptable

Calculation of job slots in use

The number of job slots in use determines whether preemptive jobs can start. The method in which the number of job slots in use is calculated can be configured to ensure that a preemptive job can start. When a job is preempted, it is suspended. If the suspended job still counts towards the total number of jobs allowed in the system, based on the limits imposed in the `lsb.resources` file, suspending the job may not be enough to allow the preemptive job to run.

The `PREEMPT_FOR` parameter is used to change the calculation of job slot usage, ignoring suspended jobs in the calculation. This ensures that if a limit is met, the preempting job can actually run.

Preemption configuration	Effect on the calculation of job slots used
--------------------------	---

Preemption configuration	Effect on the calculation of job slots used
Preemption is not enabled	<ul style="list-style-type: none"> Job slot limits are enforced based on the number of job slots taken by both running and suspended jobs. Job slot limits specified at the queue level are enforced for both running and suspended jobs.
Preemption is enabled	<ul style="list-style-type: none"> The total number of jobs at both the host and individual user level is not limited by the number of suspended jobs; only running jobs are considered. The number of running jobs never exceeds the job slot limits. If starting a preemptive job violates a job slot limit, a lower-priority job is suspended to run the preemptive job. If, however, a job slot limit is still violated (that is, the suspended job still counts in the calculation of job slots in use), the preemptive job still cannot start. Job slot limits specified at the queue level are always enforced for both running and suspended jobs. When preemptive scheduling is enabled, suspended jobs never count against the total job slot limit for individual users.
Preemption is enabled, and PREEMPT_FOR=GROUP_JLP	Only running jobs are counted when calculating the per-processor job slots in use for a user group, and comparing the result with the limit specified at the user level.
Preemption is enabled, and PREEMPT_FOR=GROUP_MAX	Only running jobs are counted when calculating the job slots in use for this user group, and comparing the result with the limit specified at the user level.
Preemption is enabled, and PREEMPT_FOR=HOST_JLU	Only running jobs are counted when calculating the total job slots in use for a user group, and comparing the result with the limit specified at the host level. Suspended jobs do not count against the limit for individual users.
Preemption is enabled, and PREEMPT_FOR=USER_JLP	Only running jobs are counted when calculating the per-processor job slots in use for an individual user, and comparing the result with the limit specified at the user level.

Preemption of backfill jobs

When a high priority queue is configured to run a job with resource or slot reservations and backfill scheduling is enabled (PREEMPT_JOBTYPE=BACKFILL in `lsb.params`), backfill jobs may use the reserved job slots. Configuring a low priority queue with a job to preempt a backfill job may delay the start of a job in a high priority queue with resource or slot reservations. LSF allows this configuration but gives a warning message.

The following example shows the resulting behavior with various queue configurations and priorities.

Queue name	Configuration	Priority	Result
queueR	With a resource or slot reservation	80	Jobs in these queue reserve resources. If backfill scheduling is enabled, backfill jobs with a defined run limit can use the resources.
queueB	As a preemptable backfill queue	50	Jobs in queueB with a defined run limit use job slots reserved by jobs in queueR.
queueP	As a preemptive queue	75	Jobs in this queue do not necessarily have a run limit and may take resources from jobs with reservation.

To guarantee a minimum run time for interruptible backfill jobs, LSF suspends them upon preemption. To change this behavior so that LSF terminates interruptible backfill jobs upon preemption, you must define the parameter `TERMINATE_WHEN=PREEMPT` in `lsb.queues`.

Configuration to modify preemptive scheduling behavior

There are configuration parameters that modify various aspects of preemptive scheduling behavior.

- Modifying the selection of the queue from which to preempt jobs
- Modifying the selection of the job to preempt
- Modifying preemption of backfill and exclusive jobs
- Modifying the way job slot limits are calculated
- Modifying the number of jobs to preempt for a parallel job
- Modifying the control action applied to preempted jobs
- Control how many times a job can be preempted
- Specify a grace period before preemption to improve cluster performance

Configuration to modify selection of queue to preempt

File	Parameter	Syntax and description
lsb.queues	PREEMPTION	PREEMPTION=PREEMPTIVE [<i>low_queue+pref ...</i>] <ul style="list-style-type: none">• Jobs in the queue can preempt running jobs from the specified queues, starting with jobs in the queue with the highest value set for preference
		PREEMPTION=PREEMPTABLE [<i>hi_queue ...</i>] <ul style="list-style-type: none">• Jobs in this queue can be preempted by jobs from the specified queues
	PRIORITY= <i>integer</i>	<ul style="list-style-type: none">• Sets the priority for this queue relative to all other queues• The higher the priority value, the more likely it is that jobs from this queue may preempt jobs from other queues, and the less likely it is for jobs from this queue to be preempted by jobs from other queues

Configuration to modify selection of job to preempt

Files	Parameter	Syntax and description
lsb.params	PREEMPT_FOR	PREEMPT_FOR=LEAST_RUN_TIME <ul style="list-style-type: none">• Preempts the job that has been running for the shortest time
lsb.applications	NO_PREEMPT_RUN_TIME	NO_PREEMPT_RUN_TIME=% <ul style="list-style-type: none">• Prevents preemption of jobs that have been running for the specified percentage of minutes, or longer• If NO_PREEMPT_RUN_TIME is specified as a percentage, the job cannot be preempted after running the percentage of the job duration. For example, if the job run limit is 60 minutes and NO_PREEMPT_RUN_TIME=50%, the job cannot be preempted after it running 30 minutes or longer.• If you specify percentage for NO_PREEMPT_RUN_TIME, requires a run time (bsub -We or RUNTIME in lsb.applications), or run limit to be specified for the job (bsub -W, or RUNLIMIT in lsb.queues, or RUNLIMIT in lsb.applications)

Files	Parameter	Syntax and description
	NO_PREEMPT_FINISH_TIME	NO_PREEMPT_FINISH_TIME=% <ul style="list-style-type: none"> Prevents preemption of jobs that will finish within the specified percentage of minutes. If NO_PREEMPT_FINISH_TIME is specified as a percentage, the job cannot be preempted if the job finishes within the percentage of the job duration. For example, if the job run limit is 60 minutes and NO_PREEMPT_FINISH_TIME=10%, the job cannot be preempted after it running 54 minutes or longer. If you specify percentage for NO_PREEMPT_RUN_TIME, requires a run time (bsub -We or RUNTIME in <i>lsb.applications</i>), or run limit to be specified for the job (bsub -W, or RUNLIMIT in <i>lsb.queue</i>s, or RUNLIMIT in <i>lsb.applications</i>)
lsb.params lsb.queue lsb.applications	MAX_TOTAL_TIME_PREEMPT	MAX_TOTAL_TIME_PREEMPT= <i>minutes</i> <ul style="list-style-type: none"> Prevents preemption of jobs that already have an accumulated preemption time of <i>minutes</i> or greater. The accumulated preemption time is reset in the following cases: <ul style="list-style-type: none"> Job status becomes EXIT or DONE Job is re-queued Job is re-run Job is migrated and restarted MAX_TOTAL_TIME_PREEMPT does not affect preemption triggered by advance reservation or License Scheduler. Accumulated preemption time does not include preemption by advance reservation or License Scheduler.
	NO_PREEMPT_INTERVAL	NO_PREEMPT_INTERVAL= <i>minutes</i> <ul style="list-style-type: none"> Prevents preemption of jobs until after an uninterrupted run time interval of <i>minutes</i> since the job was dispatched or last resumed. NO_PREEMPT_INTERVAL does not affect preemption triggered by advance reservation or License Scheduler.

Configuration to modify preemption of backfill and exclusive jobs

File	Parameter	Syntax and description
lsb.params	PREEMPT_JOBTYPE	PREEMPT_JOBTYPE=BACKFILL <ul style="list-style-type: none"> Enables preemption of backfill jobs. Requires the line PREEMPTION=PREEMPTABLE in the queue definition. Only jobs from queues with a higher priority than queues that define resource or slot reservations can preempt jobs from backfill queues.
		PREEMPT_JOBTYPE=EXCLUSIVE <ul style="list-style-type: none"> Enables preemption of and preemption by exclusive jobs. Requires the line PREEMPTION=PREEMPTABLE or PREEMPTION=PREEMPTIVE in the queue definition. Requires the definition of LSB_DISABLE_LIMLOCK_EXCL in <i>lsf.conf</i>.
		PREEMPT_JOBTYPE=EXCLUSIVE BACKFILL <ul style="list-style-type: none"> Enables preemption of exclusive jobs, backfill jobs, or both.

File	Parameter	Syntax and description
lsf.conf	LSB_DISABLE_LIMLOCK_EXCL	LSB_DISABLE_LIMLOCK_EXCL=y <ul style="list-style-type: none"> Enables preemption of exclusive jobs. For a host running an exclusive job: <ul style="list-style-type: none"> lsload displays the host status ok. bhosts displays the host status closed. Users can run tasks on the host using lsrun or lsgrun. To prevent users from running tasks during execution of an exclusive job, the parameter LSB_DISABLE_LSRUN=y must be defined in lsf.conf. Changing this parameter requires a restart of all sbatchds in the cluster (bctrlid restart sbd). Do not change this parameter while exclusive jobs are running.

Configuration to modify how job slot usage is calculated

File	Parameter	Syntax and description
lsb.params	PREEMPT_FOR	PREEMPT_FOR=GROUP_JLP <ul style="list-style-type: none"> Counts only running jobs when evaluating if a user group is approaching its per-processor job slot limit (SLOTS_PER_PROCESSOR, USERS, and PER_HOST=all in the lsb.resources file), ignoring suspended jobs
		PREEMPT_FOR=GROUP_MAX <ul style="list-style-type: none"> Counts only running jobs when evaluating if a user group is approaching its total job slot limit (SLOTS, PER_USER=all, and HOSTS in the lsb.resources file), ignoring suspended jobs
		PREEMPT_FOR=HOST_JLU <ul style="list-style-type: none"> Counts only running jobs when evaluating if a user or user group is approaching its per-host job slot limit (SLOTS, PER_USER=all, and HOSTS in the lsb.resources file), ignoring suspended jobs
		PREEMPT_FOR=USER_JLP <ul style="list-style-type: none"> Counts only running jobs when evaluating if a user is approaching their per-processor job slot limit (SLOTS_PER_PROCESSOR, USERS, and PER_HOST=all in the lsb.resources file) Ignores suspended jobs when calculating the per-processor job slot limit for individual users

Configuration to modify preemption of parallel jobs

File	Parameter	Syntax and description
lsb.params	PREEMPT_FOR	PREEMPT_FOR=MINI_JOB <ul style="list-style-type: none"> Optimizes preemption of parallel jobs by preempting only enough low-priority parallel jobs to start the high-priority parallel job
		PREEMPT_FOR=OPTIMAL_MINI_JOB <ul style="list-style-type: none"> Optimizes preemption of parallel jobs by preempting only low-priority parallel jobs based on the least number of jobs that will be suspended to allow the high-priority parallel job to start

Configuration to modify the control action applied to preempted jobs

File	Parameter	Syntax and description
lsb.queues	TERMINATE_WHEN	TERMINATE_WHEN=PREEMPT <ul style="list-style-type: none">Changes the default control action of SUSPEND to TERMINATE so that LSF terminates preempted jobs

Configuration to control how many times a job can be preempted

By default, if preemption is enabled, there is actually no guarantee that a job will ever actually complete. A lower priority job could be preempted again and again, and ultimately end up being killed due to a run limit.

Limiting the number of times a job can be preempted is configured cluster-wide (lsb.params), at the queue level (lsb.queues), and at the application level (lsb.applications). MAX_JOB_PREEMPT in lsb.applications overrides lsb.queues, and lsb.queues overrides lsb.params configuration.

Files	Parameter	Syntax and description
lsb.params lsb.queues lsb.applications	MAX_JOB_PREEMPT	MAX_JOB_PREEMPT= <i>integer</i> <ul style="list-style-type: none">Specifies the maximum number of times a job can be preempted.Specify a value within the following ranges: $0 < \text{MAX_JOB_PREEMPT} < \text{INFINIT_INT}$ INFINIT_INT is defined in lsf.hBy default, the number of preemption times is unlimited.

When MAX_JOB_PREEMPT is set, and a job is preempted by higher priority job, the number of job preemption times is set to 1. When the number of preemption times exceeds MAX_JOB_PREEMPT, the job will run to completion and cannot be preempted again.

The job preemption limit times is recovered when LSF is restarted or reconfigured.

If **brequeue** or **bmig** is invoked under a job suspend control (SUSPEND_CONTROL in lsb.applications or JOB_CONTROLS in lsb.queues), the job will be requeued or migrated and the preempted counter reset to 0. To prevent the preempted counter from resetting to 0 under job suspend control, set MAX_JOB_PREEMPT_RESET in lsb.params to N. LSF will not reset the preempted count for MAX_JOB_PREEMPT when the started job is requeued, migrated or rerun.

Configuration of a grace period before preemption

For details, see PREEMPT_DELAY in the file configuration reference.

Files	Parameter	Syntax and description
(in order of precedence) lsb.applications lsb.queues lsb.params	PREEMPT_DELAY	PREEMPT_DELAY= <i>seconds</i> <ul style="list-style-type: none">Specifies the number seconds for a preemptive job in the pending state to wait before a lower-priority job can be preempted.By default, the preemption is immediate.

Preemptive scheduling commands

Commands for submission

Command	Description
bsub -q <i>queue_name</i>	Submits the job to the specified queue, which may have a run limit that is associated with it
bsub -W <i>minutes</i>	Submits the job with the specified run limit, in minutes
bsub -app <i>application_profile_name</i>	Submits the job to the specified application profile, which may have a run limit that is associated with it

Commands to monitor

Command	Description
bjobs -s	Displays suspended jobs, together with the reason the job was suspended

Commands to control

Command	Description
brun	Forces a pending job to run immediately on specified hosts. For an exclusive job, when LSB_DISABLE_LIMLOCK_EXCL=y, LSF allows other jobs already running on the host to finish but does not dispatch any additional jobs to that host until the exclusive job finishes.

Commands to display configuration

Command	Description
bqueues	Displays the priority (PRIO) and run limit (RUNLIMIT) for the queue, and whether the queue is configured to be preemptive, preemptable, or both
bhosts	<ul style="list-style-type: none">Displays the number of job slots per user for a hostDisplays the number of job slots available
bparams	Displays the value of parameters defined in lsb.params.
badmin showconf	<ul style="list-style-type: none">Displays all configured parameters and their values set in lsf.conf or ego.conf that affect mbatchd and sbatchd. Use a text editor to view other parameters in the lsf.conf or ego.conf configuration files.In a multicluster environment, displays the parameters of daemons on the local cluster.

Goal-oriented SLA-driven scheduling

Note: This feature is deprecated and might be removed in a future version of LSF.

- [Using goal-oriented SLA scheduling](#)
- [Service classes for SLA scheduling](#)
Configure service classes in LSB_CONFDIR/cluster_name/configdir/lsb.serviceclasses. Each service class is defined in a **ServiceClass** section.
- [Configuring service classes using the bconf command](#)
Use the **bconf** command to configure service classes without restarting daemons.
- [Time-based service classes](#)
Time-based service classes configure workload based on the number of jobs running at any one time. Goals for deadline, throughput, and velocity of jobs ensure that your jobs are completed on time and reduce the risk of missed deadlines.

- [Submitting jobs to a service class](#)

Using goal-oriented SLA scheduling

Note: This feature is deprecated and might be removed in a future version of LSF.

Goal-oriented SLA scheduling policies help you configure your workload so jobs are completed on time. They enable you to focus on the “what and when” of your projects, not the low-level details of “how” resources need to be allocated to satisfy various workloads.

Service-level agreements in LSF

A service-level agreement (SLA) defines how a service is delivered and the parameters for the delivery of a service. It specifies what a service provider and a service recipient agree to, defining the relationship between the provider and recipient with respect to a number of issues, among them:

- Services to be delivered
- Performance
- Tracking and reporting
- Problem management

An SLA in LSF is a “just-in-time” scheduling policy that defines an agreement between LSF administrators and LSF users. The SLA scheduling policy defines how many jobs should be run from each SLA to meet the configured goals.

Service classes

SLA definitions consist of service-level goals that are expressed in individual service classes. A service class is the actual configured policy that sets the service-level goals for the LSF system. The SLA defines the workload (jobs or other services) and users that need the work done, while the service class that addresses the SLA defines individual goals, and a time window when the service class is active.

Service-level goals can be grouped into two mutually exclusive varieties: guarantee goals which are resource based, and time-based goals which include velocity, throughput, and deadline goals. Time-based goals allow control over the number of jobs running at any one time, while resource-based goals allow control over resource allocation.

Service level goals

You configure the following kinds of goals:

Deadline goals

A specified number of jobs should be completed within a specified time window. For example, run all jobs submitted over a weekend. Deadline goals are time-based.

Velocity goals

Expressed as concurrently running jobs. For example: maintain 10 running jobs between 9:00 a.m. and 5:00 p.m. Velocity goals are well suited for short jobs (run time less than one hour). Such jobs leave the system quickly, and configuring a velocity goal ensures a steady flow of jobs through the system.

Throughput goals

Expressed as number of finished jobs per hour. For example: Finish 15 jobs per hour between the hours of 6:00 PM. and 7:00 AM. Throughput goals are suitable for medium to long running jobs. These jobs stay longer in the system, so you typically want to control their rate of completion rather than their flow.

Combined goals

You might want to set velocity goals to maximize quick work during the day, and set deadline and throughput goals to manage longer running work on nights and over weekends.

How service classes perform goal-oriented scheduling

Goal-oriented scheduling makes use of other, lower level LSF policies like queues and host partitions to satisfy the service-level goal that the service class expresses. The decisions of a service class are considered first before any queue or host partition decisions. Limits are still enforced with respect to lower level scheduling objects like queues, hosts, and users.

Optimum number of running jobs

As jobs are submitted, LSF determines the optimum number of job slots (or concurrently running jobs) needed for the service class to meet its service-level goals. LSF schedules a number of jobs at least equal to the optimum number of slots calculated for the service class.

LSF attempts to meet SLA goals in the most efficient way, using the optimum number of job slots so that other service classes or other types of work in the cluster can still progress. For example, in a service class that defines a deadline goal, LSF spreads out the work over the entire time window for the goal, which avoids blocking other work by not allocating as many slots as possible at the beginning to finish earlier than the deadline.

Submitting jobs to a service class

Use the **bsub -sla service_class_name** to submit a job to a service class for SLA- driven scheduling.

You submit jobs to a service class as you would to a queue, except that a service class is a higher level scheduling policy that makes use of other, lower level LSF policies like queues and host partitions to satisfy the service-level goal that the service class expresses.

For example:

% bsub -W 15 -sla Kyuquot sleep 100

submits the UNIX command sleep together with its argument 100 as a job to the service class named Kyuquot.

The service class name where the job is to run is configured in lsb.serviceclasses. If the SLA does not exist or the user is not a member of the service class, the job is rejected.

Outside of the configured time windows, the SLA is not active and LSF schedules jobs without enforcing any service-level goals. Jobs will flow through queues following queue priorities even if they are submitted with -sla.

Submit with run limit

You should submit your jobs with a run time limit (-W option) or the queue should specify a run time limit (RUNLIMIT in the queue definition in lsb.queues). If you do not specify a run time limit, LSF automatically adjusts the optimum number of running jobs according to the observed run time of finished jobs.

-sla and -g options

You cannot use the -g option with -sla. A job can either be attached to a job group or a service class, but not both.

Modifying SLA jobs (bmod)

Use the -sla option of bmod to modify the service class a job is attached to, or to attach a submitted job to a service class. Use bmod -slan to detach a job from a service class. For example:

% bmod -sla Kyuquot 2307

Attaches job 2307 to the service class Kyuquot.

% bmod -slan 2307

Detaches job 2307 from the service class Kyuquot.

You cannot:

- Use **-sla** with other **bmod** options.
- Modify the service class of jobs already attached to a job group.

Service classes for SLA scheduling

Configure service classes in `LSB_CONFDIR/cluster_name/configdir/lb.serviceclasses`. Each service class is defined in a **ServiceClass** section.

Each service class section begins with the line **Begin ServiceClass** and ends with the line **End ServiceClass**. You must specify:

- A service class name
- At least one goal (deadline, throughput, or velocity) and a time window when the goal is active
- A service class priority

All other parameters are optional. You can configure as many service class sections as you need.

Note: The name you use for your service classes cannot be the same as an existing host partition or user group name.

User groups for service classes

You can control access to the SLA by configuring a user group for the service class. If LSF user groups are specified in `lsb.users`, each user in the group can submit jobs to this service class. If a group contains a subgroup, the service class policy applies to each member in the subgroup recursively. The group can define fair share among its members, and the SLA defined by the service class enforces the fair share policy among the users in the user group configured for the SLA.

Service class priority

A higher value indicates a higher priority, relative to other service classes. Similar to queue priority, service classes access the cluster resources in priority order. LSF schedules jobs from one service class at a time, starting with the highest-priority service class. If multiple service classes have the same priority, LSF runs all the jobs from these service classes in first-come, first-served order.

Service class priority in LSF is completely independent of the UNIX scheduler's priority system for time-sharing processes. In LSF, the `NICE` parameter is used to set the UNIX time-sharing priority for batch jobs.

Any guaranteed resources remaining idle at the end of a scheduling session may be loaned to jobs if loaning is enabled in the guaranteed resource pool (`lsb.resources`).

Service class configuration examples

- The service class `Uclulet` defines one deadline goal that is active during working hours between 8:30 AM and 4:00 PM. All jobs in the service class should complete by the end of the specified time window. Outside of this time window, the SLA is inactive and jobs are scheduled without any goal being enforced:

```
Begin ServiceClass
NAME = Uclulet
PRIORITY = 20
GOALS = [DEADLINE timeWindow (8:30-16:00)]
DESCRIPTION = "working hours"
End ServiceClass
```

- The service class `Nanaimo` defines a deadline goal that is active during the weekends and at nights:

```
Begin ServiceClass
NAME = Nanaimo
PRIORITY = 20
GOALS = [DEADLINE timeWindow (5:18:00-1:8:30 20:00-8:30)]
DESCRIPTION = "weekend nighttime regression tests"
End ServiceClass
```

- The service class `Inuvik` defines a throughput goal of 6 jobs per hour that is always active:

```
Begin ServiceClass
NAME = Inuvik
PRIORITY = 20
GOALS = [THROUGHPUT 6 timeWindow ()]
DESCRIPTION = "constant throughput"
End ServiceClass
```

To configure a time window that is always open, use the `timeWindow` keyword with empty parentheses.

- The service class `Tofino` defines two velocity goals in a 24 hour period. The first goal is to have a maximum of 10 concurrently running jobs during business hours (9:00 AM to 5:00 PM). The second goal is a maximum of 30 concurrently running jobs during off-hours (5:30 PM to 8:30 AM):

```
Begin ServiceClass
NAME = Tofino
PRIORITY = 20
GOALS = [VELOCITY 10 timeWindow (9:00-17:00)] \
        [VELOCITY 30 timeWindow (17:30-8:30)]
DESCRIPTION = "day and night velocity"
End ServiceClass
```

- The service class `Kyuquot` defines a velocity goal that is active during working hours (9:00 AM to 5:30 PM) and a deadline goal that is active during off-hours (5:30 PM to 9:00 AM) Only users `user1` and `user2` can submit jobs to this service class:

```
Begin ServiceClass
NAME = Kyuquot
PRIORITY = 23
GOALS = [VELOCITY 8 timeWindow (9:00-17:30)] \
        [DEADLINE timeWindow (17:30-9:00)]
DESCRIPTION = "Daytime/Nighttime SLA"
End ServiceClass
```

- The service class `Tevere` defines a combination similar to `Kyuquot`, but with a deadline goal that takes effect overnight and on weekends. During the working hours in weekdays the velocity goal favors a mix of short and medium jobs:

```
Begin ServiceClass
NAME = Tevere
PRIORITY = 20
GOALS = [VELOCITY 100 timeWindow (9:00-17:00)] \
        [DEADLINE timeWindow (17:30-8:30 5:17:30-1:8:30)]
DESCRIPTION = "nine to five"
End ServiceClass
```

When an SLA is missing its goal

Use the `CONTROL_ACTION` parameter in your service class to configure an action to be run if the SLA goal is delayed for a specified number of minutes.

```
CONTROL_ACTION=VIOLATION_PERIOD[minutes] CMD [action]
```

If the SLA goal is delayed for longer than `VIOLATION_PERIOD`, the action specified by `CMD` is invoked. The violation period is reset and the action runs again if the SLA is still active when the violation period expires again. If the SLA has multiple active goals that are in violation, the action is run for each of them. For example:

```
CONTROL ACTION=VIOLATION_PERIOD[10] CMD [echo `date` :
SLA is in violation >> !/tmp/sla_violation.log]
```

SLA policies: preemption, chunk jobs and statistics files

- SLA jobs cannot be preempted. You should avoid running jobs belonging to an SLA in low priority queues.
- SLA jobs will not get chunked. You should avoid submitting SLA jobs to a chunk job queue.
- Each active SLA goal generates a statistics file for monitoring and analyzing the system. When the goal becomes inactive the file is no longer updated. The files are created in the `LSB_SHARED/cluster_name/logdir/SLA` directory. Each file name consists of the name of the service class and the goal type.
For example the file named `Quadra.deadline` is created for the deadline goal of the service class name `Quadra`. The following file named `Tofino.velocity` refers to a velocity goal of the service class named `Tofino`:

```
% cat Tofino.velocity
# service class Tofino velocity, NJOBS, NPEND (NRUN + NSSUSP + NUSUSP), (NDONE + NEXIT)
17/9      15:7:34      1063782454 2 0 0 0 0
17/9      15:8:34      1063782514 2 0 0 0 0
17/9      15:9:34      1063782574 2 0 0 0 0
```

```
# service class Tofino velocity, NJOBS, NPEND (NRUN + NSSUSP + NUSUSP), (NDONE + NEXIT)
17/9      15:10:10      1063782610 2 0 0 0 0
```

Configuring service classes using the bconf command

Use the **bconf** command to configure service classes without restarting daemons.

About this task

For a full list of **bconf** command actions for service classes, refer to [bconf](#).

Create a service class

Procedure

Run **bconf create serviceclass=serviceclass_name**.

```
bconf create serviceclass=ClassA "PRIORITY=20 ; GOALS=[DEADLINE timeWindow (8:30-16:00)] ;
USER_GROUP=user1 user2 ugroup1 ; DESCRIPTION=Working hours"
bconf: Request for serviceclass <ClassA> accepted
```

Update service class parameters

Use the **bconf update serviceclass=serviceclass_name** subcommand to replace an old parameter value with a new one.

Procedure

Run **bconf update serviceclass=serviceclass_name**.

```
bconf update serviceclass=ClassA "PRIORITY=10 ; GOALS=[DEADLINE timeWindow (7:00-16:00)]"
bconf: Request for serviceclass <ClassA> accepted
```

Add a service class member

Use the **bconf addmember serviceclass=serviceclass_name** subcommand to add a group member to the specified service class.

Procedure

Run **bconf addmember serviceclass=serviceclass_name**.

```
bconf addmember serviceclass=ClassA "USER_GROUP=user3 ugroup2"
bconf: Request for serviceclass <ClassA> accepted
```

Remove a service class member

Use the **bconf rmmember** subcommand to remove a group member from the specified service class

Procedure

Run **bconf rmmember serviceclass=serviceclass_name**.

```
bconf rmmember serviceclass=ClassA "USER_GROUP=user2"
bconf: Request for serviceclass <ClassA> accepted
```

Time-based service classes

Time-based service classes configure workload based on the number of jobs running at any one time. Goals for deadline, throughput, and velocity of jobs ensure that your jobs are completed on time and reduce the risk of missed deadlines.

Time-based SLA scheduling makes use of other, lower level LSF policies like queues and host partitions to satisfy the service-level goal that the service class expresses. The decisions of a time-based service class are considered first before any queue or host partition decisions. Limits are still enforced with respect to lower level scheduling objects like queues, hosts, and users.

Optimum number of running jobs

As jobs are submitted, LSF determines the optimum number of job slots (or concurrently running jobs) needed for the time-based service class to meet its goals. LSF schedules a number of jobs at least equal to the optimum number of slots that are calculated for the service class.

LSF attempts to meet time-based goals in the most efficient way, using the optimum number of job slots so that other service classes or other types of work in the cluster can still progress. For example, in a time-based service class that defines a deadline goal, LSF spreads out the work over the entire time window for the goal, which avoids blocking other work by not allocating as many slots as possible at the beginning to finish earlier than the deadline.

You should submit time-based SLA jobs with a run time limit at the job level (**-W** option), the application level (RUNLIMIT parameter in the application definition in `lsb.applications`), or the queue level (RUNLIMIT parameter in the queue definition in **lsb.queues**). You can also submit the job with a run time estimate defined at the application level (RUNTIME parameter in `lsb.applications`) instead of or in conjunction with the run time limit.

The following table describes how LSF uses the values that you provide for time-based SLA scheduling.

If you specify...	And...	Then...
A run time limit and a run time estimate	The run time estimate is less than or equal to the run time limit	LSF uses the run time estimate to compute the optimum number of running jobs.
A run time limit	You do not specify a run time estimate, or the estimate is greater than the limit	LSF uses the run time limit to compute the optimum number of running jobs.
A run time estimate	You do not specify a run time limit	LSF uses the run time estimate to compute the optimum number of running jobs.
Neither a run time limit nor a run time estimate		LSF automatically adjusts the optimum number of running jobs according to the observed run time of finished jobs.

Time-based service class priority

A higher value indicates a higher priority, relative to other time-based service classes. Similar to queue priority, time-based service classes access the cluster resources in priority order.

LSF schedules jobs from one time-based service class at a time, starting with the highest-priority service class. If multiple time-based service classes have the same priority, LSF runs the jobs from these service classes in the order the service classes are configured in `lsb.serviceclasses`.

Time-based service class priority in LSF is completely independent of the UNIX scheduler's priority system for time-sharing processes. In LSF, the NICE parameter is used to set the UNIX time-sharing priority for batch jobs.

User groups for time-based service classes

You can control access to time-based SLAs by configuring a user group for the service class. If LSF user groups are specified in `lsb.users`, each user in the group can submit jobs to this service class. If a group contains a subgroup, the service class policy applies to each member in the subgroup recursively. The group can define fair share among its members, and the SLA defined by the service class enforces the fair share policy among the users in the user group configured for the SLA.

By default, all users in the cluster can submit jobs to the service class.

Time-based SLA limitations

Multiclusterc

Multiclusterc does not support time-based SLAs.

Preemption

Time-based SLA jobs cannot be preempted. You should avoid running jobs belonging to an SLA in low priority queues.

Chunk jobs

SLA jobs will not get chunked. You should avoid submitting SLA jobs to a chunk job queue.

Resizable jobs

For resizable job allocation requests, since the job itself has already started to run, LSF bypasses dispatch rate checking and continues scheduling the allocation request.

Time-based SLA statistics files

Each time-based SLA goal generates a statistics file for monitoring and analyzing the system. When the goal becomes inactive the file is no longer updated. Files are created in the LSB_SHAREDIR/cluster_name/logdir/SLA directory. Each file name consists of the name of the service class and the goal type.

For example, the file named Quadra.deadline is created for the deadline goal of the service class name **Quadra**. The following file named Tofino.velocity refers to a velocity goal of the service class named **Tofino**:

```
cat Tofino.velocity
# service class Tofino velocity, NJOBS, NPEND (NRUN + NSSUSP + NUSUSP), (NDONE + NEXIT)
17/9      15:7:34      1063782454 2 0 0 0 0
17/9      15:8:34      1063782514 2 0 0 0 0
17/9      15:9:34      1063782574 2 0 0 0 0
# service class Tofino velocity, NJOBS, NPEND (NRUN + NSSUSP + NUSUSP), (NDONE + NEXIT)
17/9      15:10:10     1063782610 2 0 0 0 0
```

- [Configuring time-based service classes](#)
- [Time-based SLA examples](#)
- [Configuring the SLA CONTROL ACTION parameter \(lsb.serviceclasses\)](#)

Configuring time-based service classes

About this task

Configure time-based service classes in LSB_CONFDIR/cluster_name/configdir/lsb.serviceclasses.

Procedure

Each ServiceClass section begins with the line Begin ServiceClass and ends with the line End ServiceClass. For time-based service classes, you must specify:

- a. A service class name
- b. At least one goal (deadline, throughput, or velocity) and a time window when the goal is active
- c. A service class priority

Other parameters are optional. You can configure as many service class sections as you need.

Important: The name that you use for your service class cannot be the same as an existing host partition or user group name.

Time-based configuration examples

- The service class **Sooke** defines one deadline goal that is active during working hours between 8:30 AM and 4:00 PM. All jobs in the service class should complete by the end of the specified time window. Outside of this time window, the SLA is inactive and jobs are scheduled without any goal being enforced:

```

Begin ServiceClass
NAME = Sooke
PRIORITY = 20
GOALS = [DEADLINE timeWindow (8:30-16:00)]
DESCRIPTION="working hours"
End ServiceClass

```

- The service class `Nanaimo` defines a deadline goal that is active during the weekends and at nights.

```

Begin ServiceClass
NAME = Nanaimo
PRIORITY = 20
GOALS = [DEADLINE timeWindow (5:18:00-1:8:30 20:00-8:30)]
DESCRIPTION="weekend nighttime regression tests"
End ServiceClass

```

- The service class `Sidney` defines a throughput goal of 6 jobs per hour that is always active:

```

Begin ServiceClass
NAME = Sidney
PRIORITY = 20
GOALS = [THROUGHPUT 6 timeWindow ()]
DESCRIPTION="constant throughput"
End ServiceClass

```

Tip: To configure a time window that is always open, use the `timeWindow` keyword with empty parentheses.

- The service class `Tofino` defines two velocity goals in a 24 hour period. The first goal is to have a maximum of 10 concurrently running jobs during business hours (9:00 AM to 5:00 PM). The second goal is a maximum of 30 concurrently running jobs during off-hours (5:30 PM to 8:30 AM)

```

Begin ServiceClass
NAME = Tofino
PRIORITY = 20
GOALS = [VELOCITY 10 timeWindow (9:00-17:00)] \
        [VELOCITY 30 timeWindow (17:30-8:30)]
DESCRIPTION="day and night velocity"
End ServiceClass

```

- The service class `Duncan` defines a velocity goal that is active during working hours (9:00 AM to 5:30 PM) and a deadline goal that is active during off-hours (5:30 PM to 9:00 AM) Only users `user1` and `user2` can submit jobs to this service class.

```

Begin ServiceClass
NAME = Duncan
PRIORITY = 23
USER_GROUP = user1 user2
GOALS = [VELOCITY 8 timeWindow (9:00-17:30)] \
        [DEADLINE timeWindow (17:30-9:00)]
DESCRIPTION="Daytime/Nighttime SLA"
End ServiceClass

```

- The service class `Tevere` defines a combination similar to `Duncan`, but with a deadline goal that takes effect overnight and on weekends. During the working hours in weekdays the velocity goal favors a mix of short and medium jobs.

```

Begin ServiceClass
NAME = Tevere
PRIORITY = 20
GOALS = [VELOCITY 100 timeWindow (9:00-17:00)] \
        [DEADLINE timeWindow (17:30-8:30 5:17:30-1:8:30)]
DESCRIPTION="nine to five" End ServiceClass

```

Time-based SLA examples

A simple deadline goal

The following service class configures an SLA with a simple deadline goal with a half hour time window.

```

Begin ServiceClass
NAME = Quadra
PRIORITY = 20
GOALS = [DEADLINE timeWindow (16:15-16:45)]
DESCRIPTION = short window
End ServiceClass

```

Six jobs submitted with a run time of 5 minutes each will use 1 slot for the half hour time window. **bsla** shows that the deadline can be met:

```

bsla Quadra
SERVICE CLASS NAME: Quadra
-- short window
PRIORITY: 20

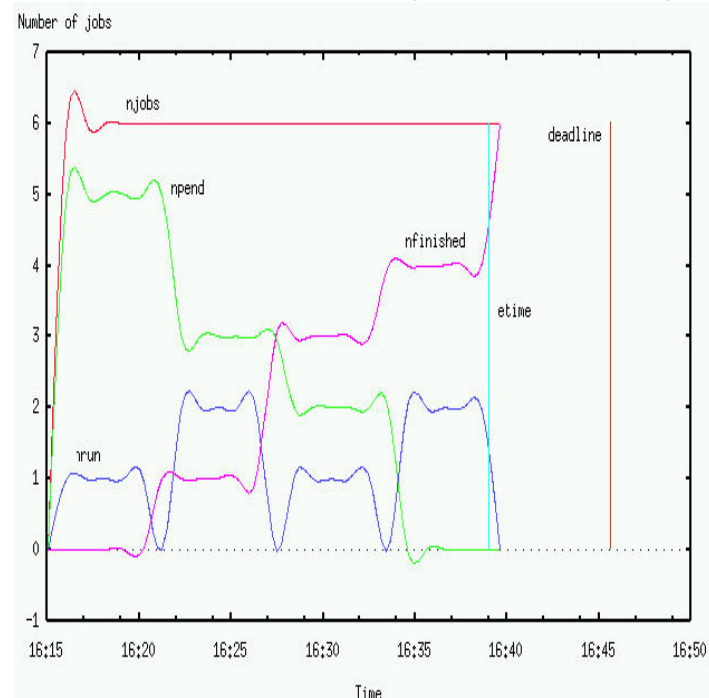
GOAL: DEADLINE
ACTIVE WINDOW: (16:15-16:45)
STATUS: Active:On time
ESTIMATED FINISH TIME: (Wed Jul 2 16:38)
OPTIMUM NUMBER OF RUNNING JOBS: 1

```

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
6	5	1	0	0	0

The following illustrates the progress of the SLA to the deadline. The optimum number of running jobs in the service class (*nrun*) is maintained at a steady rate of 1 job at a time until near the completion of the SLA.

When the finished job curve (*nfinished*) meets the total number of jobs curve (*njobs*) the deadline is met. All jobs are finished well ahead of the actual configured deadline, and the goal of the SLA was met.



An overnight run with two service classes

bsla shows the configuration and status of two service classes *Qualicum* and *Comox*:

- *Qualicum* has a deadline goal with a time window which is active overnight:

```

bsla Qualicum
SERVICE CLASS NAME: Qualicum
PRIORITY: 23

GOAL: VELOCITY 8
ACTIVE WINDOW: (8:00-18:00)
STATUS: Inactive

```

SLA THROUGHPUT: 0.00 JOBS/CLEAN_PERIOD

GOAL: DEADLINE

ACTIVE WINDOW: (18:00-8:00)

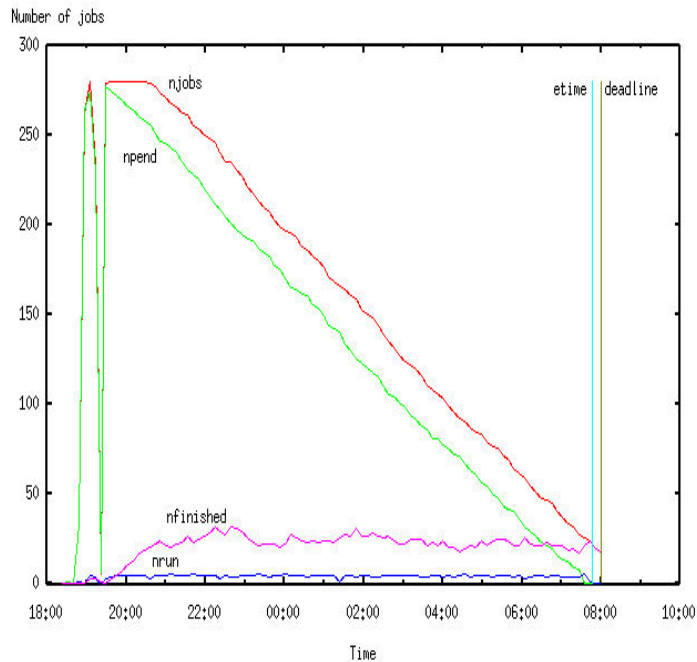
STATUS: Active:On time

ESTIMATED FINISH TIME: (Thu Jul 10 07:53)

OPTIMUM NUMBER OF RUNNING JOBS: 2

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
280	278	2	0	0	0

The following illustrates the progress of the deadline SLA *Qualicum* running 280 jobs overnight with random runtimes until the morning deadline. As with the simple deadline goal example, when the finished job curve (*nfinished*) meets the total number of jobs curve (*njobs*) the deadline is met with all jobs completed ahead of the configured deadline.



- *Comox* has a velocity goal of 2 concurrently running jobs that is always active:

bsla *Comox*

SERVICE CLASS NAME: *Comox*

PRIORITY: 20

GOAL: VELOCITY 2

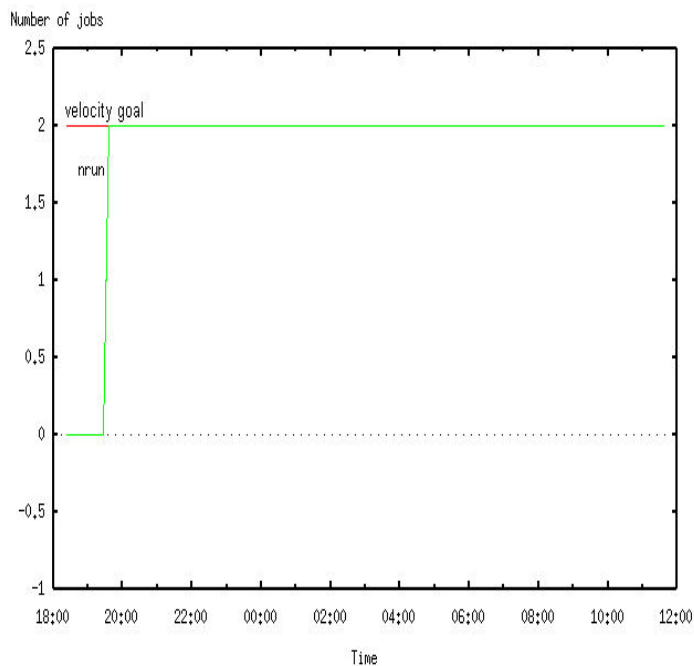
ACTIVE WINDOW: Always Open

STATUS: Active:On time

SLA THROUGHPUT: 2.00 JOBS/CLEAN_PERIOD

NJOBS	PEND	RUN	SSUSP	USUSP	FINISH
100	98	2	0	0	0

The following illustrates the progress of the velocity SLA *Comox* running 100 jobs with random runtimes over a 14 hour period.



Configuring the SLA CONTROL_ACTION parameter (lsb.serviceclasses)

About this task

Configure a specific action to occur when a time-based SLA is missing its goal.

Procedure

Use the CONTROL_ACTION parameter in your service class to configure an action to be run if the time-based SLA goal is delayed for a specified number of minutes.

```
CONTROL_ACTION=VIOLATION_PERIOD[minutes] CMD [action]
```

If the SLA goal is delayed for longer than VIOLATION_PERIOD, the action specified by CMD is invoked. The violation period is reset and the action runs again if the SLA is still active when the violation period expires again. If the time-based SLA has multiple active goals that are in violation, the action is run for each of them.

Example

```
CONTROL ACTION=VIOLATION_PERIOD[10] CMD [echo `date` :  
SLA is in violation >> !/tmp/sla_violation.log]
```

Submitting jobs to a service class

About this task

The service class name where the job is to run is configured in lsb.serviceclasses. If the SLA does not exist or the user is not a member of the service class, the job is rejected.

If the SLA is not active or the guarantee SLA has used all guaranteed resources, LSF schedules jobs without enforcing any service-level goals. Jobs will flow through queues following queue priorities even if they are submitted with **-sla**, and will not make use of any guaranteed resources.

Procedure

Run **bsub -sla service_class_name** to submit a job to a service class for SLA-driven scheduling.

```
bsub -W 15 -sla Duncan sleep 100
```

submits the UNIX command **sleep** together with its argument 100 as a job to the service class named `Duncan`.

- [Modifying SLA jobs \(bmod\)](#)

Modifying SLA jobs (bmod)

Procedure

Run **bmod -sla** to modify the service class a job is attached to, or to attach a submitted job to a service class. Run **bmod -slan** to detach a job from a service class:

```
bmod -sla Duncan 2307
```

Attaches job 2307 to the service class `Duncan`.

```
bmod -slan 2307
```

Detaches job 2307 from the service class `Duncan`.

For all SLAs, you cannot:

- Use **-sla** with other **bmod** options
- Modify the service class of jobs that are already attached to a job group

For time-based SLAs, you cannot:

- Move job array elements from one service class to another, only entire job arrays

Global resources

Global resources are resources that are shared between all connected clusters.

You can configure global resources to be either static or dynamic. Dynamic global resources use a global resource ELIM (called GRES) to dynamically collect global resources.

LSF handles global resources in the same way as local resources. For resource requirement strings, specify global resources for simple, compound, and alternative resource requirement strings in the same way as local resources. In addition, specify global resources for the `usage`, `select`, `span`, `same`, `cu`, `order`, `duration`, and `decay` keywords in the resource requirement in the same way as local resources.

Global resources cannot have the same name as a local resource in the global policy daemon (GPD) cluster and local limits do not apply to global resources.

- [Global resource collection](#)
LSF collects information for static global resources from the configuration in the `lsb.globalpolicies` file, and collects information for dynamic global resources from the global resource collection daemon (called GRES).
- [Configuring global resources](#)
Define global resources in the `lsb.globalpolicies` file.

- [Using and displaying global resources](#)

Specify global resources in the same way as local resources. Use the **bgpinfo** command to display information on global resources.

Global resource collection

LSF collects information for static global resources from the configuration in the `lsb.globalpolicies` file, and collects information for dynamic global resources from the global resource collection daemon (called GRES).

The global resource collection daemon is named **gres**, but has the same syntax and usage as the local ELIMs. The management GRES is the MGRES. The prefix for a GRES executable file is **gres.**, for example **gres.myscript**.

The **gres** and **mgres** executable files are in the `$LSF_SERVERDIR` directory in the GPD cluster. The **gpolicyd** daemon starts, controls, and collects dynamic global resource information from the MGRES. The MGRES and GRES instances only start up in the GPD cluster.

Configuring global resources

Define global resources in the `lsb.globalpolicies` file.

Procedure

1. Edit the `lsb.globalpolicies` file to define your global resources.
2. Use the Resource section to define your global resources.

The syntax for global resources is the same as the syntax specifying local resources (that is, the Resource section of the `lsf.shared` file), with some restrictions.

- a. Specify a name, resource type, and description for the resource using the `RESOURCENAME`, `TYPE`, and `DESCRIPTION` keywords.

RESOURCENAME

Resource names are case sensitive and can be up to 39 characters in length, with the following restrictions:

- Cannot begin with a number
- Cannot contain the following special characters

```
: . ( ) [ + - * / ! & | < > @ = ,
```

- Cannot be any of the following reserved names:

```
cpu cpuf io logins ls idle maxmem maxswp maxtmp type model
status it mem ncpus nprocs ncores nthreads
define_ncpus_cores define_ncpus_procs define_ncpus_threads
ndisks pg r15m r15s r1m swap swp tmp ut local
dchost jobvm
```

- Cannot begin with `inf` or `nan` (uppercase or lowercase). Use `infxx` or `nanxx` instead if required.
- For Solaris machines, the keyword `int` is reserved and cannot be used.

If the global resource has the same name as a local resource in the global policy daemon (GPD) cluster, LSF ignores the global resources and logs a warning message.

TYPE

For global resources, the resource type must be `Numeric`.

DESCRIPTION

The information here describes the meaning of the resource.

- b. Optional. Specify optional attributes for the resource.

INTERVAL

For dynamic resources, set the update interval in seconds.

INCREASING

Set to `Y` to indicate that a higher value indicates a greater load, or set to `N` to indicate that a higher value indicates a lower load.

CONSUMABLE

Set to `Y` since all global resources are numeric resources.

RELEASE

Set to `Y` since all global resources are numeric resources.

The following example specifies a global static resource and two global dynamic resources:

```
Begin Resource
RESOURCENAME      TYPE      INTERVAL INCREASING CONSUMABLE RELEASE DESCRIPTION #
Keywords
global_res_static  Numeric  ()      N      Y      Y      (Global static
resources)
global_res_dynamic Numeric  60      N      Y      Y      (Global dynamic
resources)
global_res_dynamic2 Numeric  30      N      Y      Y      (Global dynamic
resources)
End Resource
```

3. Use the ResourceMap section to define the mapping between the global resources and their sharing clusters.
The ResourceMap section specifies the global resources that are shared to all clusters. For static resources, you must also define an initial value for the resource.

RESOURCENAME

Specify the name of the resource. This resource must have a corresponding entry in the Resource section.

LOCATION

[resource_value@][all]

For a static resource, specify an initial value and the at symbol (@). Do not specify an initial value for a dynamic resource. Specify [all] to share the global resource with all connected clusters.

The following example shares the global resources with all connected clusters, and specifies an initial value of 5 for the static resource.

```
Begin ResourceMap
RESOURCENAME      LOCATION
global_res_static (5@[all])
global_res_dynamic ([all])
global_res_dynamic2 ([all])
End ResourceMap
```

4. Use the DistributePolicy sections to define the distribution policy for the global resources and the global limits.
All clusters must use the same policy for global resources and the same policy for global limits.

NAME

Specify the name of the distribution policy. This name is any ASCII string that is 40 characters or less. You can use letters, digits, underscores (_) or dashes (-). You cannot use blank spaces.

DISTRIBUTE

To distribute global resources, specify `resource`. To distribute global limits, specify `limit`.

POLICY

Specify `evenly` for LSF to divide global resources evenly among all the connected clusters.

The default policy is `compete`, which has each connected cluster compete to use the global resource. Under the compete distribution policy, each cluster can get all the available amount of global resources during each scheduling cycle. LSF reverts the allocation if there is any conflicts for different clusters. This policy might have negative effects on the scheduling efficiency if there are several conflicts, but has better performance than the evenly policy under most circumstances.

The following example enables LSF to use the evenly distribution policy for both global resources and global limits.

```
Begin DistributePolicy
NAME=Resource_Even_Policy
DISTRIBUTE=resource
```

```

POLICY=evenly
End DistributePolicy

Begin DistributePolicy
NAME=Limit_Even_Policy
DISTRIBUTE=limit
POLICY=evenly
End DistributePolicy

```

5. Use the `ReservationUsage` section to specify the global resources to reserve and to define the method of resource reservation.

RESOURCE

Specify the name of the resource. This resource must have a corresponding entry in the Resource section.

METHOD

Specify the resource reservation method: `PER_JOB`, `PER_TASK`, or `PER_HOST`.

RESERVE

Reserves the resource for pending jobs that are waiting for another resource to become available.

The following example specifies the three resources, each with a different resource reservation method.

```

Begin ReservationUsage
RESOURCE          METHOD          RESERVE
global_res_static PER_HOST       N
global_res_dynamic PER_TASK       Y
global_res_dynamic2 PER_JOB       N
End ReservationUsage

```

Using and displaying global resources

Specify global resources in the same way as local resources. Use the **bgpinfo** command to display information on global resources.

Procedure

1. For resource requirement strings, specify global resources for simple, compound, and alternative resource requirement strings in the same way as local resources.
Specify global resources for the `rusage`, `select`, `span`, `same`, `cu`, `order`, `duration`, and `decay` keywords in the resource requirement in the same way as local resources. Global resources can be static or dynamic resources, but must be numeric, consumable, and releasable.
For example, if the resource reservation method of the `global_res_dynamic` resource is defined as a `PER_TASK` in the `lsb.globalpolicies` file:

```

Begin ReservationUsage
RESOURCE          METHOD          RESERVE
global_res_dynamic PER_TASK       Y
End ReservationUsage

```

The following example reserves the `global_res_dynamic` resource for all 4 job tasks instead of only 1 on the host where the job runs:

```

bsub -n 4 -R "global_res_dynamic > 0 rusage[my_resource=1]" myjob

```

2. Use the **bgpinfo** command to display information on global resources.
 - a. Use the **bgpinfo resource** subcommand to display the following information on global resources: resource usage, resource type, and reservation type.
 - b. Use the **bgpinfo policy** subcommand to display the distribution policies for global resources and global limits.
3. Use the **bhosts -s** command option to display global resources (and other shared numeric resources) that are on the hosts.
The **bhosts -s** command option, which displays information about shared numeric resources on the hosts, also displays information on global resources, since global resources are numeric and consumable.

4. If you are using LSF License Scheduler, the **blstat -cl** command option, which shows license tokens in the specified cluster, also shows license tokens that are shared as global resources.

GPU resources

Learn how to configure and use GPU resources for your LSF jobs.

- NVIDIA GPU resources are supported on x64 and IBM Power LE (Little Endian) platforms on Linux.
- AMD GPU resources are supported on x64 platforms on Linux.

For an example of how to use LSF with GPU resources on NVIDIA DGX systems, refer to [IBM Spectrum LSF with NVIDIA DGX systems](#).

- [Enabling GPU features](#)
Learn how to enable GPU features in IBM Spectrum LSF.
- [Monitoring GPU resources](#)
Learn how to monitor GPU resources in IBM Spectrum LSF.
- [Submitting and monitoring GPU jobs](#)
Learn how to submit and monitor jobs that use GPU resources in IBM Spectrum LSF.
- [Legacy GPU features using ELIM](#)
Learn how to manually use the legacy ELIM to use GPU features in IBM Spectrum LSF.

Enabling GPU features

Learn how to enable GPU features in IBM® Spectrum LSF.

- [Automatic GPU configuration](#)
LSF automatically configures GPU resources, which allows your LSF jobs to use GPU resources by default.
- [Enabling jobs to use GPU resources](#)
LSF jobs can specify GPU resource requirements in one statement.
- [Optimizing GPU resource metric collection](#)
LSF optimizes resource metric collection to improve LSF performance.
- [Nvidia Data Center GPU Manager \(DCGM\) features](#)
The Nvidia Data Center GPU Manager (DCGM) is a suite of data center management tools that allow you to manage and monitor GPU resources in an accelerated data center.
- [GPU access enforcement](#)
LSF can enforce GPU access on systems that support the Linux cgroup devices subsystem. To enable GPU access through Linux cgroups, configure the `LSB_RESOURCE_ENFORCE="gpu"` parameter in the `lsf.conf` file. LSF creates devices to contain job processes if the job has GPU resource requirements so that the job processes cannot escape from the allocated GPUs. Each GPU job device includes only the GPUs that LSF distributes. Linux cgroup devices are only created for GPU jobs.
- [Decreasing GPU power consumption when a GPU is not in use](#)
A GPU consumes significant power even when it idles. LSF provides configuration parameters to decrease the GPU power that is consumed if GPU is not in use within a specified time. By default, LSF does not power off a GPU even when it is idle.
- [Nvidia Multi-Instance GPU \(MIG\) features](#)
Use the `LSF_MANAGE_MIG` parameter in the `lsf.conf` file to enable dynamic MIG scheduling.

Automatic GPU configuration

LSF automatically configures GPU resources, which allows your LSF jobs to use GPU resources by default.

Note: Automatic GPU configuration is now enabled and fixed at `LSF_GPU_AUTOCONFIG=Y`.

The **lsload -l** command option does not show GPU metrics. However, they are shown in the **lsload -gpu**, **lsload -gpuload**, and **lshosts -gpu** command options. Scheduler-related GPU resources are shown in the command output.

Related reference

- [LSF_GPU_AUTOCONFIG](#)
- [LSF_HOST_MAX_GPU](#)

Enabling jobs to use GPU resources

LSF jobs can specify GPU resource requirements in one statement.

Note: Extended GPU resource requirement specification is now enabled and fixed at `LSF_GPU_NEW_SYNTAX=extend`. You can specify all GPU requirements for your job together with the **bsub -gpu** option or in configuration in a queue, application profile, or in a default GPU requirement. The resource requirements of your job submission cannot use the legacy GPU resources (`ngpus_shared`, `ngpus_excl_t`, `ngpus_excl_p`) as job resource requirements. In addition, if the `PREEMPTABLE_RESOURCES` parameter in the `lsb.params` file includes the `ngpus_physical` resource, GPU preemption is enabled with only one restriction: higher priority GPU jobs cannot preempt GPU jobs with `mode=shared` configuration in the GPU resource requirements if there are multiple jobs running on the GPU. (Note that as of Fix Pack 14, this restriction has been removed so that higher priority GPU jobs with `j_exclusive=yes` or `mode=exclusive_process` settings can preempt shared-mode GPU jobs if there were multiple jobs running on the GPU.) Ensure that you properly configure the `MIG`, `RERUNNABLE`, or `REQUEUE` parameters to ensure that GPU resources are properly released after the job is preempted.

If any option of the GPU requirements is not defined, the default value is used for each option:

"num=1:mode=shared:mps=no:j_exclusive=no". Use the `LSB_GPU_REQ` parameter in the `lsf.conf` file to specify a different default GPU resource requirement.

You can also specify GPU resource requirements with the `GPU_REQ` parameter in a queue (`lsb.queues` file) or application profile (`lsb.applications` file).

If a GPU requirement is specified at the cluster level (`lsf.conf` file), queue, or application profile, and at job level, each option (`num`, `mode`, `mps`, and `j_exclusive`) of the GPU requirement is merged separately. Job level overrides application level, which overrides queue level, which overrides cluster level configuration. For example, if the `mode` option of GPU requirement is defined on the `-gpu` option, and the `mps` option is defined in the queue, the `mode` of job level and the `mps` value of queue is used.

Optimizing GPU resource metric collection

LSF optimizes resource metric collection to improve LSF performance.

Note: Optimized GPU resource metric collection is now enabled and fixed at `LSF_GPU_RESOURCE_IGNORE=Y`.

The **mbatchd** does not collect GPU resource information from the management host LIM. This means that the **lsload -s**, **lsload -l**, and **bhosts -l** commands, which display LSF resources, no longer display information about GPU resources. That is, this option does not display `gpu_<num>n` resources. This improves LSF response time because there are fewer LSF resources to manage and display.

In addition, all built-in GPU resources (`gpu_<num>n`) are completely removed from the management host LIM. LSF uses a different method for the management host LIM and server host LIMs to collect GPU information. This further improves performance by having fewer built-in LSF resources.

Note:

1. If you are using LSF RTM, Version 10.2 Fix Pack 11 or later, you can configure `LSF_GPU_RESOURCE_IGNORE` to either `Y` or `N`.
2. If you are using LSF RTM, Version 10.2 Fix Pack 1 or earlier, configure `LSF_GPU_RESOURCE_IGNORE` to `N`.

Nvidia Data Center GPU Manager (DCGM) features

The Nvidia Data Center GPU Manager (DCGM) is a suite of data center management tools that allow you to manage and monitor GPU resources in an accelerated data center.

LSF integrates with Nvidia DCGM to work more effectively with GPUs in the LSF cluster. DCGM provides additional functionality when working with jobs that request GPU resources by:

- providing GPU usage information for EXCLUSIVE_PROCESS mode jobs.
- checking the status of GPUs to automatically filter out unhealthy GPUs when the job allocates GPU resources. This ensures that jobs are running on healthy GPUs. DCGM provides mechanisms to check the GPU health and LSF integrates these mechanisms to check the GPU status before, during, and after the job is running to meet the GPU requirements. If the execution host's DCGM status is not valid, the **bjobs -l** command shows an error message. The job still runs, but GPU resource usage reports are not available from that host.
- automatically adding back any previously-unhealthy GPUs that are healthy again so that these GPUs are available for job allocation.
- synchronizing the GPU auto-boost feature to support jobs that run across multiple GPUs, including jobs that run across multiple GPUs on a single host.

Enable the DCGM integration by defining the LSF_DCGM_PORT parameter in the lsf.conf file. After enabling the parameter, you must start up DCGM to use the features.

Note: If the DCGM integration does not work as expected due to a missing libdcgm.so file, create a softlink to ensure that the libdcgm.so file exists and is accessible:

```
sudo ln -s /usr/lib64/libdcgm.so.1 /usr/lib64/libdcgm.so
```

Run the -gpu option with the **bjobs**, **bhist**, and **bacct** commands to display GPU usage information from DCGM after the job finishes. The -gpu option must be used with the following command options:

- For the **bjobs** command, you must run the -gpu option with the -l or -UF options.
- For the **bhist** command, you must run the -gpu option with the -l option.
- For the **bacct** command, you must run the -gpu option with the -l option.

GPU access enforcement

LSF can enforce GPU access on systems that support the Linux cgroup devices subsystem. To enable GPU access through Linux cgroups, configure the LSB_RESOURCE_ENFORCE="gpu" parameter in the lsf.conf file. LSF creates devices to contain job processes if the job has GPU resource requirements so that the job processes cannot escape from the allocated GPUs. Each GPU job device includes only the GPUs that LSF distributes. Linux cgroup devices are only created for GPU jobs.

GPU enforcement for Linux cgroup device subsystems is supported on Red Hat Enterprise Linux 6.2 and later, and SuSe Linux Enterprise Linux 11 SP2 and later.

Note:

- GPU enforcement is not supported on AMD GPUs.
- When GPU enforcement is enabled, the GPUs that are contained in one device cgroup are reallocated new GPU IDs, beginning with 0. CUDA Version 7.0 or later supports cgroup completely.

Jobs can specify how job processes are to be bound to these computing elements. LSF uses the environment variable CUDA_VISIBLE_DEVICES to tell user applications which GPUs are allocated. It is possible for user applications to escape from the allocated GPUs by changing the CUDA_VISIBLE_DEVICES variable to use other GPUs directly.

For example, the following command submits a job with one exclusive thread GPU requirement:

```
bsub -R "rusage[ngpus_excl_t=1]" ./myapp
```


LSF creates a device that contains one exclusive thread GPU and attaches the process ID of the application `./myapp` to this device. The device serves as a strict container for job processes so that the application `./myapp` cannot use other GPUs.

Decreasing GPU power consumption when a GPU is not in use

A GPU consumes significant power even when it idles. LSF provides configuration parameters to decrease the GPU power that is consumed if GPU is not in use within a specified time. By default, LSF does not power off a GPU even when it is idle.

Set the `LSB_GPU_POWEROFF_DURATION` parameter in the `lsf.conf` file to specify the minimum number of seconds before LSF can power off an idle GPU. When the `LSB_GPU_POWEROFF_DURATION` parameter is set, LSF tries to allocate the GPU that is not running in "MIN power limit" mode. If not enough GPUs are in "MAX power limit" mode, LSF allocates the GPUs that are in "MIN power limit" mode and switches those GPUs to run in "MAX power limit" mode.

If the `LSB_GPU_POWEROFF_DURATION=0` parameter is set, LSF powers off GPUs immediately after the job finishes.

LSF uses the following criteria to allocate the GPU flow:

- All GPUs are in the same PCI.
- Check whether the "MAX power limit" mode GPUs meets job requirements. If they do, LSF does not allocate the "MIN power limit" mode GPUs first. If they do not meet the requirements, LSF allocates all the GPUs to the job, including both "MAX power limit" and "MIN power limit" mode GPUs.

If the **sbatchd** daemon is restarted, the GPU idle time is recalculated.

NVIDIA K80 hardware supports switch power limits. The NVML library must be Version 6.340 or newer.

Nvidia Multi-Instance GPU (MIG) features

Use the `LSF_MANAGE_MIG` parameter in the `lsf.conf` file to enable dynamic MIG scheduling.

Nvidia Multi-Instance GPU (MIG) features allow a single supported GPU to be securely partitioned into up to seven independent GPU instances, providing multiple users with independent GPU resources.

When dynamic MIG scheduling is enabled, LSF dynamically creates GPU instances (GI) and compute instances (CI) on each host, and LSF controls the MIG of each host. If you enable dynamic MIG scheduling, do not manually create or destroy MIG devices outside of LSF. Set the `LSF_MANAGE_MIG` parameter to `Y` in the `lsf.conf` file to enable dynamic MIG scheduling.

Starting in Fix Pack 14, additionally, LSF leverages cgroups to enforce MIG device isolation. Set this enforcement by configuring `LSB_RESOURCE_ENFORCE="gpu"` in the `lsf.conf` file.

If `LSF_MANAGE_MIG` is set to `N` or is undefined, LSF uses static MIG scheduling. LSF allocates the GI and CI based on the configuration of each MIG host, and dispatches jobs to the MIG hosts. LSF does not create or destroy the GI and CI on the MIG hosts. If you use static MIG scheduling and want to change MIG devices, you must wait for the running MIG job to finish, then destroy the existing MIG device, create a new MIG device, and restart the LSF daemons.

If you change the value of this parameter, you must wait until all MIG jobs that are running on a cluster are done, then restart the LSF daemons for your changes take effect.

After changing the value of this parameter, you must restart the LSF daemons for your changes to take effect.

Related reference

- [LSF_MANAGE_MIG](#)
- [LSB_RESOURCE_ENFORCE](#)

Monitoring GPU resources

Learn how to monitor GPU resources in IBM® Spectrum LSF.

- [Monitor GPU resources with lsload command](#)
Options within the **lsload** command show the host-based and GPU-based GPU information for a cluster.
- [Monitor GPU resources with lshosts command](#)
Options within the **lshosts** command show the GPU topology information for a cluster.

Monitor GPU resources with lsload command

Options within the **lsload** command show the host-based and GPU-based GPU information for a cluster.

lsload -gpu

The `-gpu` option for **lsload** shows the host-based GPU information:

```
% lsload -gpu
HOST_NAME  status  ngpus  gpu_shared_avg_mut  gpu_shared_avg_ut  ngpus_prohibited  ngpus_physical
hosta      ok      4      30%                 35%                 1                  2
hostc      ok      4      30%                 35%                 1
```

Options within the **lshosts** command show the GPU topology information for a cluster.

lshosts -gpu

The `-gpu` option for `lshosts` shows the GPU topology information for a cluster:

```
% lshosts -gpu
HOST_NAME  gpu_id  gpu_model      gpu_driver  gpu_factor  numa_id  vendor
hosta      0  TeskaK20c      384.10      70          0        Nvidia
            1  TeskaK20c      384.10      70          0        Nvidia
            2  TeskaK20c      384.10      70          1        Nvidia
            3  TeskaK20c      384.10      70          1        Nvidia
hostb      0  TeskaK80c      384.10      90          0        Nvidia
            1  TeskaK80c      384.10      90          0        Nvidia
            2  TeskaK20c      384.10      70          1        Nvidia
            3  TeskaK20c      384.10      70          1        Nvidia
hostc      0  Vega10_RadeonIn 5.6.12      -           0        AMD
            1  Vega10_RadeonIn 5.6.12      -           0        AMD
```

Related reference

- [lshosts](#)

Submitting and monitoring GPU jobs

Learn how to submit and monitor jobs that use GPU resources in IBM® Spectrum LSF.

- [Configuring GPU resource requirements](#)
To configure GPU resource requirements, define the `GPU_REQ` parameter in the `lsb.applications` file for the application profile, the `GPU_REQ` parameter in the `lsb.queues` file for the queue, or the `LSB_GPU_REQ` parameter in the `lsf.conf` file for the cluster. For complex GPU resource requirements (including alternative or compound resource requirements), use the `RES_REQ` parameter in the `lsb.applications` or `lsb.queues` file.
- [Submitting jobs that require GPU resources](#)
Use the `bsub -gpu` option to specify GPU resource requirements during job submission or submit your job to a queue or application profile that has GPU resource requirements configured in the `GPU_REQ` parameter. For complex GPU resource requirements (including alternative or compound resource requirements), use the `bsub -R` option.
- [Monitoring GPU jobs](#)
For jobs submitted with the default GPU requirements (with the option `-gpu -`), use the `bjobs -l` command to see the default job-level resource requirement without details like `<num=1...>: Requested GPU`.
- [Example GPU job submissions](#)
The following are examples of possible submissions for jobs that use GPU resources.

Configuring GPU resource requirements

To configure GPU resource requirements, define the `GPU_REQ` parameter in the `lsb.applications` file for the application profile, the `GPU_REQ` parameter in the `lsb.queues` file for the queue, or the `LSB_GPU_REQ` parameter in the `lsf.conf` file for the cluster. For complex GPU resource requirements (including alternative or compound resource requirements), use the `RES_REQ` parameter in the `lsb.applications` or `lsb.queues` file.

About this task

The GPU resource requirements string has the following syntax in the `lsb.applications` and `lsb.queues` files:

```
GPU_REQ = "[num=num_gpus[/task | host]] [:mode=shared | exclusive_process] [:mps=yes[,shared][,nocvd]] no |
per_socket[,shared][,nocvd] | per_gpu[,shared][,nocvd] [:aff=yes | no] [:j_exclusive=yes | no][:block=yes | no] [:glink=yes]
[:gpack=yes | no] [:gvendor=amd | nvidia] [:gmodel=model_name[#mem_size]] [:gtile=tile_num|''] [:gmem=mem_value]"
```

The GPU resource requirements string has the following syntax in the lsf.conf file:

```
LSB_GPU_REQ = "[num=num_gpus[/task | host]] [:mode=shared | exclusive_process] [:mps=yes[,shared][,nocvd]] no |  
per_socket[,shared][,nocvd] | per_gpu[,shared][,nocvd] [:aff=yes | no] [:j_exclusive=yes | no][:block=yes | no] [:gpack=yes | no]  
[:gvendor=amd | nvidia]"
```

num=num_gpus[/task | host]

The number of physical GPUs required by the job. By default, the number is per host. You can also specify that the number is per task by specifying /task after the number.

If you specified that the number is per task, the configuration of the ngpus_physical resource in the lsb.resources file is set to PER_TASK, or the RESOURCE_RESERVE_PER_TASK=Y parameter is set in the lsb.params file, this number is the requested count per task.

mode=shared | exclusive_process

The GPU mode when the job is running, either shared or exclusive_process. The default mode is shared.

The shared mode corresponds to the Nvidia or AMD DEFAULT compute mode. The exclusive_process mode corresponds to the Nvidia EXCLUSIVE_PROCESS compute mode.

Note: Do not specify exclusive_process when you are using AMD GPUs (that is, when gvendor=amd is specified).

mps=yes[,nocvd][,shared] | per_socket[,shared][,nocvd] | per_gpu[,shared][,nocvd] | no

Enables or disables the Nvidia Multi-Process Service (MPS) for the GPUs that are allocated to the job. Using MPS effectively causes the EXCLUSIVE_PROCESS mode to behave like the DEFAULT mode for all MPS clients. MPS always allows multiple clients to use the GPU through the MPS server.

Note: To avoid inconsistent behavior, do not enable mps when you are using AMD GPUs (that is, when gvendor=amd is specified). If the result of merging the GPU requirements at the cluster, queue, application, and job levels is gvendor=amd and mps is enabled (for example, if gvendor=amd is specified at the job level without specifying mps=no, but mps=yes is specified at the application, queue, or cluster level), LSF ignores the mps requirement.

MPS is useful for both shared and exclusive process GPUs, and allows more efficient sharing of GPU resources and better GPU utilization. See the Nvidia documentation for more information and limitations.

When using MPS, use the EXCLUSIVE_PROCESS mode to ensure that only a single MPS server is using the GPU, which provides additional insurance that the MPS server is the single point of arbitration between all CUDA process for that GPU.

You can also enable MPS daemon sharing by adding the share keyword with a comma and no space (for example, mps=yes,shared enables MPS daemon sharing on the host). If sharing is enabled, all jobs that are submitted by the same user with the same resource requirements share the same MPS daemon on the host, socket, or GPU.

LSF starts MPS daemons on a per-host, per-socket, or per-GPU basis depending on value of the mps keyword:

- If mps=yes is set, LSF starts one MPS daemon per host per job.
When share is enabled (that is, if mps=yes,shared is set), LSF starts one MPS daemon per host for all jobs that are submitted by the same user with the same resource requirements. These jobs all use the same MPS daemon on the host.

When the CUDA_VISIBLE_DEVICES environment variable is disabled (that is, if mps=yes,nocvd is set), LSF does not set the CUDA_VISIBLE_DEVICES<number> environment variables for tasks, so LSF MPI does not set CUDA_VISIBLE_DEVICES for the tasks. LSF just sets the CUDA_VISIBLE_DEVICES<number> environment variables for tasks, not CUDA_VISIBLE_DEVICES. LSF MPI converts the CUDA_VISIBLE_DEVICES<number> environment variables into CUDA_VISIBLE_DEVICES and sets that for the tasks.

- If mps=per_socket is set, LSF starts one MPS daemon per socket per job. When enabled with share (that is, if mps=per_socket,shared is set), LSF starts one MPS daemon per socket for all jobs that are submitted by the same user with the same resource requirements. These jobs all use the same MPS daemon for the socket.
- If mps=per_gpu is set, LSF starts one MPS daemon per GPU per job. When enabled with share (that is, if mps=per_gpu,shared is set), LSF starts one MPS daemon per GPU for all jobs that are submitted by the same user with the same resource requirements. These jobs all use the same MPS daemon for the GPU.

Important: Using EXCLUSIVE_THREAD mode with MPS is not supported and might cause unexpected behavior.

j_exclusive=yes | no

Specifies whether the allocated GPUs can be used by other jobs. When the mode is set to exclusive_process, the

j_exclusive=yes option is set automatically.

aff=yes | no

Specifies whether to enforce strict GPU-CPU affinity binding. If set to `no`, LSF relaxes GPU affinity while maintaining CPU affinity. By default, `aff=yes` is set to maintain strict GPU-CPU affinity binding.

Note: The `aff=yes` setting conflicts with `block=yes` (distribute allocated GPUs as blocks when the number of tasks is greater than the requested number of GPUs). This is because strict CPU-GPU binding allocates GPUs to tasks based on the CPU NUMA ID, which conflicts with the distribution of allocated GPUs as blocks. If `aff=yes` and `block=yes` are both specified in the GPU requirements string, the `block=yes` setting takes precedence and strict CPU-GPU affinity binding is disabled (that is, `aff=no` is automatically set).

`block=yes | no`

Specifies whether to enable block distribution, that is, to distribute the allocated GPUs of a job as blocks when the number of tasks is greater than the requested number of GPUs. If set to `yes`, LSF distributes all the allocated GPUs of a job as blocks when the number of tasks is bigger than the requested number of GPUs. By default, `block=no` is set so that allocated GPUs are not distributed as blocks.

For example, if a GPU job requests to run on a host with 4 GPUs and 40 tasks, block distribution assigns GPU0 for ranks 0-9, GPU1 for ranks 10-19, GPU2 for tanks 20-29, and GPU3 for ranks 30-39.

Note: The `block=yes` setting conflicts with `aff=yes` (strict CPU-GPU affinity binding). This is because strict CPU-GPU binding allocates GPUs to tasks based on the CPU NUMA ID, which conflicts with the distribution of allocated GPUs as blocks. If `block=yes` and `aff=yes` are both specified in the GPU requirements string, the `block=yes` setting takes precedence and strict CPU-GPU affinity binding is disabled (that is, `aff=no` is automatically set).

`gpack=yes | no`

For shared mode jobs only. Specifies whether to enable pack scheduling. If set to `yes`, LSF packs multiple shared mode GPU jobs to allocated GPUs. LSF schedules shared mode GPUs as follows:

1. LSF sorts the candidate hosts (from largest to smallest) based on the number of shared GPUs that already have running jobs, then by the number of GPUs that are not exclusive.
If the `order[]` keyword is defined in the resource requirements string, after sorting `order[]`, LSF re-sorts the candidate hosts by the `gpack` policy (by shared GPUs that already have running jobs first, then by the number of GPUs that are not exclusive). The `gpack` policy sort priority is higher than the `order[]` sort.
2. LSF sorts the candidate GPUs on each host (from largest to smallest) based on the number of running jobs.

After scheduling, the shared mode GPU job packs to the allocated shared GPU that is sorted first, not to a new shared GPU.

If Docker attribute affinity is enabled, the order of candidate hosts are sorted by Docker attribute affinity before sorting by GPUs.

By default, `gpack=no` is set so that pack scheduling is disabled.

`gvendor=amd | nvidia`

Specifies the GPU vendor type. LSF allocates GPUs with the specified vendor type.

Specify `amd` to request AMD GPUs, or specify `nvidia` to request Nvidia GPUs.

By default, LSF requests Nvidia GPUs.

`gmodel=model_name[-mem_size]`

Specifies GPUs with the specific model name and, optionally, its total GPU memory. By default, LSF allocates the GPUs with the same model, if available.

The `gmodel` keyword supports the following formats:

`gmodel=model_name`

Requests GPUs with the specified brand and model name (for example, `TeslaK80`).

`gmodel=short_model_name`

Requests GPUs with a specific brand name (for example, `Tesla`, `Quadro`, `NVS`,) or model type name (for example, `K80`, `P100`).

`gmodel=model_name-mem_size`

Requests GPUs with the specified brand name and total GPU memory size. The GPU memory size consists of the number and its unit, which includes `M`, `G`, `T`, `MB`, `GB`, and `TB` (for example, `12G`).

To find the available GPU model names on each host, run the **`lsload -gpuload`**, **`lshosts -gpu`**, or **`bhosts -gpu`** commands. The model name string does not contain space characters. In addition, the slash (/) and hyphen (-) characters are replaced with the underscore character (_). For example, the GPU model name “`Tesla C2050 / C2070`” is converted to “`TeslaC2050_C2070`” in LSF.

`gmem=mem_value`

Specify the GPU memory on each GPU required by the job. The format of *mem_value* is the same to other resource value (for example, `mem` or `swap`) in the `rusage` section of the job resource requirements (-R).

`gtile=! | tile_num`

Specifies the number of GPUs per socket. Specify an number to explicitly define the number of GPUs per socket on the host, or specify an exclamation mark (!) to enable LSF to automatically calculate the number, which evenly divides the GPUs along all sockets on the host. LSF guarantees the `gtile` requirements even for affinity jobs. This means that LSF might not allocate the GPU's affinity to the allocated CPUs when the `gtile` requirements cannot be satisfied.

If the `gtile` keyword is not specified for an affinity job, LSF attempts to allocate enough GPUs on the sockets that allocated GPUs. If there are not enough GPUs on the optimal sockets, jobs cannot go to this host.

If the `gtile` keyword is not specified for a non-affinity job, LSF attempts to allocate enough GPUs on the same socket. If this is not available, LSF might allocate GPUs on separate GPUs.

`nvlink=yes`

Obsolete in LSF, Version 10.1 Fix Pack 11. Use the `glink` keyword instead. Enables the job enforcement for NVLink connections among GPUs. LSF allocates GPUs with NVLink connections in force.

`glink=yes`

Enables job enforcement for special connections among GPUs. LSF must allocate GPUs with the special connections that are specific to the GPU vendor.

If the job requests AMD GPUs, LSF must allocate GPUs with the xGMI connection. If the job requests Nvidia GPUs, LSF must allocate GPUs with the NVLink connection.

Do not use `glink` together with the obsolete `nvlink` keyword.

By default, LSF can allocate GPUs without special connections when there are not enough GPUs with these connections.

`mig=GI_size[/CI_size]`

Specifies Nvidia Multi-Instance GPU (MIG) device requirements.

Specify the requested number of GPU instances for the MIG job. Valid GPU instance sizes are 1, 2, 3, 4, 7.

Optionally, specify the requested number of compute instances after the specified GPU instance size and a slash character (/). The requested compute instance size must be less than or equal to the requested GPU instance size. In addition, Nvidia MIG does not support the following GPU/compute instance size combinations: 4/3, 7/5, 7/6. If this is not specified, the default compute instance size is 1.

If the `GPU_REQ_MERGE` parameter is defined as `y` or `y` in the `lsb.params` file and a GPU requirement is specified at multiple levels (at least two of the default cluster, queue, application profile, or job level requirements), each option of the GPU requirement is merged separately. Job level overrides application level, which overrides queue level, which overrides the default cluster GPU requirement. For example, if the `mode` option of the GPU requirement is defined on the `-gpu` option, and the `mps` option is defined in the queue, the `mode` of job level and the `mps` value of queue is used.

If the `GPU_REQ_MERGE` parameter is not defined as `y` or `y` in the `lsb.params` file and a GPU requirement is specified at multiple levels (at least two of the default cluster, queue, application profile, or job level requirements), the entire GPU requirement string is replaced. The entire job level GPU requirement string overrides application level, which overrides queue level, which overrides the default GPU requirement.

The GPU requirements are converted to `rusage` resource requirements for the job. For example, `num=2` is converted to `rusage[ngpus_physical=2]`. Use the **`bjobs`**, **`bhist`**, and **`bacct`** commands to see the merged resource requirement.

For more details on specifying the `GPU_REQ` parameter in the `lsb.applications` and `lsb.queues` files, or on specifying the `LSB_GPU_REQ` parameter in the `lsf.conf` file refer to [configuration reference](#).

Procedure

- To configure GPU resource requirements for an application profile, specify the `GPU_REQ` parameter in the `lsb.applications` file.
`GPU_REQ="gpu_req"`
- To configure GPU resource requirements for a queue, specify the `GPU_REQ` parameter in the `lsb.queues` file.
`GPU_REQ="gpu_req"`

- To configure default GPU resource requirements for the cluster, specify the `LSB_GPU_REQ` parameter in the `lsf.conf` file.
`LSB_GPU_REQ="gpu_req"`

Submitting jobs that require GPU resources

Use the `bsub -gpu` option to specify GPU resource requirements during job submission or submit your job to a queue or application profile that has GPU resource requirements configured in the `GPU_REQ` parameter. For complex GPU resource requirements (including alternative or compound resource requirements), use the `bsub -R` option.

Procedure

- Use the **`bsub -gpu`** option to submit a job with GPU resource requirements.

```
bsub -gpu - | "[num=num_gpus[/task | host]] [:mode=shared | exclusive_process] [:mps=yes[,shared][,nocvd]] | no |
per_socket[,shared][,nocvd] | per_gpu[,shared][,nocvd]] [:j_exclusive=yes | no] [:aff=yes | no] [:block=yes | no]
[:gpack=yes | no] [:glink=yes] [:gvendor=amd | nvidia] [:gmodel=model_name[-mem_size]] [:gtile=tile_num['!']]
[:gmem=mem_value]"
```

-

Specifies that the job does not set job-level GPU requirements. Use the hyphen with no letter to set the effective GPU requirements, which are defined at the cluster, queue, or application profile level.

If a GPU requirement is specified at the cluster, queue, and application profile level, each option (`num`, `mode`, `mps`, `j_exclusive`, `gmodel`, `gtile`, `gmem`, and `nvlink`) of the GPU requirement is merged separately. Application profile level overrides queue level, which overrides the cluster level default GPU requirement.

If there are no GPU requirements defined in the cluster, queue, or application level, the default value is `"num=1:mode=shared:mps=no:j_exclusive=no"`.

`num=num_gpus[/task | host]`

The number of physical GPUs required by the job. By default, the number is per host. You can also specify that the number is per task by specifying `/task` after the number.

If you specified that the number is per task, the configuration of the `ngpus_physical` resource in the `lsb.resources` file is set to `PER_TASK`, or the `RESOURCE_RESERVE_PER_TASK=Y` parameter is set in the `lsb.params` file, this number is the requested count per task.

`mode=shared | exclusive_process`

The GPU mode when the job is running, either `shared` or `exclusive_process`. The default mode is `shared`.

The `shared` mode corresponds to the Nvidia or AMD `DEFAULT` compute mode. The `exclusive_process` mode corresponds to the Nvidia `EXCLUSIVE_PROCESS` compute mode.

Note: Do not specify `exclusive_process` when you are using AMD GPUs (that is, when `gvendor=amd` is specified).

`mps=yes[,nocvd][,shared] | per_socket[,shared][,nocvd] | per_gpu[,shared][,nocvd] | no`

Enables or disables the Nvidia Multi-Process Service (MPS) for the GPUs that are allocated to the job. Using MPS effectively causes the `EXCLUSIVE_PROCESS` mode to behave like the `DEFAULT` mode for all MPS clients. MPS always allows multiple clients to use the GPU through the MPS server.

Note: To avoid inconsistent behavior, do not enable `mps` when you are using AMD GPUs (that is, when `gvendor=amd` is specified). If the result of merging the GPU requirements at the cluster, queue, application, and job levels is `gvendor=amd` and `mps` is enabled (for example, if `gvendor=amd` is specified at the job level without specifying `mps=no`, but `mps=yes` is specified at the application, queue, or cluster level), LSF ignores the `mps` requirement.

MPS is useful for both `shared` and `exclusive_process` GPUs, and allows more efficient sharing of GPU resources and better GPU utilization. See the Nvidia documentation for more information and limitations.

When using MPS, use the `EXCLUSIVE_PROCESS` mode to ensure that only a single MPS server is using the GPU, which provides additional insurance that the MPS server is the single point of arbitration between all CUDA process for that GPU.

You can also enable MPS daemon sharing by adding the `share` keyword with a comma and no space (for example, `mps=yes,shared` enables MPS daemon sharing on the host). If sharing is enabled, all jobs that are submitted by the same user with the same resource requirements share the same MPS daemon on the host, socket, or GPU.

LSF starts MPS daemons on a per-host, per-socket, or per-GPU basis depending on value of the mps keyword:

- If `mps=yes` is set, LSF starts one MPS daemon per host per job.
When share is enabled (that is, if `mps=yes`, `shared` is set), LSF starts one MPS daemon per host for all jobs that are submitted by the same user with the same resource requirements. These jobs all use the same MPS daemon on the host.

When the `CUDA_VISIBLE_DEVICES` environment variable is disabled (that is, if `mps=yes`, `nocvd` is set), LSF does not set the `CUDA_VISIBLE_DEVICES<number>` environment variables for tasks, so LSF MPI does not set `CUDA_VISIBLE_DEVICES` for the tasks. LSF just sets the `CUDA_VISIBLE_DEVICES<number>` environment variables for tasks, not `CUDA_VISIBLE_DEVICES`. LSF MPI converts the `CUDA_VISIBLE_DEVICES<number>` environment variables into `CUDA_VISIBLE_DEVICES` and sets that for the tasks.
- If `mps=per_socket` is set, LSF starts one MPS daemon per socket per job. When enabled with share (that is, if `mps=per_socket`, `shared` is set), LSF starts one MPS daemon per socket for all jobs that are submitted by the same user with the same resource requirements. These jobs all use the same MPS daemon for the socket.
- If `mps=per_gpu` is set, LSF starts one MPS daemon per GPU per job. When enabled with share (that is, if `mps=per_gpu`, `shared` is set), LSF starts one MPS daemon per GPU for all jobs that are submitted by the same user with the same resource requirements. These jobs all use the same MPS daemon for the GPU.

Important: Using `EXCLUSIVE_THREAD` mode with MPS is not supported and might cause unexpected behavior.

`j_exclusive=yes | no`

Specifies whether the allocated GPUs can be used by other jobs. When the mode is set to `exclusive_process`, the `j_exclusive=yes` option is set automatically.

`aff=yes | no`

Specifies whether to enforce strict GPU-CPU affinity binding. If set to `no`, LSF relaxes GPU affinity while maintaining CPU affinity. By default, `aff=yes` is set to maintain strict GPU-CPU affinity binding.

Note: The `aff=yes` setting conflicts with `block=yes` (distribute allocated GPUs as blocks when the number of tasks is greater than the requested number of GPUs). This is because strict CPU-GPU binding allocates GPUs to tasks based on the CPU NUMA ID, which conflicts with the distribution of allocated GPUs as blocks. If `aff=yes` and `block=yes` are both specified in the GPU requirements string, the `block=yes` setting takes precedence and strict CPU-GPU affinity binding is disabled (that is, `aff=no` is automatically set).

`block=yes | no`

Specifies whether to enable block distribution, that is, to distribute the allocated GPUs of a job as blocks when the number of tasks is greater than the requested number of GPUs. If set to `yes`, LSF distributes all the allocated GPUs of a job as blocks when the number of tasks is bigger than the requested number of GPUs. By default, `block=no` is set so that allocated GPUs are not distributed as blocks.

For example, if a GPU job requests to run on a host with 4 GPUs and 40 tasks, block distribution assigns GPU0 for ranks 0-9, GPU1 for ranks 10-19, GPU2 for tanks 20-29, and GPU3 for ranks 30-39.

Note: The `block=yes` setting conflicts with `aff=yes` (strict CPU-GPU affinity binding). This is because strict CPU-GPU binding allocates GPUs to tasks based on the CPU NUMA ID, which conflicts with the distribution of allocated GPUs as blocks. If `block=yes` and `aff=yes` are both specified in the GPU requirements string, the `block=yes` setting takes precedence and strict CPU-GPU affinity binding is disabled (that is, `aff=no` is automatically set).

`gpack=yes | no`

For shared mode jobs only. Specifies whether to enable pack scheduling. If set to `yes`, LSF packs multiple shared mode GPU jobs to allocated GPUs. LSF schedules shared mode GPUs as follows:

1. LSF sorts the candidate hosts (from largest to smallest) based on the number of shared GPUs that already have running jobs, then by the number of GPUs that are not exclusive.
If the `order[]` keyword is defined in the resource requirements string, after sorting order[], LSF re-sorts the candidate hosts by the gpack policy (by shared GPUs that already have running jobs first, then by the number of GPUs that are not exclusive). The gpack policy sort priority is higher than the order[] sort.

2. LSF sorts the candidate GPUs on each host (from largest to smallest) based on the number of running jobs. After scheduling, the shared mode GPU job packs to the allocated shared GPU that is sorted first, not to a new shared GPU.

If Docker attribute affinity is enabled, the order of candidate hosts are sorted by Docker attribute affinity before sorting by GPUs.

By default, `gpac=no` is set so that pack scheduling is disabled.

`gvendor=amd | nvidia`

Specifies the GPU vendor type. LSF allocates GPUs with the specified vendor type.

Specify `amd` to request AMD GPUs, or specify `nvidia` to request Nvidia GPUs.

By default, LSF requests Nvidia GPUs.

`gmodel=model_name[-mem_size]`

Specifies GPUs with the specific model name and, optionally, its total GPU memory. By default, LSF allocates the GPUs with the same model, if available.

The `gmodel` keyword supports the following formats:

`gmodel=model_name`

Requests GPUs with the specified brand and model name (for example, `TeslaK80`).

`gmodel=short_model_name`

Requests GPUs with a specific brand name (for example, `Tesla`, `Quadro`, `NVS`,) or model type name (for example, `K80`, `P100`).

`gmodel=model_name-mem_size`

Requests GPUs with the specified brand name and total GPU memory size. The GPU memory size consists of the number and its unit, which includes `M`, `G`, `T`, `MB`, `GB`, and `TB` (for example, `12G`).

To find the available GPU model names on each host, run the **`lsload -gpuload`**, **`lshosts -gpu`**, or **`bhosts -gpu`** commands. The model name string does not contain space characters. In addition, the slash (/) and hyphen (-) characters are replaced with the underscore character (_). For example, the GPU model name “`Tesla C2050 / C2070`” is converted to “`TeslaC2050_C2070`” in LSF.

`gmem=mem_value`

Specify the GPU memory on each GPU required by the job. The format of `mem_value` is the same to other resource value (for example, `mem` or `swap`) in the `usage` section of the job resource requirements (-R).

`gtile=! | tile_num`

Specifies the number of GPUs per socket. Specify an number to explicitly define the number of GPUs per socket on the host, or specify an exclamation mark (!) to enable LSF to automatically calculate the number, which evenly divides the GPUs along all sockets on the host. LSF guarantees the `gtile` requirements even for affinity jobs. This means that LSF might not allocate the GPU's affinity to the allocated CPUs when the `gtile` requirements cannot be satisfied.

If the `gtile` keyword is not specified for an affinity job, LSF attempts to allocate enough GPUs on the sockets that allocated GPUs. If there are not enough GPUs on the optimal sockets, jobs cannot go to this host.

If the `gtile` keyword is not specified for a non-affinity job, LSF attempts to allocate enough GPUs on the same socket. If this is not available, LSF might allocate GPUs on separate GPUs.

`nvlink=yes`

Obsolete in LSF, Version 10.1 Fix Pack 11. Use the `glink` keyword instead. Enables the job enforcement for NVLink connections among GPUs. LSF allocates GPUs with NVLink connections in force.

`glink=yes`

Enables job enforcement for special connections among GPUs. LSF must allocate GPUs with the special connections that are specific to the GPU vendor.

If the job requests AMD GPUs, LSF must allocate GPUs with the xGMI connection. If the job requests Nvidia GPUs, LSF must allocate GPUs with the NVLink connection.

Do not use `glink` together with the obsolete `nvlink` keyword.

By default, LSF can allocate GPUs without special connections when there are not enough GPUs with these connections.

`mig=GI_size[/CI_size]`

Specifies Nvidia Multi-Instance GPU (MIG) device requirements.

Specify the requested number of GPU instances for the MIG job. Valid GPU instance sizes are 1, 2, 3, 4, 7.

Optionally, specify the requested number of compute instances after the specified GPU instance size and a slash character (/). The requested compute instance size must be less than or equal to the requested GPU instance

size. In addition, Nvidia MIG does not support the following GPU/compute instance size combinations: 4/3, 7/5, 7/6. If this is not specified, the default compute instance size is 1.

If the GPU_REQ_MERGE parameter is defined as `Y` or `y` in the `lsb.params` file and a GPU requirement is specified at multiple levels (at least two of the default cluster, queue, application profile, or job level requirements), each option of the GPU requirement is merged separately. Job level overrides application level, which overrides queue level, which overrides the default cluster GPU requirement. For example, if the mode option of the GPU requirement is defined on the `-gpu` option, and the mps option is defined in the queue, the mode of job level and the mps value of queue is used.

If the GPU_REQ_MERGE parameter is not defined as `Y` or `y` in the `lsb.params` file and a GPU requirement is specified at multiple levels (at least two of the default cluster, queue, application profile, or job level requirements), the entire GPU requirement string is replaced. The entire job level GPU requirement string overrides application level, which overrides queue level, which overrides the default GPU requirement.

The **esub** parameter `LSB_SUB4_GPU_REQ` modifies the value of the `-gpu` option.

The GPU requirements are converted to rusage resource requirements for the job. For example, `num=2` is converted to `rusage[ngpus_physical=2]`. Use the **bjobs**, **bhist**, and **bacct** commands to see the merged resource requirement.

For more details on using the **bsub -gpu** command option, refer to [bsub-gpu](#).

For example, the following command requests 2 Tesla GPUs with a total size of 12 GB on each GPU, reserves 8 GB GPU memory on each GPU, and evenly spreads the GPUs over all sockets on the host. The job submission enables NVIDIA MPS on the GPUs, and any allocated GPUs must have NVLink connections:

```
bsub -gpu "num=2:gmodel=Tesla-12G:gmem=8G:gtile='!':mps=yes:glink=nvlink ./myjob
```

- If you have GPU resource requirements specified in an application profile or queue, submit a job to the application profile or queue with GPU resource requirements.

```
bsub -app gpu_app ./myjob
```



```
bsub -q "gpu_queue" ./myjob
```
- Use the **bsub -R** command option to submit a job with complex GPU resource requirements.
There might be complex GPU resource requirements that the **bsub -gpu** option and GPU_REQ parameter syntax cannot cover, including compound GPU requirements (for different GPU requirements for jobs on different hosts, or for different parts of a parallel job) and alternate GPU requirements (if more than one set of GPU requirements might be acceptable for a job to run).

The following **bsub -R** options are specific for GPU resource requirements:

```
bsub -R "span[gtile=tile_num | '!'] rusage[ngpus_physical=num_gpus:gmodel=model_name[-mem_size]:gmem=mem_value:glink=yes]"
```

- In the `span[]` section, use the `gtile` keyword to specify the number of GPUs requested on each socket.
- In the `rusage[]` section, use the `ngpus_physical` resource to request the number of physical GPUs, together with the `gmodel` option specify the GPU model (and the optional total memory size), the `gmem` option to specify the amount of reserved GPU memory, and the `glink` option to request only GPUs with special connections (xGMI connections for AMD GPUs or NVLink connections for Nvidia GPUs).

If you need to use the `mode`, `j_exclusive`, and `mps` options, you must use simple GPU requirements. You cannot use these options in the `rusage[]` section for complex GPU resource requirements.

Important: If you specify complex GPU resource requirements at the job level with the **bsub -R** command option, the queue level with the `RES_REQ` parameter in the `lsb.queues` file, or the application profile level with the `RES_REQ` parameter in the `lsb.applications` file, LSF ignores the simple GPU requirements at all levels (that is, LSF ignores the **bsub -gpu** command option, the `LSB_GPU_REQ` parameter in the `lsf.conf` file, and `GPU_REQ` parameter in the `lsb.queues` and `lsb.applications` files).

For more details on specifying resource requirements, including complex GPU resource requirements, see [Specifying resource requirements](#).

For example, the following alternative resource requirement string specifies a parallel job with 2 tasks, with each task running on a separate host that either has 4 K80 GPUs or 2 P100 GPUs evenly divided on each socket:

```
bsub -n 2 -R "span[ptile=1,gtile=!] rusage[ngpus_physical=4:gmodel=K80 || ngpus_physical=2:gmodel=P100]" ./myjob
```

Results

The order of GPU conditions when allocating the GPUs are as follows:

1. The largest GPU compute capability (gpu_factor value).
2. GPUs with direct NVLink connections.
3. GPUs with the same model, including the GPU total memory size.
4. The largest available GPU memory.
5. The number of concurrent jobs on the same GPU.
6. The current GPU mode.

Monitoring GPU jobs

For jobs submitted with the default GPU requirements (with the option -gpu -), use the **bjobs -l** command to see the default job-level resource requirement without details like <num=1...>: Requested GPU.

About this task

If the -gpu option specifies GPU requirements (for example, -gpu num=3, the **bjobs -l** shows the details as Requested GPU <num=3>.

The **bjobs -l** command displays an output section for GPU jobs that shows the combined and effective GPU requirements that are specified for the job. The GPU requirement string is in the format of the GPU_REQ parameter in the application profile or queue.

- The combined GPU requirement is merged based on the current GPU requirements in job, queue, application, or cluster levels for a job.
- The effective GPU requirement is the one used by a started job. It never changes after a job is started.

For example,

```
bjobs -l
Job <101>, User <user1>, Project <default>, Status <RUN>, Queue <normal>, Co
mmmand <sleep 10000>, Share group charged </user1>
Wed Jul 12 04:51:00: Submitted from host <hosta>, CWD </home/user1/,
Specified Hosts <hosta>, Requested GPU;
...
...
```

EXTERNAL MESSAGES:

MSG_ID	FROM	POST_TIME	MESSAGE	ATTACHMENT
0	user1	Jul 12 04:51	hosta:gpus=2,0,1;	N

RESOURCE REQUIREMENT DETAILS:

```
Combined: select[type == any] order[r15s:pg] rusage[ngpus_physical=2.00]
Effective: select[type == any] order[r15s:pg] rusage[ngpus_physical=3.00]
```

GPU REQUIREMENT DETAILS:

```
Combined: num=2:mode=shared:mps=no:j_exclusive=no
Effective: num=3:mode=shared:mps=no:j_exclusive=no
```

Use the **bhist -l** command to see the effective GPU requirements string for a GPU allocation, for example,

```
bhist -l
Job <204>, User <user1>, Project <default>, Command <blaunch sleep 60>
Wed Jul 12 22:40:54: Submitted from host <hosta>, to Queue <normal>, CWD </
scratch/user1>, 8 Task(s), Requested Resources <span[ptile=4]
.....rusage[ngpus_physical=4]>, Specified Hosts <haswell05>,
<hosta!>, Requested GPU <num=4:mode=shared:j_exclusive=yes>;
Wed Jul 12 22:40:55: Dispatched 8 Task(s) on Host(s) <hosta> <hosta> <h
hosta> <hosta> <hostb> <hostb> <hostb>
<hostb>, Allocated 8 Slot(s) on Host(s) <hosta> <h
hosta> <hosta> <hosta> <hostb> <hostb>
```

```

<hostb> <hostb>, Effective RES_REQ <select[type ==
any] order[r15s:pg] rusage[ngpus_physical=4.00] span[ptil
e=4] >;
Wed Jul 12 22:40:56: Starting (Pid 116194);
Wed Jul 12 22:40:56: External Message "hostb:gpus=0,3,1,2;haswell103:gpus=0,1,2,3;
EFFECTIVE GPU REQ: num=4:mode=shared:mps=no:j_exclusive=yes;"
was posted from "user1" to message box 0;

```

Example GPU job submissions

The following are examples of possible submissions for jobs that use GPU resources.

- The following job requests the default GPU resource requirement `num=1:mode=shared:mps=no:j_exclusive=no`. The job requests one GPU in `DEFAULT` mode, without starting MPS, and the GPU can be used by other jobs since `j_exclusive` is set to `no`.

```
bsub -gpu - ./app
```
- The following job requires 2 `EXCLUSIVE_PROCESS` mode GPUs and starts MPS before the job runs:

```
bsub -gpu "num=2:mode=exclusive_process:mps=yes" ./app
```
- The following job requires 2 `EXCLUSIVE_PROCESS` mode GPUs, starts MPS before the job runs, and allows multiple jobs in the host to share the same MPS daemon if those jobs are submitted by the same user with the same GPU requirements:

```
bsub -gpu "num=2:mode=exclusive_process:mps=yes,share" ./app
```
- The following job requires 2 `EXCLUSIVE_PROCESS` mode GPUs and starts multiple MPS daemons (one MPS daemon per socket):

```
bsub -gpu "num=2:mode=exclusive_process:mps=per_socket" ./app
```
- The following job requires 2 `EXCLUSIVE_PROCESS` mode GPUs and starts multiple MPS daemons (one MPS daemon per socket), and allows multiple jobs in the socket to share the same MPS daemon if those jobs are submitted by the same user with the same GPU requirements:

```
bsub -gpu "num=2:mode=exclusive_process:mps=per_socket,share" ./app
```
- The following job requires 2 `EXCLUSIVE_PROCESS` mode GPUs and starts multiple MPS daemons (one MPS daemon per GPU):

```
bsub -gpu "num=2:mode=exclusive_process:mps=per_gpu" ./app
```
- The following job requires 2 `EXCLUSIVE_PROCESS` mode GPUs and starts multiple MPS daemons (one MPS daemon per GPU), and allows multiple jobs in the GPU to share the same MPS daemon if those jobs are submitted by the same user with the same GPU requirements:

```
bsub -gpu "num=2:mode=exclusive_process:mps=per_gpu,share" ./app
```
- The following job requires 2 `DEFAULT` mode GPUs and uses them exclusively. The two GPUs cannot be used by other jobs even though the mode is shared:

```
bsub -gpu "num=2:mode=shared:j_exclusive=yes" ./app
```
- The following job uses 3 `DEFAULT` mode GPUs and shares them with other jobs:

```
bsub -gpu "num=3:mode=shared:j_exclusive=no" ./app
```
- The following job requests 2 AMD GPUs:

```
bsub -gpu "num=2:gvendor=amd" ./app
```
- The following job requests 2 Vega GPUs with xGMI connections:

```
bsub -gpu "num=2:gmodel=Vega:glink=yes" ./app
```

- The following job requests 2 Nvidia GPUs:

```
bsub -gpu "num=2:gvendor=nvidia" ./app
```

- The following job requests 2 Tesla C2050/C2070 GPUs:

```
bsub -gpu "num=2:gmodel=C2050_C2070"
```

- The following job requests 2 Tesla GPUs of any model with a total memory size of 12 GB on each GPU:

```
bsub -gpu "num=2:gmodel=Tesla-12G"
```

- The following job requests 2 Tesla GPUs of any model with a total memory size of 12 GB on each GPU, but with relaxed GPU affinity enforcement:

```
bsub -gpu "num=2:gmodel=Tesla-12G":aff=no
```

- The following job requests 2 Tesla GPUs of any model with a total memory size of 12 GB on each GPU and reserves 8 GB of GPU memory on each GPU:

```
bsub -gpu "num=2:gmodel=Tesla-12G:gmem=8G"
```

- The following job requests 4 Tesla K80 GPUs per host and 2 GPUs on each socket:

```
bsub -gpu "num=4:gmodel=K80:gtile=2"
```

- The following job requests 4 Tesla K80 GPUs per host and the GPUs are spread evenly on each socket:

```
bsub -gpu "num=4:gmodel=K80:gtile='!'"
```

- The following job requests 4 Tesla P100 GPUs per host with NVLink connections and the GPUs are spread evenly on each socket:

```
bsub -gpu "num=4:gmodel=TeslaP100:gtile='!':glink=yes"
```

- The following job uses 2 Nvidia MIG devices with a GPU instance size of 3 and a compute instance size of 2.

```
bsub -gpu "num=2:mig=3/2" ./app
```

- The following job uses 4 `EXCLUSIVE_PROCESS` GPUs that cannot be used by other jobs. The `j_exclusive` option defaults to yes for this job.

```
bsub -gpu "num=4:mode=exclusive_process" ./app
```

- The following job requires two tasks. Each task requires 2 `EXCLUSIVE_PROCESS` GPUs on two hosts. The GPUs are allocated in the same NUMA as the allocated CPU.

```
bsub -gpu "num=2:mode=exclusive_process" -n2 -R "span[ptile=1] affinity[core(1)]" ./app
```

- The following job ignores the simple GPU resource requirements that are specified in the `-gpu` option because the `-R` option is specifying the `ngpus_physical` GPU resource:

```
bsub -gpu "num=2:mode=exclusive_process" -n2 -R "span[ptile=1]  
rusage[ngpus_physical=2:gmodel=TeslaP100:glink=yes]" ./app
```

Since you can only request `EXCLUSIVE_PROCESS` GPUs with the `-gpu` option, move the `rusage[]` string contents to the `-gpu` option arguments. The following corrected job submission requires two tasks, and each task requires 2 `EXCLUSIVE_PROCESS` Tesla P100 GPUs with NVLink connections on two hosts:

```
bsub -gpu "num=2:mode=exclusive_process:gmodel=TeslaP100:glink=yes" -n2 -R  
"span[ptile=1]" ./app
```

Legacy GPU features using ELIM

Learn how to manually use the legacy ELIM to use GPU features in IBM® Spectrum LSF.

Note: The GPU features ELIM is now deprecated and might be removed in a future version of LSF.

- [Manually configure and use GPU resources \(legacy ELIM procedure\)](#)
Use this procedure to configure and use GPU resources using the legacy ELIM method (pre-LSF 10.1.0.5).
- [Controlling GPU auto-boost](#)
Controlling GPU auto-boost is obsolete as of LSF 10.1 Fix Pack 4 because LSF now synchronizes with GPU auto-boost to resolve any issues that previously required disabling the auto-boost.

Manually configure and use GPU resources (legacy ELIM procedure)

Use this procedure to configure and use GPU resources using the legacy ELIM method (pre-LSF 10.1.0.5).

Procedure

1. Binary files for base elim.gpu are located under \$LSF_SERVERDIR. The binary for optional elim.gpu.ext.c and elim.gpu.topology.c, their makefiles and README files are under LSF_TOP/10.1.0/misc/examples/elim.gpu.ext/. See the README file for steps to build, install, configure, and debug the ELIMs.
Make sure **elim** executable files are in the LSF_SERVERDIR directory.

For GPU support, make sure the following third-party software is installed correctly:

- CUDA driver
 - CUDA toolkit
 - Tesla Deployment Kit
 - NVIDIA Management Library (NVML)
 - CUDA sample is optional.
 - CUDA version must be 4.0 or later.
 - From CUDA 5.0, the CUDA driver, CUDA toolkit, and CUDA samples are in one package.
 - Nodes must have at least one NVIDIA GPU from the Fermi/Kepler family. Earlier Tesla and desktop GPUs of 8800 and later cards are supported. Not all features are available for the earlier cards. Cards earlier than Fermi cards do not support ECC errors, and some do not support Temperature queries.
2. Optionally, enable integration with NVIDIA Data Center GPU Manager (DCGM).
The NVIDIA Data Center GPU Manager (DCGM) is a suite of data center management tools that allow you to manage and monitor GPU resources in an accelerated data center.

Enable the DCGM integration by defining the LSF_DCGM_PORT parameter in the lsf.conf file.

3. Configure the LSF cluster that contains the GPU resources:

- Configure lsf.shared.

For GPU support, define the following resources in the Resource section, assuming that the maximum number of GPUs per host is three. The first four GPUs are provided by base ELIMs. The others are optional. The `ngpus` resource is not consumable. Remove changes that are related to the old GPU solution before you define the new one:

Begin Resource						
RESOURCENAME	TYPE	INTERVAL	INCREASING	CONSUMABLE	DESCRIPTION	
ngpus_prohibited (Prohibited Mode)	Numeric	60	N	N	(Number of GPUs in	
ngpus	Numeric	60	N	N	(Number of GPUs)	
ngpus_shared (Shared Mode)	Numeric	60	N	Y	(Number of GPUs in	
ngpus_excl_t (Exclusive Thread Mode)	Numeric	60	N	Y	(Number of GPUs in	
ngpus_excl_p (Exclusive Process Mode)	Numeric	60	N	Y	(Number of GPUs in	
ngpus_physical (GPUs)	Numeric	60	N	Y	(Number of physical	
gpu_driver	String	60	()	()	(GPU driver version)	
gpu_mode0	String	60	()	()	(Mode of 1st GPU)	
gpu_temp0 (GPU)	Numeric	60	Y	()	(Temperature of 1st	
gpu_ecc0	Numeric	60	N	()	(ECC errors on 1st	

GPU)						
gpu_model0	String	60	()	()	(Model name of 1st GPU)	
gpu_model1	String	60	()	()	(Mode of 2nd GPU)	
gpu_temp1	Numeric	60	Y	()	(Temperature of 2nd GPU)	
gpu_ecc1	Numeric	60	N	()	(ECC errors on 2nd GPU)	
gpu_model11	String	60	()	()	(Model name of 2nd GPU)	
gpu_mode2	String	60	()	()	(Mode of 3rd GPU)	
gpu_temp2	Numeric	60	Y	()	(Temperature of 3rd GPU)	
gpu_ecc2	Numeric	60	N	()	(ECC errors on 3rd GPU)	
gpu_model2	String	60	()	()	(Model name of 3rd GPU)	
gpu_ut0	Numeric	60	Y	()	(GPU utilization of 1st GPU)	
gpu_ut1	Numeric	60	Y	()	(GPU utilization of 2nd GPU)	
gpu_ut2	Numeric	60	Y	()	(GPU utilization of 3rd GPU)	
gpu_shared_avg_ut	Numeric	60	Y	()	(Average of all shared mode GPUs utilization)	
gpu_topology	String	60	()	()	(GPU topology on host)	
gpu_mut0	Numeric	60	Y	()	(GPU memory utilization of 1st GPU)	
gpu_mut1	Numeric	60	Y	()	(GPU memory utilization of 2nd GPU)	
gpu_mut2	Numeric	60	Y	()	(GPU memory utilization of 3rd GPU)	
gpu_mtotal0	Numeric	60	Y	()	(Memory total of 1st GPU)	
gpu_mtotal1	Numeric	60	Y	()	(Memory total of 2nd GPU)	
gpu_mtotal2	Numeric	60	Y	()	(Memory total of 3rd GPU)	
gpu_mused0	Numeric	60	Y	()	(Memory used of 1st GPU)	
gpu_mused1	Numeric	60	Y	()	(Memory used of 2nd GPU)	
gpu_mused2	Numeric	60	Y	()	(Memory used of 3rd GPU)	
gpu_pstate0	String	60	()	()	(Performance state of 1st GPU)	
gpu_pstate1	String	60	()	()	(Performance state of 2nd GPU)	
gpu_pstate2	String	60	()	()	(Performance state of 3rd GPU)	
gpu_shared_avg_mut	Numeric	60	Y	()	(Average memory of all shared mode GPUs)	
gpu_status0	String	60	()	()	(GPU status)	
gpu_status1	String	60	()	()	(GPU status)	
gpu_status2	String	60	()	()	(GPU status)	
gpu_error0	String	60	()	()	(GPU error)	
gpu_error1	String	60	()	()	(GPU error)	
gpu_error2	String	60	()	()	(GPU error)	
...						
End Resource						

The `gpu_status*` and `gpu_error*` resources are only available if you enabled the DCGM integration by defining the `LSF_DCGM_PORT` parameter in the `lsf.conf` file.

- Configure the `lsf.cluster.cluster_name` file.
For GPU support, define the following resources in the `ResourceMap` section. The first four GPUs are provided by the **elims.gpu** ELIM. The others are optional. Remove changes that are related to the old GPU solution before you define the new one:

```
Begin ResourceMap
RESOURCENAME          LOCATION
```

```

...
ngpus_prohibited      ([default])
ngpus                  ([default])
ngpus_shared           ([default])
ngpus_excl_t           ([default])
ngpus_excl_p           ([default])
ngpus_physical         ([hostA] [hostB])
gpu_mode0              ([default])
gpu_temp0              ([default])
gpu_ecc0               ([default])
gpu_mode1              ([default])
gpu_temp1              ([default])
gpu_ecc1               ([default])
gpu_mode2              ([default])
gpu_temp2              ([default])
gpu_ecc2               ([default])
gpu_model0             ([default])
gpu_model1             ([default])
gpu_model2             ([default])
gpu_driver             ([default])
gpu_ut0                ([default])
gpu_ut1                ([default])
gpu_ut2                ([default])
gpu_shared_avg_ut      ([default])
gpu_topology           ([default])
gpu_mut0               ([default])
gpu_mut1               ([default])
gpu_mut2               ([default])
gpu_mtotal0            ([default])
gpu_mtotal1            ([default])
gpu_mtotal2            ([default])
gpu_mused0             ([default])
gpu_mused1             ([default])
gpu_mused2             ([default])
gpu_pstate0            ([default])
gpu_pstate1            ([default])
gpu_pstate2            ([default])
gpu_shared_avg_mut     ([default])
gpu_status0            ([default])
gpu_status1            ([default])
gpu_status2            ([default])
gpu_error0             ([default])
gpu_error1             ([default])
gpu_error2             ([default])
...
End ResourceMap

```

The `gpu_status*` and `gpu_error*` resources are only available if you enabled the DCGM integration by defining the `LSF_DCGM_PORT` parameter in the `lsf.conf` file.

- Optionally, configure `lsb.resources`.
For the `ngpus_shared`, `gpuexcl_t` and `gpuexcl_p` resources, you can set attributes in the `ReservationUsage` section with the following values:

```

Begin ReservationUsage
RESOURCE      METHOD      RESERVE
ngpus_shared  PER_HOST    N
ngpus_excl_t  PER_HOST    N
ngpus_excl_p  PER_HOST    N
nmics         PER_TASK    N
End ReservationUsage

```

If this file has no configuration for GPU resources, by default LSF considers all resources as `PER_HOST`.

- Run the **lsadmin reconfig** and **badadmin mbdrestart** commands to make configuration changes take effect. If you configure the resource `gpu_topology`, run the **badadmin hrestart** command too.
4. Use the **lsload -l** command to show GPU resources:

```

$ lsload -I ngpus:ngpus_shared:ngpus_excl_t:ngpus_excl_p
HOST_NAME      status ngpus ngpus_shared ngpus_excl_t ngpus_excl_p

```


hostA	ok	3.0	12.0	0.0	0.0
hostB	ok	-	-	-	-
hostC	ok	-	-	-	-
hostD	ok	-	-	-	-
hostE	ok	-	-	-	-
hostF	ok	3.0	12.0	0.0	0.0
hostG	ok	3.0	12.0	0.0	1.0
hostH	ok	3.0	12.0	1.0	0.0
hostI	ok	-	-	-	-

5. Use the **bhost -l** command to see how the LSF scheduler allocated the GPU resources. These resources are treated as normal host-based resources:

```
$ bhosts -l hostA
HOST hostA
STATUS  CPUF  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV  DISPATCH_WINDOW
ok      60.00 -    12   2      2      0      0      0   -

CURRENT LOAD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem  slots  nmics
Total    0.0   0.0  0.0  0%  0.0  3   4   0  28G  3.9G  22.5G  10   0.0
Reserved 0.0   0.0  0.0  0%  0.0  0   0   0  0M   0M   0M    -   -

          ngpus ngpus_shared ngpus_excl_t ngpus_excl_p
Total      3.0   10.0          0.0          0.0
Reserved   0.0    2.0          0.0          0.0

LOAD THRESHOLD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

          nmics ngpus ngpus_shared ngpus_excl_t ngpus_excl_p
loadSched -    -    -          -          -
loadStop  -    -    -          -          -
```

6. Use the **lshosts -l** command to see the information for GPUs collected by **elim**:

```
$ lshosts -l hostA

HOST_NAME:  hostA
type      model      cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores
nthreads
X86_64    Intel_EM64T    60.0 12    1      23.9G  3.9G  40317M 0      Yes    2      6      1

RESOURCES: (mg)
RUN_WINDOWS: (always open)

LOAD_THRESHOLDS:
r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem  nmics ngpus ngpus_shared ngpus_excl_t
ngpus_excl_p
-      3.5 -    -    -    -    -    -    -    -    -    -    -    -    -    -
```

You can also use the **bpost** command to display which GPUs are allocated to the job.

7. Submit jobs with GPU resources in resource requirements.

Use the `select[]` string in a resource requirement (`-R`) to choose the hosts that have GPU resources. Use the `rusage[]` resource requirement to tell LSF how many GPU resources to use.

Note: If the `LSB_GPU_NEW_SYNTAX=Y` parameter is specified in the `lsf.conf` file, you must submit your job with the **bsub -gpu** option. You cannot use the GPU resources `ngpus_shared`, `ngpus_excl_t` and `ngpus_excl_p`.

Examples:

- Use a GPU in shared mode:

```
bsub -R "select[ngpus>0] rusage [ngpus_shared=2]" gpu_app
```

- Use a GPU in exclusive thread mode for a PMPI job:

```
bsub -n 2 -R 'select[ngpus>0] rusage[ngpus_excl_t=2]' mpirun -lsf gpu_app1
```

- Use a GPU in exclusive process mode for a PMPI job:

```
bsub -n 4 -R "select[ngpus>0] rusage[ngpus_excl_p=2]" mpirun -lsf gpu_app2
```

- Run a job on 1 host with 8 tasks on it, using 2 ngpus_excl_p in total:

```
bsub -n 8 -R "select[ngpus > 0] rusage[ngpus_excl_p=2] span[hosts=1]" mpirun -lsf gpu_app2
```

- Run a job on 8 hosts with 1 task per host, where every task uses 2 gpushared per host:

```
bsub -n 8 -R "select[ngpus > 0] rusage[ngpus_shared=2] span[ptile=1]" mpirun -lsf gpu_app2
```

- Run a job on 4 hosts with 2 tasks per host, where the tasks use a total of 2 ngpus_excl_t per host.

```
bsub -n 8 -R "select[ngpus > 0] rusage[ngpus_excl_t=2] span[ptile=2]" mpirun -lsf gpu_app2
```

8. Submit jobs with the **bsub -gpu**

The LSB_GPU_NEW_SYNTAX=Y parameter must be specified in the lsf.conf file to submit your job with the **bsub -gpu** option.

Controlling GPU auto-boost

Controlling GPU auto-boost is obsolete as of LSF 10.1 Fix Pack 4 because LSF now synchronizes with GPU auto-boost to resolve any issues that previously required disabling the auto-boost.

NVIDIA K80 hardware provides the auto-boost feature by default. Auto-boost is designed for applications that use a single GPU. For multi-GPU jobs, auto-boost results in different GPUs running at different speeds. LSF can turn off the auto-boost for jobs with exclusive thread/process multi-GPU requirements.

Set the LSB_GPU_AUTOBOOST parameter in the lsf.conf file to control this feature.

If set to N, LSF disables auto-boost for jobs that require multiple exclusive mode GPUs whether these GPUs are on one host or multiple hosts.

When a job requires shared mode GPUs and exclusive mode GPUs at the same time, LSF does not turn off auto-boost for this type of job. When this job is done or exited, LSF restores the original auto-boost status.

Note: LSF ignores an auto-boost configuration change while the **sbatchd** daemon is running. Because of NVML library API limitations, the **sbatchd** daemon cannot call the NVML library periodically to get the newest auto-boost configuration. If you need to change the GPU auto-boost configuration on one host, clean all the jobs on the host, modify the GPU auto-boost configuration, then restart **sbatchd**.

NVIDIA K80 hardware supports auto-boost. The NVML library must be Version 6.340 or newer.

Configuring containers with LSF

Configure and use LSF integrations for containers.

A container is a lightweight operating system level virtualization that is based on Linux control groups (**cgroups**) and namespace. A container runs efficiently and starts based on a predefined image. You can pack an application and release it as a container image. A container is portable and can run on any Linux distribution for any image. LSF supports Docker, Singularity, and Shifter container runtimes.

When used with LSF GPU scheduling, LSF can use the **nvidia-docker** runtime to make allocated GPUs work in the container for application accelerations. LSF starts a job-based container for the job, and the life cycle of the container is the same as the job's. For parallel jobs, LSF starts a set of containers for a job. When the job is finished, LSF destroys all containers.

LSF configures container runtime control in the application profile. It is the LSF administrator's responsibility to configure container runtime in the application profile, and end users do not need to consider which containers are used for their jobs. End users submit their jobs to the application profile and LSF automatically manages the container runtime control.

- [IBM Spectrum LSF with Docker](#)
Configure and use LSF to run jobs in Docker containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.
- [IBM Spectrum LSF with Shifter](#)
Configure and use LSF to run jobs in Shifter containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.
- [IBM Spectrum LSF with Singularity](#)
Configure and use LSF to run jobs in Singularity containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.
- [IBM Spectrum LSF with Podman](#)
Configure and use LSF to run jobs in Pod Manager (podman) OCI containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.
- [IBM Spectrum LSF with Enroot](#)
Configure and use LSF to run jobs in Enroot containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.

IBM Spectrum LSF with Docker

Configure and use LSF to run jobs in Docker containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.

LSF provides the following benefits when using Docker containers:

Resolves Docker security concerns

The most serious security concern with Docker is its privilege escalation in a container. A user that starts a Docker container can be root within that container. LSF takes over permission management for docker containers, ensuring that jobs are launched within the container and only has permissions of the execution user.

LSF also enables administrators to configure Docker images and runtime options in an application profile. Administrators have full control of the images, including the register from where the image originated, the configured image name, and the image version to be used. LSF forbids common cluster users from using arbitrary images to avoid potential security risks.

Schedules Docker jobs on Docker hosts

LSF uses a boolean resource named docker to identify a host in which the Docker service is started. LSF automatically attaches the docker boolean resource to Docker jobs and schedules Docker jobs to hosts with Docker available.

Automatically sets the Docker runtime environment

Before an LSF job starts on the execution host, LSF sets the execution context similar to the submission environment and settings. This includes current environment variables, CPU affinity, process limits, and the execution user. In Docker environments, LSF automatically sets the job execution context in the container.

Launches containers for parallel jobs

A parallel job might have several ranks across hosts. For container parallel jobs, LSF launches a container on each host for tasks. The task container has the same execution environment settings as the job container. LSF ensures that the MPI environment that is used for the job is packed in the image. Therefore, there are no dependencies on the host MPI environment.

Supports official cgroup drivers

Docker supports both **cgroupfs** and **systemd** cgroup drivers to manage container resources with Linux control groups. Docker uses **cgroupfs** by default, but RHEL releases now use **systemd** as the default cgroup driver. LSF supports both cgroups with

execution driver scripts.

Uses GPUs for application acceleration

When submitting a Docker job with GPU resource requirements, LSF schedules GPUs for such jobs. LSF schedules the jobs with GPUs allocated and dispatched onto the Docker host. While the Docker container is starting up, LSF uses the **nvidia-docker** runtime to start the container with the allocated GPU attached.

- [Preparing LSF to run Docker jobs](#)
Prepare LSF to run jobs in Docker containers.
- [Configuring LSF to run Docker jobs](#)
Configure the Docker application profile or queue in LSF to run Docker jobs.
- [Configuring LSF to run NVIDIA Docker jobs](#)
Configure the NVIDIA Docker application profile or queue in LSF to run NVIDIA Docker jobs.
- [Submitting Docker jobs to LSF](#)
Use the Docker application profiles or queues to submit Docker jobs to LSF.
- [Submitting NVIDIA Docker jobs to LSF](#)
Use the NVIDIA Docker application profiles or queues to submit NVIDIA Docker jobs to LSF.

Preparing LSF to run Docker jobs

Prepare LSF to run jobs in Docker containers.

Before you begin

The Docker Engine, Version 1.12, or later, must be installed on an LSF server host. The Docker daemon must be started on this host and can successfully start containers.

Procedure

1. Edit the `lsf.conf` file and configure the following parameters:

```
LSF_PROCESS_TRACKING=Y
LSF_LINUX_CGROUP_ACCT=Y
LSB_RESOURCE_ENFORCE="cpu memory"
```

2. Edit the `lsf.shared` file and configure a new Boolean resource named **docker**.

```
...
Begin Resource
RESOURCENAME  TYPE      INTERVAL  INCREASING  CONSUMABLE  DESCRIPTION  # Keywords
...
  docker      Boolean  ()         ()           ()           (Docker container)
...
End Resource
```

3. Edit the `lsf.cluster` file and attach the **docker** Boolean resource to the LSF server host that is running Docker Engine. This configuration enables LSF to automatically dispatch Docker jobs to the LSF server host that is running Docker Engine.

```
...
Begin Host
  HOSTNAME  model  type  server  rlm  mem  swp  RESOURCES
  ...
  host1     !      !      1       3.5  ()   ()   (docker)
  ...
End Host
```

4. Check if `/usr/bin/python` is linked to the proper Python interpreter.
In some versions of Linux (such as RHEL 8.x), there is no `/usr/bin/python` executable file on the host.

Since the execution driver uses `/usr/bin/python` to execute the Python script, you must create a link from `/usr/bin/python` linking to the available Python executable file (both **python2.x** and **python3.x** are available).

For example, to link to python3,

```
$ ln -s /usr/bin/python3 /usr/bin/python
```

Configuring LSF to run Docker jobs

Configure the Docker application profile or queue in LSF to run Docker jobs.

About this task

You cannot run pre-execution and post-execution scripts in container jobs. The following are workarounds for specific pre-execution and post-execution operations:

- To prepare data for the container as a pre-execution or post-execution operation, put this data into a directory that is mounted to a job container.
- To customize the internal job container, you can customize the starter scripts to prepare the appropriate environment.

Procedure

1. Edit the `lsb.applications` or `lsb.queues` file and define the `CONTAINER` parameter for the application profile or queue to run Docker jobs.

If this parameter is specified in both files, the parameter value in the `lsb.applications` file overrides the value in the `lsb.queues` file.

Use this syntax:

```
CONTAINER=docker[image(image_name) options(docker_run_options)]
```

Note: Do not use the following options when configuring Docker options in the options keyword configuration or the options script:

- The `--cgroup-parent`, `--user`, `-u`, and `--name` options are reserved for LSF internal use. Do not use these options in the options keyword configuration or in the options script.
- The `-w` and `--ulimit` options are automatically set for LSF. Do not use these options in the options keyword configuration or options script because the specifications here override the LSF settings.

For more details, refer to [the CONTAINER parameter in the lsb.applications file](#) or [the CONTAINER parameter in the lsb.queues file](#).

You can enable LSF to automatically assign a name to a Docker container when it creates the Docker container. To enable this feature, set the `ENABLE_CONTAINER_NAME` parameter to `True` in the `lsfdockerlib.py` file.

The container name uses the following naming conventions:

- Normal jobs and **blaunch** parallel job containers: `<cluster_name> . job . <job_id>`
- Array jobs and array **blaunch** parallel job containers: `<cluster_name> . job . <job_id> . <job_index>`
- **blaunch** parallel job task containers: `<cluster_name> . job . <job_id> . task . <task_id>`
- Array **blaunch** parallel job task containers: `<cluster_name> . job . <job_id> . <job_index> . task . <task_id>`

In the following examples, LSF uses the `ubuntu` image to run the job in the Docker container.

- For sequential jobs:

```
CONTAINER=docker[image(ubuntu) options(--rm)]
```

The container for the job is removed after the job is done, which is enabled with the **docker run --rm** option.

- For parallel jobs:

```
CONTAINER = docker[image(ubuntu) options(--rm --net=host --ipc=host -v /path/to/my/passwd:/etc/passwd)]
```

This command uses the following **docker run** options:

--rm
The container for the job is removed after the job is done

--net=host
LSF needs the host network for launching parallel tasks.

-v
LSF needs the user ID and user name for launching parallel tasks.

Note: The passwd file must be in the standard format for UNIX and Linux password files, such as the following format:

```
user1:x:10001:10001:::  
user2:x:10002:10002:::
```

2. Edit the lsb.applications or lsb.queues file and define the EXEC_DRIVER parameter for the application profile or queue to run Docker jobs.

If this parameter is specified in both files, the parameter value in the lsb.applications file overrides the value in the lsb.queues file.

```
EXEC_DRIVER=context[user(lsfadmin)]  
    starter[/path/to/serverdir/docker-starter.py]  
    controller[/path/to/serverdir/docker-control.py]  
    monitor[/path/to/serverdir/docker-monitor.py]
```

Replace `/path/to/serverdir` with the actual file path of the `LSF_SERVERDIR` directory.

For more details, refer to [the EXEC_DRIVER parameter in the lsb.applications file](#) or [the EXEC_DRIVER parameter in the lsb.queues file](#).

3. Optional: Enable Docker image affinity by defining DOCKER_IMAGE_AFFINITY=Y in the lsb.applications file for the application profile to run Docker jobs, the lsb.queues file for the queue to run Docker jobs, or the lsb.params file for the entire cluster.

Docker image affinity enables LSF to give preference for execution hosts that have already have the requested Docker image. This reduces network bandwidth and the job start time because the execution host does not have to pull the Docker image from the repository and the job can immediately start on the execution host.

If this parameter is specified in both files, the parameter value in the lsb.applications file overrides the value in the lsb.queues file, which overrides the value in the lsb.params file.

For more details, refer to [the DOCKER_IMAGE_AFFINITY parameter in the lsb.applications file](#), [the DOCKER_IMAGE_AFFINITY parameter in the lsb.queues file](#), or [the DOCKER_IMAGE_AFFINITY parameter in the lsb.params file](#).

Configuring LSF to run NVIDIA Docker jobs

Configure the NVIDIA Docker application profile or queue in LSF to run NVIDIA Docker jobs.

About this task

If you are using the NVIDIA Docker integration, you need to configure separate application profiles or queues to run NVIDIA Docker jobs.

You cannot run pre-execution and post-execution scripts in container jobs. The following are workarounds for specific pre-execution and post-execution operations:

- To prepare data for the container as a pre-execution or post-execution operation, put this data into a directory that is mounted to a job container.
- To customize the internal job container, you can customize the starter scripts to prepare the appropriate environment.

Procedure

Edit the `lsb.applications` or `lsb.queues` file and define the `CONTAINER` parameter for the application profile or queue to run NVIDIA Docker jobs.

If this parameter is specified in both files, the parameter value in the `lsb.applications` file overrides the value in the `lsb.queues` file.

```
CONTAINER=nvidia-docker[image (image_name) options (docker_run_options) ]
```

In the following examples, LSF uses the `ubuntu` image to run the job in the Docker container.

- For sequential jobs:

```
CONTAINER=nvidia-docker[image (ubuntu) options (--rm) ]
```

The container for the job is removed after the job is done, which is enabled with the **docker run --rm** option.

- For parallel jobs:

```
CONTAINER = nvidia-docker[image (ubuntu) options (--rm --net=host --ipc=host -v --runtime=nvidia /path/to/my/passwd:/etc/passwd)
```

This command uses the following **docker run** options:

`--rm`

The container for the job is removed after the job is done

`--net=host`

LSF needs the host network for launching parallel tasks.

`-v`

LSF needs the user ID and user name for launching parallel tasks.

`--runtime=nvidia`

You must specify this option if the container image is using NVIDIA Docker, version 2.0.

Note: The `passwd` file must be in the standard format for UNIX and Linux password files, such as the following format:

```
user1:x:10001:10001:::  
user2:x:10002:10002:::
```

For more details, refer to the `CONTAINER` parameter in the [lsb.applications file](#) or [lsb.queues file](#).

Submitting Docker jobs to LSF

Use the Docker application profiles or queues to submit Docker jobs to LSF.

About this task

The following job submission (**bsub**) options are not supported for container jobs: `-E`, `-Ep`.

Procedure

1. Submit a job to a Docker application profile or queue.

- Use the **bsub -app** option to submit a job to a Docker application profile (that is, an application profile with a specified value for the `CONTAINER` parameter).

For example, if `CONTAINER` is defined in the **dockerapp** application profile, run the following command to submit a Docker job:

```
bsub -app dockerapp ./myjob.sh
```

- Use the **bsub -q** option to submit a job to a Docker queue (that is, a queue with a specified value for the `CONTAINER` parameter).

For example, if `CONTAINER` is defined in the `dockerq` queue, run the following command to submit a Docker job:

```
bsub -q dockerq ./myjob.sh
```

If you specify both the application profile and the queue in the same job submission, the `CONTAINER`, `EXEC_DRIVER`, and `DOCKER_IMAGE_AFFINITY` parameter settings in the application profile take precedence over the same parameter settings in the queue.

For example, if you submit a job with the following command, the parameters in the `dockerapp` application profile override the corresponding parameters in the `dockerq` queue:

```
bsub -app dockerapp -q dockerq ./myjob.sh
```

2. If users are allowed to specify image names for Docker container jobs at job submission time (that is, if the `$LSB_CONTAINER_IMAGE` environment variable is used as the image name when specifying the image keyword), set the `LSB_CONTAINER_IMAGE` environment variable at job submission time to specify the Docker image name. Specify a container image name (such as `ubuntu`) at the job submission time by setting the `LSB_CONTAINER_IMAGE` environment using one of the following methods:

- Specify the `LSB_CONTAINER_IMAGE` environment variable according to your shell environment. For example, to specify the Docker image named `ubuntu`, set the environment variable as follows:

- In **csh** or **tcsh**:
`setenv LSB_CONTAINER_IMAGE ubuntu`
- In **sh**, **ksh**, or **bash**:
`export LSB_CONTAINER_IMAGE=ubuntu`

- Use the **bsub -env** option. For example, to specify the Docker image named `ubuntu`
`bsub -env LSB_CONTAINER_IMAGE=ubuntu -app udocker a.out -in in.dat -out out.dat`

- Use an **esub** script to set the `LSB_CONTAINER_IMAGE` environment variable, then call the **esub** with the **bsub** command:

For example, create an **esub.docker** script in the `$LSF_SERVERDIR` directory with the following contents:

```
#!/bin/sh
exec 1>&2
echo "LSB_CONTAINER_IMAGE=\"$1\" >> $LSB_SUB_MODIFY_ENVFILE
```

Submit a job to call the **esub.docker** script by running the following command:

```
bsub -a "docker(ubuntu)" -app udocker a.out -in in.dat -out out.dat
```

3. To submit a job with a Docker entry point image, but no command, use the `LSB_DOCKER_PLACE_HOLDER` keyword instead of the command.

For example, run the following command to submit a Docker job with a Docker entry point image:

```
bsub -app dockerapp LSB_DOCKER_PLACE_HOLDER
```

4. Optional: To override the Docker image affinity settings at the cluster, queue, or application level, define the `LSB_DOCKER_IMAGE_AFFINITY` environment variable with the `-env` option when submitting the Docker job. `LSB_DOCKER_IMAGE_AFFINITY=Y | y | N | n`

The job level environment variable overrides application level, which overrides queue level, which overrides the cluster level configuration.

```
bsub -env LSB_DOCKER_IMAGE_AFFINITY=N -app udocker a.out -in in.dat -out out.dat
```

Submitting NVIDIA Docker jobs to LSF

Use the NVIDIA Docker application profiles or queues to submit NVIDIA Docker jobs to LSF.

About this task

The following job submission (**bsub**) options are not supported for container jobs: -E, -Ep.

Procedure

1. Submit a job to an NVIDIA Docker application profile or queue.

- Use the **bsub -app** option to submit a job to an NVIDIA Docker application profile (that is, an application profile with a specified value for the CONTAINER parameter).

For example, if CONTAINER is defined in the **ndockerapp** application profile, run the following command to submit a Docker job:

```
bsub -app ndockerapp ./myjob.sh
```

- Use the **bsub -q** option to submit a job to an NVIDIA Docker queue (that is, a queue with a specified value for the CONTAINER parameter).

For example, if CONTAINER is defined in the **ndockerq** queue, run the following command to submit a Docker job:

```
bsub -queue ndockerq ./myjob.sh
```

If you specify both the application profile and the queue in the same job submission, the CONTAINER and EXEC_DRIVER parameter settings in the application profile take precedence over the same parameter settings in the queue.

For example, if you submit a job with the following command, the parameters in the **ndockerapp** application profile override the corresponding parameters in the **ndockerq** queue:

```
bsub -app ndockerapp -q ndockerq ./myjob.sh
```

2. If users need GPUs for the job, use the **-gpu** option to specify GPU resources.

For example, if CONTAINER is defined in the **ndockerapp** application profile, run the following command to request 1 GPU exclusively for the Docker job:

```
bsub -gpu "num=1:j_exclusive=yes" -app ndockerapp ./myjob.sh
```

Submitting Docker jobs to LSF

Use the Docker application profiles or queues to submit Docker jobs to LSF.

About this task

The following job submission (**bsub**) options are not supported for container jobs: -E, -Ep.

Procedure

1. Submit a job to a Docker application profile or queue.

- Use the **bsub -app** option to submit a job to a Docker application profile (that is, an application profile with a specified value for the CONTAINER parameter).

For example, if CONTAINER is defined in the **dockerapp** application profile, run the following command to submit a Docker job:

```
bsub -app dockerapp ./myjob.sh
```

- Use the **bsub -q** option to submit a job to a Docker queue (that is, a queue with a specified value for the CONTAINER parameter).

For example, if CONTAINER is defined in the **dockerq** queue, run the following command to submit a Docker job:

```
bsub -q dockerq ./myjob.sh
```

If you specify both the application profile and the queue in the same job submission, the CONTAINER, EXEC_DRIVER, and DOCKER_IMAGE_AFFINITY parameter settings in the application profile take precedence over the same parameter settings in the queue.

For example, if you submit a job with the following command, the parameters in the **dockerapp** application profile override the corresponding parameters in the **dockerq** queue:

```
bsub -app dockerapp -q dockerq ./myjob.sh
```

2. If users are allowed to specify image names for Docker container jobs at job submission time (that is, if the `$LSB_CONTAINER_IMAGE` environment variable is used as the image name when specifying the image keyword), set the `LSB_CONTAINER_IMAGE` environment variable at job submission time to specify the Docker image name. Specify a container image name (such as `ubuntu`) at the job submission time by setting the `LSB_CONTAINER_IMAGE` environment using one of the following methods:

- Specify the `LSB_CONTAINER_IMAGE` environment variable according to your shell environment. For example, to specify the Docker image named `ubuntu`, set the environment variable as follows:

- In **csh** or **tcsh**:

```
setenv LSB_CONTAINER_IMAGE ubuntu
```

- In **sh**, **ksh**, or **bash**:

```
export LSB_CONTAINER_IMAGE=ubuntu
```

- Use the **bsub -env** option. For example, to specify the Docker image named `ubuntu`

```
bsub -env LSB_CONTAINER_IMAGE=ubuntu -app udocker a.out -in in.dat -out out.dat
```

- Use an **esub** script to set the `LSB_CONTAINER_IMAGE` environment variable, then call the **esub** with the **bsub** command:
For example, create an **esub.docker** script in the `$LSF_SERVERDIR` directory with the following contents:

```
#!/bin/sh
exec 1>&2
echo "LSB_CONTAINER_IMAGE=\"$1\" >> $LSB_SUB_MODIFY_ENVFILE
```

Submit a job to call the **esub.docker** script by running the following command:

```
bsub -a "docker(ubuntu)" -app udocker a.out -in in.dat -out out.dat
```

3. To submit a job with a Docker entry point image, but no command, use the `LSB_DOCKER_PLACE_HOLDER` keyword instead of the command.

For example, run the following command to submit a Docker job with a Docker entry point image:

```
bsub -app dockerapp LSB_DOCKER_PLACE_HOLDER
```

4. Optional: To override the Docker image affinity settings at the cluster, queue, or application level, define the `LSB_DOCKER_IMAGE_AFFINITY` environment variable with the `-env` option when submitting the Docker job.
`LSB_DOCKER_IMAGE_AFFINITY=Y | y | N | n`

The job level environment variable overrides application level, which overrides queue level, which overrides the cluster level configuration.

```
bsub -env LSB_DOCKER_IMAGE_AFFINITY=N -app udocker a.out -in in.dat -out out.dat
```

IBM Spectrum LSF with Shifter

Configure and use LSF to run jobs in Shifter containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.

LSF launches containers for a parallel job. LSF also launches the related MPI service (such as **orted**) inside the container instead of launching only the task in the container and not the MPI service. This ensures that the container uses a pure image MPI environment, which is easier to maintain.

- [Configuring LSF to run Shifter jobs](#)
Configure a Shifter application profile or queue in LSF to run Shifter container jobs.
- [Submitting Shifter jobs to LSF](#)
Use the Shifter application profiles or queues to submit Shifter jobs to LSF.

Configuring LSF to run Shifter jobs

Configure a Shifter application profile or queue in LSF to run Shifter container jobs.

Before you begin

- Shifter, Version 16.08.3, or later (that is, built from code later than 3 August 2016), must be installed on an LSF server host.
- Add the path to the Shifter executable file to the \$PATH environment variable before submitting jobs, and ensure that it is accessible on each execution host.
- The required Shifter images that you intend to use must be prepared before running LSF jobs in containers. Ensure that these images are accessible to each execution host.
- If you use mounted file systems for LSF, other software installations, or for input/output directories, you must also bind required directories into containers to ensure that LSF works properly. To bind directories for Shifter containers, add the pre-mount hook script sitePreMountHook to the udiRoot.conf file. You can define which directories to bind to the container in your script.
- To run jobs with **bsub -L login_shell** or where the execution user is changed by user mapping, the execution user must be able to execute the **shifter** command under its \$PATH environment variable. You can install the **shifter** binary file in the /usr/bin directory or add a symbolic link to the installation path of **shifter**.

About this task

You cannot run pre-execution and post-execution scripts in container jobs. The following are workarounds for specific pre-execution and post-execution operations:

- To prepare data for the container as a pre-execution or post-execution operation, put this data into a directory that is mounted to a job container.
- To customize the internal job container, you can customize the starter scripts to prepare the appropriate environment.

Procedure

Edit the lsb.applications or lsb.queues file and define the CONTAINER parameter for the application profile or queue to run Shifter container jobs.

If this parameter is specified in both files, the parameter value in the lsb.applications file overrides the value in the lsb.queues file.

```
CONTAINER=shifter[image(image_name) options(container_options)]
```

image

Required. This keyword specifies the Shifter image name that is used in running jobs.

options

Optional. This keyword specifies the Shifter job run options, which are passed to the job container with the **shifter** command in LSF.

To enable a pre-execution script to run, specify an at sign (@) and a full file path to the script, which the execution host must be able to access. Before the container job runs, LSF runs this script with LSF administrator privileges. While the script is running, the jobs' environment variables are passed to the script. When the script finishes running, the output is used as container startup options. The script must provide this output on one line.

Note:

- Run **shifter --help** in the command line to view the options that the **shifter** command supports.
- Before you specify the Shifter job run options, make sure that these options work with the **shifter** command in the command line.
- The \$LD_LIBRARY_PATH directory is cleaned according to the setuid bit that Shifter uses to work. Therefore, for programs that depend on \$LD_LIBRARY_PATH to work (such as **openmpi**), ensure that you added LD_LIBRARY_PATH to the siteEnvAppend section of the udiRoot.conf file.

For example, to specify the ubuntu:latest image for use with Shifter container jobs without specifying any optional keywords,

```
Begin Application
NAME = shifterapp
CONTAINER = shifter[image(ubuntu:latest)]
```

```
DESCRIPTION = Shifter User Service
End Application
```

To specify the pre-execution script `/share/usr/shifter-options.sh`, which generates the container startup options,

```
Begin Application
NAME = shifterappoptions
CONTAINER = shifter[image(ubuntu:latest) options(@/share/usr/shifter-options.sh)]
DESCRIPTION = Shifter User Service with pre-execution script for options
End Application
```

For more details, refer to the CONTAINER parameter in the [lsb.applications file](#) or [lsb.queues file](#).

Submitting Shifter jobs to LSF

Use the Shifter application profiles or queues to submit Shifter jobs to LSF.

Before you begin

Ensure that you configured the Shifter application profile or queue before submitting Shifter jobs.

About this task

The following job submission (**bsub**) options are not supported for container jobs: `-E`, `-Ep`.

Procedure

Submit a job to a Shifter application profile or queue.

- Use the **bsub -app** option to submit a job to a Shifter application profile (that is, an application profile with a specified value for the CONTAINER parameter).
For example, if CONTAINER is defined in the **shifterapp** application profile, run the following command to submit a Shifter job:

```
bsub -app shifterapp ./myjob.sh
```

- Use the **bsub -q** option to submit a job to a Shifter queue (that is, a queue with a specified value for the CONTAINER parameter).
For example, if CONTAINER is defined in the **shifterq** queue, run the following command to submit a Shifter job:

```
bsub -q shifterq ./myjob.sh
```

If you specify both the application profile and the queue in the same job submission, the CONTAINER and EXEC_DRIVER parameter settings in the application profile take precedence over the same parameter settings in the queue.

For example, if you submit a job with the following command, the parameters in the **shifterapp** application profile override the corresponding parameters in the **shifterq** queue:

```
bsub -app shifterapp -q shifterq ./myjob.sh
```

Note:

- If the CWD is not accessible inside the container, the CWD is automatically set to the root directory in the container.
- IBM® Spectrum LSF Session Scheduler cannot run in Shifter containers.

IBM Spectrum LSF with Singularity

Configure and use LSF to run jobs in Singularity containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.

LSF launches containers for a parallel job. LSF also launches the related MPI service (such as **orted**) inside the container instead of launching only the task in the container and not the MPI service. This ensures that the container uses a pure image MPI environment, which is easier to maintain.

- [Configuring LSF to run Singularity jobs](#)
Configure a Singularity application profile or queue in LSF to run Singularity container jobs.
- [Submitting Singularity jobs](#)
Use the Singularity application profiles or queues to submit Singularity jobs to LSF.

Configuring LSF to run Singularity jobs

Configure a Singularity application profile or queue in LSF to run Singularity container jobs.

Before you begin

- Singularity, Version 2.2, or later, must be installed on an LSF server host.
- Add the path to the Singularity executable file to the \$PATH environment variable before submitting jobs, and ensure that it is accessible on each execution host.
- The required Singularity images that you intend to use must be prepared before running LSF jobs in containers. Ensure that these images are accessible to each execution host.
- If you use mounted file systems for LSF, other software installations, or for input/output directories, you must also bind required directories into containers to ensure that LSF works properly. To bind directories for Singularity containers, add `mount hostfs = yes` to the singularity.conf file. This parameter definition binds all mounted file systems to the container.
- To run jobs with **bsub -L login_shell** or where the execution user is changed by user mapping, the execution user must be able to execute the **singularity** command under its \$PATH environment variable. You can install the **singularity** binary file in the /usr/bin directory or add a symbolic link to the installation path of **singularity**.

About this task

You cannot run pre-execution and post-execution scripts in container jobs. The following are workarounds for specific pre-execution and post-execution operations:

- To prepare data for the container as a pre-execution or post-execution operation, put this data into a directory that is mounted to a job container.
- To customize the internal job container, you can customize the starter scripts to prepare the appropriate environment.

Procedure

Edit the lsb.applications or lsb.queues file and define the CONTAINER parameter for the application profile or queue to run Singularity container jobs.

If this parameter is specified in both files, the parameter value in the lsb.applications file overrides the value in the lsb.queues file.

```
CONTAINER=singularity[image(image_name) options(container_options)]
```

image

Required. This keyword specifies the Singularity image name that is used in running jobs.

options

Optional. This keyword specifies the Singularity job run options, which are passed to the job container with the **singularity exec** command in LSF.

To enable a pre-execution script to run, specify an at sign (@) and a full file path to the script, which the execution host must be able to access. Before the container job runs, LSF runs this script with LSF administrator privileges. While the

script is running, the jobs' environment variables are passed to the script. When the script finishes running, the output is used as container startup options. The script must provide this output on one line.

Note:

- Run **singularity exec --help** in the command line to view the options that the **singularity** command supports.
- Before you specify the Singularity job run options, make sure that these options work with the **singularity exec** command in the command line.
- The `$LD_LIBRARY_PATH` directory is cleaned according to the setuid bit that Singularity uses to work. Therefore, for programs that depend on `$LD_LIBRARY_PATH` to work (such as **openmpi**), ensure that you add `$LD_LIBRARY_PATH` to the `ld.so.conf` file and run **ldconfig**.

For example, to specify the `ubuntu.img` image for use with Singularity container jobs without specifying any optional keywords,

```
Begin Application
NAME = singapp
CONTAINER = singularity[image(/file/path/ubuntu.img)]
DESCRIPTION = Singularity User Service
End Application
```

To specify the pre-execution script `/share/usr/sing-options.sh`, which generates the container startup options,

```
Begin Application
NAME = singappoptions
CONTAINER = singularity[image(/file/path/ubuntu.img) options(@/share/usr/sing-options.sh)]
DESCRIPTION = Singularity User Service with pre-execution script for options
End Application
```

For more details, refer to the `CONTAINER` parameter in the [lsb.applications file](#) or [lsb.queues file](#).

Submitting Singularity jobs

Use the Singularity application profiles or queues to submit Singularity jobs to LSF.

Before you begin

Ensure that you configured the Singularity application profile or queue before submitting Singularity jobs.

About this task

The following job submission (**bsub**) options are not supported for container jobs: `-E`, `-Ep`.

Procedure

Submit a job to a Singularity application profile or queue.

- Use the **bsub -app** option to submit a job to a Singularity application profile (that is, an application profile with a specified value for the `CONTAINER` parameter).
For example, if `CONTAINER` is defined in the **singapp** application profile, run the following command to submit a Singularity job:

```
bsub -app singapp ./myjob.sh
```

- Use the **bsub -q** option to submit a job to a Singularity queue (that is, a queue with a specified value for the `CONTAINER` parameter).
For example, if `CONTAINER` is defined in the **singq** queue, run the following command to submit a Singularity job:

```
bsub -q singq ./myjob.sh
```

If you specify both the application profile and the queue in the same job submission, the `CONTAINER` and `EXEC_DRIVER` parameter settings in the application profile take precedence over the same parameter settings in the queue.

For example, if you submit a job with the following command, the parameters in the `singapp` application profile override the corresponding parameters in the `singq` queue:

```
bsub -app singapp -q singq ./myjob.sh
```

Note:

- If the CWD is not accessible inside the container, the CWD is automatically set to the root directory in the container.
- IBM® Spectrum LSF Session Scheduler cannot run in Shifter containers.

IBM Spectrum LSF with Podman

Configure and use LSF to run jobs in Pod Manager (podman) OCI containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.

You cannot use LSF to run Docker jobs if you are using LSF to run Podman jobs.

- [Preparing LSF to run Podman jobs](#)
Prepare LSF to run jobs in Podman containers.
- [Configuring LSF to run Podman jobs](#)
Configure the Podman application profile or queue in LSF to run jobs in Podman containers.
- [Submitting Podman jobs to LSF](#)
Use the Podman application profiles or queues to submit Podman jobs to LSF.

Preparing LSF to run Podman jobs

Prepare LSF to run jobs in Podman containers.

Before you begin

- You cannot run Docker container jobs on LSF if you are also using LSF to run Podman jobs.
- Get the following podman and podman-docker packages for the Podman 3.3.1 container engine.
The Podman installation packages will be installed on the LSF server hosts where you intend to run Podman container jobs. The minimum Podman version is Podman 3.3.1 on a RHEL 8.2 host. For optimal performance, use Podman 3.3.1, or newer, on a RHEL 8.2.1 host.

All LSF server hosts for Podman container jobs must be using the same version of the Podman packages.

- **cgroups** v1 or v2 is enabled on the LSF server hosts where you intend to run Podman container jobs. Since Podman does not use **cgroups** when running in daemon-less mode, all Podman-related processes (**podman**, **fuse-overlayfs**, and **common**) are put in the job or task cgroup.

Procedure

1. Set up the Podman container engine on each LSF server host that will run Podman container jobs.
 - a. Log in to the LSF server host as root.
 - b. Install the podman and podman-docker packages on the LSF server host.
For example, for Podman 3.3.1:

```
# rpm -qa |grep podman
podman-3.3.1.module_e18.2.0+305+5e198a41.x86_64
podman-docker-3.3.1.module_e18.2.0+305+5e198a41.noarch
```

- c. Create `/etc/subuid` and `/etc/subgid` files for LSF users.
For example, the following files set subordinate UIDs and GIDs for the `lsfadmin` user for Podman containers:

```
# cat /etc/subuid
lsfadmin:100000:65536
# cat /etc/subgid
lsfadmin:100000:65536
```

For a user to submit an LSF Podman job, the subordinate UID and GID for that user must be in these files.

- d. Enable the user linger state for the LSF users.

```
$ sudo systemctl restart systemd-logind.service
$ sudo loginctl enable-linger user_name
```

The user linger state is required because LSF runs Podman jobs on execution hosts even if the user is not logged in to the execution host. This is required for LSF to kill Podman jobs.

- e. Update the UID and GID values.

Run the following commands to update the **newuidmap** cap value, the **newgidmap** cap value, and the Podman **subuid** and **subgid** settings:

```
$ sudo setcap cap_setuid+eip /usr/bin/newuidmap
$ sudo setcap cap_setgid+eip /usr/bin/newgidmap
```

- f. Create a symbolic link from /usr/bin/python to the Python executable file.

In RHEL 8.x, python3 is default installed version, and there is no /usr/bin/python executable file on the host. Since the execution driver uses /usr/bin/python to execute the Python script, you must create a link from /usr/bin/python linking to the available Python executable file (both python2.x and python3.x are available).

For example, to link to python3,

```
$ ln -s /usr/bin/python3 /usr/bin/python
```

- g. Edit the /etc/containers/registries.conf file and remove redundant entries to optimize the container image download time.

For example, comment out all the registries except `docker.io`.

```
[registries.search]
#registries = ['registry.redhat.io', 'registry.access.redhat.com', 'quay.io',
'docker.io']
registries = ['docker.io']
```

- h. Create a /etc/containers/nodocker file to suppress redundant Docker messages.

```
# touch /etc/containers/nodocker
```

- i. Log in to the LSF server host as a user that will submit Podman jobs.

- j. If you use an LDAP server to manage the LSF users and the user home directories are on an NFS server, change the Podman user configuration file to save Podman images to a local file system instead of an NFS directory. Podman does not support building or loading container images on NFS home directories. For more details, refer to the [Podman troubleshooting website](https://github.com/containers/podman/blob/master/troubleshooting.md#14-rootless-podman-build-fails-eperm-on-nfs) (<https://github.com/containers/podman/blob/master/troubleshooting.md#14-rootless-podman-build-fails-eperm-on-nfs>).

For example, perform the following steps to save Podman images to a local file system:

- i. Run the **podman system reset** command to reset the Podman configuration.
- ii. Create Podman configuration files from the templates.

```
$ mkdir -p $HOME/.config/containers
$ cp /usr/share/containers/containers.conf $HOME/.config/containers
$ cp /etc/containers/storage.conf $HOME/.config/containers
```

- iii. Create a directory on the local file system, then confirm that your current user name is the owner of the directory and has read/write permissions for that directory.

```
$ ls -l -d /podmanfs/userA/
drwxr-xr-x 3 userA groupA 20 Nov 11 22:30 /podmanfs/userA/
```

- iv. Update the `volume_path`, `tmp_dir`, and `static_dir` parameters in the `containers.conf` file, and update the `graphroot`, `runroot`, and `[storage.options]` parameters in the `storage.conf` file.

For example, if you are using the **bash** shell, run the following commands:

```
$ podmandir="/podmanfs/userA/"
$ userconf=$HOME/.config/containers/containers.conf
$ storageconf=$HOME/.config/containers/storage.conf
$ userid=`id -u`
$ sed -i "s|^#.*volume_path.*|volume_path=\"$podmandir/volumes\"|g" $userconf
$ sed -i "s|^#.*tmp_dir.*|tmp_dir=\"/tmp/run-$userid\"|g" $userconf
$ sed -i "s|^#.*static_dir.*|static_dir=\"$podmandir/libpod\"|g" $userconf
$ sed -i "s|^#.*graphroot.*|graphroot=\"$podmandir/storage\"|g" $storageconf
$ sed -i "s|^#.*runroot.*|runroot=\"/run/user/$userid\"|g" $storageconf
$ sed -i 's|^#.*storage.options.*|amount_program = "/usr/bin/fuse-overlayfs"'
$storageconf
```

- v. Run the **podman system migrate** command to apply the changes.
- vi. Run the **podman info** command to confirm that the changes are applied:

```
$ podman info |grep -niE 'volumepath|graphroot|runroot'
73: graphRoot: /podmanfs/username/storage
81: runRoot: /run/user/userid
82: volumePath: /podmanfs/username/volumes
```

- k. Check to see if Podman is running correctly.
Pull a Docker image and run a Podman job to check if the container ID is created.

For example, to pull a CentOS image and run a Podman job, run the following commands:

```
$ podman pull centos
$ podman run --rm --detach centos sleep 200
15ec62cece2cc2fa5b9fae6c114e97d655240623994814c9a97c64b9635c607c
$ podman ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
NAMES
15ec62cece2c docker.io/library/centos:latest sleep 200 7 seconds ago Up 12345
fervent_sutherland
```

The Podman job output is the container ID. Verify that the container ID directory is created:

```
$ find /sys/fs/cgroup/ -iname
15ec62cece2cc2fa5b9fae6c114e97d655240623994814c9a97c64b9635c607c
```

If the container ID directory is not created, log out, then log back in and try to start the container again.

- 2. Edit the `lsf.conf` file and configure the following parameters:

```
LSF_PROCESS_TRACKING=Y
LSF_LINUX_CGROUP_ACCT=Y
LSB_RESOURCE_ENFORCE="cpu memory"
```

- 3. Edit the `lsf.shared` file and configure a new Boolean resource named **docker**. For example, copy and paste the following snippet into your `lsf.shared` file:

```
...
Begin Resource
RESOURCENAME TYPE INTERVAL INCREASING DESCRIPTION # Keywords
...
docker Boolean () () (Podman-Docker container)
...
End Resource
```

- 4. Edit the `lsf.cluster` file and attach the **docker** Boolean resource to the LSF server hosts that are running the Podman container engine.
This configuration enables LSF to automatically dispatch Podman jobs to the LSF server hosts that are running the Podman container engine.

```
...
Begin Host
HOSTNAME model type server rlm mem swp RESOURCES
...
host1 ! ! 1 3.5 () () (docker)
```

Configuring LSF to run Podman jobs

Configure the Podman application profile or queue in LSF to run jobs in Podman containers.

About this task

You cannot run pre-execution and post-execution scripts in container jobs. The following are workarounds for specific pre-execution and post-execution operations:

- To prepare data for the container as a pre-execution or post-execution operation, put this data into a directory that is mounted to a job container.
- To customize the internal job container, you can customize the starter scripts to prepare the appropriate environment.

Procedure

1. Edit the `lsb.applications` or `lsb.queues` file and define the `CONTAINER` parameter for the application profile or queue to run Podman jobs:

```
CONTAINER=podman[image(image_name) options(podman_run_options)]
```

If this parameter is specified in both files, the parameter value in the `lsb.applications` file overrides the value in the `lsb.queues` file.

For more details, refer to [the CONTAINER parameter in the lsb.applications file](#) or [the CONTAINER parameter in the lsb.queues file](#).

In the following examples, LSF uses the `centos` image to run the job in the Podman container.

- For sequential jobs:

```
CONTAINER=podman[image(centos) options(--rm)]
```

The container for the job is removed after the job is done, which is enabled with the **podman run --rm** option.

- For parallel jobs:

```
CONTAINER=podman[image(centos) options(--rm --net=host --ipc=host -v  
/path/to/my/passwd:/etc/passwd)]
```

This command uses the following **podman run** options:

--rm

The container for the job is removed after the job is done

--net=host

LSF needs the host network for launching parallel tasks.

-v

LSF needs the user ID and user name for launching parallel tasks.

Note: The `passwd` file must be in the standard format for UNIX and Linux password files, such as the following format:

```
user1:x:10001:10001:::  
user2:x:10002:10002:::
```

2. Edit the `lsb.applications` or `lsb.queues` file and define the `EXEC_DRIVER` parameter for the application profile or queue to run Podman jobs.

For Podman jobs, you must specify `context[user(default)]` as the user account for starting scripts.

If this parameter is specified in both files, the parameter value in the `lsb.applications` file overrides the value in the `lsb.queues` file.

```
EXEC_DRIVER=context[user(default)] \
    starter[/path/to/serverdir/docker-starter.py] \
    controller[/path/to/serverdir/docker-control.py]
```

Replace `/path/to/serverdir` with the actual file path of the `LSF_SERVERDIR` directory.

The file permissions of the starter and controller scripts must be 0755 for Podman jobs.

For more details, refer to [the EXEC_DRIVER parameter in the lsb.applications file](#) or [the EXEC_DRIVER parameter in the lsb.queues file](#).

Submitting Podman jobs to LSF

Use the Podman application profiles or queues to submit Podman jobs to LSF.

About this task

The following job submission (**bsub**) options are not supported for container jobs: -E, -Ep.

Procedure

1. Submit a job to a Podman application profile or queue.
 - Use the **bsub -app** option to submit a job to a Podman application profile (that is, an application profile with a specified value for the `CONTAINER` parameter).
For example, if `CONTAINER` is defined in the `podmanapp` application profile, run the following command to submit a Podman job:

```
bsub -app podmanapp ./myjob.sh
```

- Use the **bsub -q** option to submit a job to a Podman queue (that is, a queue with a specified value for the `CONTAINER` parameter).
For example, if `CONTAINER` is defined in the `podmanq` queue, run the following command to submit a Podman job:

```
bsub -q podmanq ./myjob.sh
```

If you specify both the application profile and the queue in the same job submission, the `CONTAINER` and `EXEC_DRIVER` parameter settings in the application profile take precedence over the same parameter settings in the queue.

For example, if you submit a job with the following command, the parameters in the `podmanapp` application profile override the corresponding parameters in the `podmanq` queue:

```
bsub -app podmanapp -q podmanq ./myjob.sh
```

2. If users are allowed to specify image names for Podman container jobs at job submission time (that is, if the `$LSB_CONTAINER_IMAGE` environment variable is used as the image name when specifying the image keyword), set the `LSB_CONTAINER_IMAGE` environment variable at job submission time to specify the Podman image name. Specify a container image name (such as `centos`) at the job submission time by setting the `LSB_CONTAINER_IMAGE` environment using one of the following methods:
 - Specify the `LSB_CONTAINER_IMAGE` environment variable according to your shell environment. For example, to specify the Podman image named `centos`, set the environment variable as follows:
 - In **csh** or **tcsh**:

```
setenv LSB_CONTAINER_IMAGE centos
```
 - In **sh**, **ksh**, or **bash**:

```
export LSB_CONTAINER_IMAGE=centos
```

- Use the **bsub -env** option. For example, to specify the Podman image named centos
`bsub -env LSB_CONTAINER_IMAGE=centos -app podmanapp a.out -in in.dat -out out.dat`
- Use an **esub** script to set the `LSB_CONTAINER_IMAGE` environment variable, then call the **esub** with the **bsub** command:
 For example, create an **esub.podman** script in the `$LSF_SERVERDIR` directory with the following contents:

```
#!/bin/sh
exec 1>&2
echo "LSB_CONTAINER_IMAGE=\"$1\" >> $LSB_SUB_MODIFY_ENVFILE
```

Submit a job to call the **esub.podman** script by running the following command:

```
bsub -a "docker(centos)" -app podmanapp a.out -in in.dat -out out.dat
```

IBM Spectrum LSF with Enroot

Configure and use LSF to run jobs in Enroot containers on demand. LSF manages the entire lifecycle of jobs that run in the container as common jobs.

Submit Enroot container jobs in LSF in the same way as Docker execution driver jobs.

- [Configuring LSF to run Enroot jobs](#)
 Configure the Enroot application profile or queue in LSF to run jobs in Enroot containers.
- [Submitting Enroot jobs to LSF](#)
 Use the Enroot application profiles or queues to submit Enroot jobs to LSF.

Configuring LSF to run Enroot jobs

Configure the Enroot application profile or queue in LSF to run jobs in Enroot containers.

Before you begin

- The enroot or enroot-hardened package, Version 3.10, or later must be installed and configured correctly on an LSF server host, for example:
 - enroot-3.1.0-1.el7.x86_64 or
 - enroot-hardened-3.1.0-1.el7.x86_64
- The execution driver file for Enroot (enroot-starter.py) is owned by the primary LSF administrator and the file permission is 0755.

About this task

You cannot run pre-execution and post-execution scripts in container jobs. The following are workarounds for specific pre-execution and post-execution operations:

- To prepare data for the container as a pre-execution or post-execution operation, put this data into a directory that is mounted to a job container.
- To customize the internal job container, you can customize the starter scripts to prepare the appropriate environment.

Procedure

1. Edit the `lsb.applications` or `lsb.queues` file and define the `CONTAINER` parameter for the application profile or queue to run Enroot jobs.
 If this parameter is specified in both files, the parameter value in the `lsb.applications` file overrides the value in the `lsb.queues` file.

```
CONTAINER=enroot[image (image_name) options (enroot_start_options) ]
```

For more details, refer to [the CONTAINER parameter in the lsb.applications file](#) or [the CONTAINER parameter in the lsb.queues file](#).

For example, to use the centos image to run the job in the Enroot container:

```
CONTAINER=enroot[image (centos) options (--mount /mydir:/mydir2) ]
```

This command uses the **enroot start --mount** option to perform a mount from the host inside the container.

2. Optional. Edit the lsb.applications or lsb.queues file and define the EXEC_DRIVER parameter for the application profile or queue to run Enroot jobs.

If this parameter is not defined, LSF sets `context[user (default)]` as the user account for starting scripts and sets the driver starter to `$LSF_SERVERDIR/enroot-starter.py` to start the Enroot container job.

If this parameter is specified in both files, the parameter value in the lsb.applications file overrides the value in the lsb.queues file.

```
EXEC_DRIVER=context[user (default)]
starter[/path/to/serverdir/enroot-starter.py]
```

Replace `/path/to/serverdir` with the actual file path of the `LSF_SERVERDIR` directory.

Only the starter script section is used. Other sections, such as context, monitor, and controller are ignored.

For more details, refer to [the EXEC_DRIVER parameter in the lsb.applications file](#) or [the EXEC_DRIVER parameter in the lsb.queues file](#).

Submitting Enroot jobs to LSF

Use the Enroot application profiles or queues to submit Enroot jobs to LSF.

About this task

The following job submission (**bsub**) options are not supported for container jobs: -E, -Ep.

Procedure

1. Submit a job to an Enroot application profile or queue.
 - Use the **bsub -app** option to submit a job to an Enroot application profile (that is, an application profile with a specified value for the CONTAINER parameter).
For example, if CONTAINER is defined in the **enrootapp** application profile, run the following command to submit a Enroot job:
 - Use the **bsub -q** option to submit a job to a Enroot queue (that is, a queue with a specified value for the CONTAINER parameter).
For example, if CONTAINER is defined in the **enrootq** queue, run the following command to submit a Enroot job:

```
bsub -app enrootapp ./myjob.sh
```

```
bsub -q enrootq ./myjob.sh
```

If you specify both the application profile and the queue in the same job submission, the CONTAINER and EXEC_DRIVER parameter settings in the application profile take precedence over the same parameter settings in the queue.

For example, if you submit a job with the following command, the parameters in the **enrootapp** application profile override the corresponding parameters in the **enrootq** queue:

```
bsub -app enrootapp -q enrootq ./myjob.sh
```

2. If users are allowed to specify image names for Enroot container jobs at job submission time (that is, if the `$LSB_CONTAINER_IMAGE` environment variable is used as the image name when specifying the image keyword), set the `LSB_CONTAINER_IMAGE` environment variable at job submission time to specify the Enroot image name. Specify a container image name (such as `centos`) at the job submission time by setting the `LSB_CONTAINER_IMAGE` environment using one of the following methods:

- Specify the `LSB_CONTAINER_IMAGE` environment variable according to your shell environment. For example, to specify the Enroot image named `centos`, set the environment variable as follows:

- In **csh** or **tcsh**:

```
setenv LSB_CONTAINER_IMAGE centos
```

- In **sh**, **ksh**, or **bash**:

```
export LSB_CONTAINER_IMAGE=centos
```

- Use the **bsub -env** option. For example, to specify the Enroot image named `centos`
`bsub -env LSB_CONTAINER_IMAGE=centos -app enrootapp a.out -in in.dat -out out.dat`
- Use an **esub** script to set the `LSB_CONTAINER_IMAGE` environment variable, then call the **esub** with the **bsub** command:

For example, create an **esub.enroot** script in the `$LSF_SERVERDIR` directory with the following contents:

```
#!/bin/sh
exec 1>&2
echo "LSB_CONTAINER_IMAGE=\"$1\" >> $LSB_SUB_MODIFY_ENVFILE
```

Submit a job to call the **esub.enroot** script by running the following command:

```
bsub -a "docker(centos)" -app enrootapp a.out -in in.dat -out out.dat
```

Administering IBM Spectrum LSF high throughput workload

Learn how to submit, monitor, and control high throughput workload in your LSF cluster. Configure scheduling policies that enable efficient queuing, dispatch, and execution of short-running jobs.

- [Job Packs](#)
Use LSF job packs to speed up the submission of a large number of jobs. With job packs, you can submit jobs by submitting a single file containing multiple job requests.
- [Job Arrays](#)
Job arrays are groups of jobs with the same executable and resource requirements, but different input files. Job arrays can be submitted, controlled, and monitored as a single unit or as individual jobs or groups of jobs.
- [Fair share scheduling](#)
fair share scheduling divides the processing power of the LSF cluster among users and queues to provide fair access to resources, so that no user or queue can monopolize the resources of the cluster and no queue will be starved.
- [Guaranteed resource pools](#)
Guaranteed resource pools provide a minimum resource guarantee to a group of users or other consumers.
- [Reserving memory and license resources](#)
Use LSF to reserve memory and license resources for high-throughput workload.

Job Packs

Use LSF job packs to speed up the submission of a large number of jobs. With job packs, you can submit jobs by submitting a single file containing multiple job requests.

Job packs overview

Grouping jobs into packs maintains performance: while LSF is processing a job pack, `mbatchd` is blocked from processing other requests. Limiting the number of jobs in each pack ensures a reasonable response time for other job submissions. Job pack

size is configurable.

If the cluster configuration is not consistent, and LSF receives a job pack that exceeds the job pack size defined in `lsf.conf`, it will be rejected.

The job packs feature supports all **bsub** options in the job submission file except for:

-I -Ip -Is -IS -ISp -ISs -IX -XF -K -jsdl -h -V -pack

About job packs

Enable / disable

Job packs are disabled by default. You must enable the feature before you can run **bsub -pack**.

Job submission rate

Using job packs to submit multiple jobs at once, instead of submitting the jobs individually minimizes system overhead and improves the overall job submission rate.

Job submission file

Create a job submission file that defines each job request. You specify all the **bsub** options individually for each job, so unlike chunk jobs or job arrays, the jobs in this file do not need to have anything in common. To submit the jobs to LSF, you submit the file using the **bsub -pack** option.

Job pack

LSF parses the file contents and submits the job requests, sending multiple requests at one time. Each group of jobs submitted together is called a job pack. The job submission file can contain any number of job requests, and LSF will group them into job packs automatically.

Job request

After the job pack is submitted, each job request in the pack is handled by LSF as if it was submitted individually with the **bsub** command.

For example:

- If `BSUB_CHK_RESREQ` is enabled, LSF checks the syntax of the resource requirement string, instead of scheduling the job.
- If **-is** or **-Zs** is specified, LSF copies the command file to the spool directory, and this may affect the job submission rate.
- The job request cannot be submitted to `mbatchd` if the pending job or slots thresholds have been reached (`MAX_PEND_JOBS` and `MAX_PEND_SLOTS` in `lsb.params` or `lsb.users`).
- If `BSUB_QUIET` is enabled, LSF will not print information about successful job submission.

Job submission errors

By default, if any job request in a file cannot be submitted to `mbatchd`, LSF assumes the job submission file has become corrupt, and does not process any more requests from the file (the jobs already submitted to `mbatchd` successfully do continue to run). Optionally, you can modify the configuration and change this. If you do, LSF processes every request in the file and attempts to submit all the jobs, even if some previous job submissions have failed.

For example, the job submission file may contain job requests from many users, but the default behavior is that LSF stops processing requests after one job fails because the pending job threshold for the user has been reached. If you change the configuration, processing of the job submission file can continue, and job requests from other users can run.

mesub

By default, LSF runs `mesub` as usual for all jobs in the file. Optionally, you can modify configuration and change this. If you do, LSF processes the jobs in the file without running any `mesub`, even if there are `esubs` configured at the application level (`-a` option of **bsub**), or using `LSB_ESUB_METHOD` in `lsf.conf`, or through a named `esub` executable under `LSF_SERVERDIR`.

The `esub` is never executed.

Enable and configure job packs

1. Edit `lsf.conf`.

These parameters will be ignored if defined in the environment instead of the `lsf.conf` file.

2. Define the parameter `LSB_MAX_PACK_JOBS=100`.

Do this to enable the feature and set the job pack size. We recommend 100 as the initial pack size.

If the value is 1, jobs from the file are submitted individually, as if submitted directly using the **bsub** command.

If the value is 0, job packs are disabled.

3. Optionally, define the parameter `LSB_PACK_MESUB=N`.

Do this if you want to further increase the job submission rate by preventing the execution of any mesub during job submission.

This parameter only affects the jobs submitted using job packs, it does not affect jobs submitted in the usual way.

4. Optionally, define the parameter `LSB_PACK_SKIP_ERROR=Y`.

Do this if you want LSF to process all requests in a job submission file, and continue even if some requests have errors.

5. Restart mbatchd to make your changes take effect.

Submit job packs

1. Prepare the job submission file.

Prepare a text file containing all the jobs you want to submit. Each line in the file is one job request. For each request, the syntax is identical to the **bsub** command line (without the word "bsub").

For example:

```
#This file contains 2 job requests.
-R "select[mem>200] rusage[mem=100]" job1.sh
-R "select[swap>400] rusage[swap=200]" job2.sh
#end
```

The job submission file has the following limitations:

- The following **bsub** options are not supported:
-I -Ip -Is -IS -ISp -ISs -IX -XF -K -jsdl -h -V -pack
- Terminal Services jobs are not supported.
- I/O redirection is not supported.
- Blank lines and comment lines (beginning with #) are ignored. Comments at the end of a line are not supported.
- Backslash (\) is NOT considered a special character to join two lines.
- Shell scripting characters are treated as plain text, they will not be interpreted.
- Matched pairs of single and double quotations are supported, but they must have space before and after. For example, **-J "job1"** is supported, **-J'job1'** is not, and **-J "job"1** is not.

For job dependencies, a job name is recommended instead of job ID to specify the dependency condition. A job request will be rejected if the job name or job ID of the job it depends on does not already exist.

2. Submit the job submission file.

Use the **bsub -pack** option to submit all the jobs in a file.

```
bsub -pack job_submission_file
```

where *job_submission_file* is the full path to the job submission file. Do not put any other **bsub** options in the command line, they must be included in each individual job request in the file.

The **-pack** option is not supported in a job script.

Performance metrics

If you enable performance metric collection, every job submitted in a job pack is counted individually, except for the Job submission requests metric. Each job pack counts as just one job submission request.

Job Arrays

Job arrays are groups of jobs with the same executable and resource requirements, but different input files. Job arrays can be submitted, controlled, and monitored as a single unit or as individual jobs or groups of jobs.

Each job submitted from a job array shares the same job ID as the job array and are uniquely referenced using an array index. The dimension and structure of a job array is defined when the job array is created.

Syntax

The **bsub** syntax used to create a job array follows:

```
bsub -J "arrayName[indexList, ...]" myJob
```

Where:

-J "arrayName[indexList, ...]"

Names and creates the job array. The square brackets, **[]**, around **indexList** must be entered exactly as shown and the job array name specification must be enclosed in quotes. Use commas (,) to separate multiple indexList entries. The maximum length of this specification is 255 characters.

arrayName

User specified string that is used to identify the job array. Valid values are any combination of the following characters:

```
a-z | A-Z | 0-9 | . | - | _
```

indexList = start[-end[:step]]

Specifies the size and dimension of the job array, where:

start

Specifies the start of a range of indices. Can also be used to specify an individual index. Valid values are unique positive integers. For example, [1-5] and [1, 2, 3, 4, 5] specify 5 jobs with indices 1 through 5.

end

Specifies the end of a range of indices. Valid values are unique positive integers.

step

Specifies the value to increment the indices in a range. Indices begin at *start*, increment by the value of *step*, and do not increment past the value of *end*. The default value is 1. Valid values are positive integers. For example, [1-10:2] specifies a range of 1-10 with step value 2 creating indices 1, 3, 5, 7, and 9.

After the job array is created (submitted), individual jobs are referenced using the job array name or job ID and an index value. For example, both of the following series of job array statements refer to jobs submitted from a job array named *myArray* which is made up of 1000 jobs and has a job ID of 123:

```
myArray[1], myArray[2], myArray[3], ..., myArray[1000]  
123[1], 123[2], 123[3], ..., 123[1000]
```

- [Creating a job array](#)
- [Input and output files](#)

LSF provides methods for coordinating individual input and output files for the multiple jobs that are created when submitting a job array. These methods require your input files to be prepared uniformly. To accommodate an executable that uses standard input and standard output, LSF provides runtime variables (%I and %J) that are expanded at runtime. To accommodate an executable that reads command-line arguments, LSF provides an environment variable (LSB_JOBINDEX) that is set in the execution environment.
- [Passing arguments on the command line](#)
- [Setting a whole array dependency](#)
- [Controlling job arrays](#)
- [Re-queuing jobs in DONE state](#)
- [Job array job slot limit](#)

The job array job slot limit is used to specify the maximum number of jobs submitted from a job array that is allowed to run at any one time. A job array allows a large number of jobs to be submitted with one command, potentially flooding a system, and job slot limits provide a way to limit the impact a job array may have on a system.

Creating a job array

Procedure

Create a job array at job submission time.

For example, the following command creates a job array named `myArray` made up of 1000 jobs.

```
bsub -J "myArray[1-1000]" myJob  
Job <123> is submitted to default queue <normal>.
```

Changing the maximum size of a job array

About this task

A large job array allows a user to submit a large number of jobs to the system with a single job submission.

By default, the maximum number of jobs in a job array is 1000, which means the maximum size of a job array cannot exceed 1000 jobs.

Procedure

Set `MAX_JOB_ARRAY_SIZE` in `lsb.params` to any positive integer between 1 and 2147483646.

The maximum number of jobs in a job array cannot exceed the value set by `MAX_JOB_ARRAY_SIZE`.

Input and output files

LSF provides methods for coordinating individual input and output files for the multiple jobs that are created when submitting a job array. These methods require your input files to be prepared uniformly. To accommodate an executable that uses standard input and standard output, LSF provides runtime variables (`%I` and `%J`) that are expanded at runtime. To accommodate an executable that reads command-line arguments, LSF provides an environment variable (`LSB_JOBINDEX`) that is set in the execution environment.

- [Preparing input files](#)

Preparing input files

About this task

LSF needs all the input files for the jobs in your job array to be located in the same directory. By default LSF assumes the current working directory (CWD); the directory from where **bsub** was issued.

Procedure

To override CWD, specify an absolute or relative path when submitting the job array.

Each file name consists of two parts, a consistent name string and a variable integer that corresponds directly to an array index. For example, the following file names are valid input file names for a job array. They are made up of the consistent name `input` and integers that correspond to job array indices from 1 to 1000:

```
input.1, input.2, input.3, ..., input.1000
```

Redirecting standard input

About this task

The variables `%I` and `%J` are used as substitution strings to support file redirection for jobs submitted from a job array. At execution time, `%I` is expanded to provide the job array index value of the current job, and `%J` is expanded to provide the job

ID of the job array.

Procedure

Use the **-i** option of **bsub** and the **%I** variable when your executable reads from standard input.

To use **%I**, all the input files must be named consistently with a variable part that corresponds to the indices of the job array. For example:

```
input.1, input.2, input.3, ..., input.N
```

For example, the following command submits a job array of 1000 jobs whose input files are named input.1, input.2, **input.3**, ..., **input.1000** and located in the current working directory:

```
bsub -J "myArray[1-1000]" -i "input.%I" myJob
```

Redirecting standard output and error

Procedure

Use the **-o** option of **bsub** and the **%I** and **%J** variables when your executable writes to standard output and error.

- a. To create an output file that corresponds to each job submitted from a job array, specify **%I** as part of the output file name.

For example, the following command submits a job array of 1000 jobs whose output files are put in CWD and named output.1, output.2, output.3, ..., output.1000:

```
bsub -J "myArray[1-1000]" -o "output.%I" myJob
```

- b. To create output files that include the job array job ID as part of the file name specify **%J**.

For example, the following command submits a job array of 1000 jobs whose output files are put in CWD and named output.123.1, output.123.2, **output.123.3**, ..., **output.123.1000**. The job ID of the job array is 123.

```
bsub -J "myArray[1-1000]" -o "output.%J.%I" myJob
```

Passing arguments on the command line

The environment variable **LSB_JOBINDEX** is used as a substitution string to support passing job array indices on the command line. When the job is dispatched, LSF sets **LSB_JOBINDEX** in the execution environment to the job array index of the current job. **LSB_JOBINDEX** is set for all jobs. For non-array jobs, **LSB_JOBINDEX** is set to zero.

To use **LSB_JOBINDEX**, all the input files must be named consistently and with a variable part that corresponds to the indices of the job array. For example:

```
input.1, input.2, input.3, ..., input.N
```

You must escape **LSB_JOBINDEX** with a backslash, ****, to prevent the shell interpreting **bsub** from expanding the variable. For example, the following command submits a job array of 1000 jobs whose input files are named input.1, input.2, input.3, ..., input.1000 and located in the current working directory. The executable is being passed an argument that specifies the name of the input files:

```
bsub -J "myArray[1-1000]" myJob -f input.\$LSB_JOBINDEX
```

Setting a whole array dependency

About this task

Like all jobs in LSF, a job array can be dependent on the completion or partial completion of a job or another job array. A number of job-array-specific dependency conditions are provided by LSF.

Procedure

To make a job array dependent on the completion of a job or another job array use the `-w "dependency_condition"` option of **bsub**.

For example, to have an array dependent on the completion of a job or job array with job ID 123, use the following command:

```
bsub -w "done(123)" -J "myArray2[1-1000]" myJob
```

Set a partial array dependency

Procedure

1. To make a job or job array dependent on an existing job array, use one of the following dependency conditions.

Condition	Description
numrun(jobArrayJobId, op num)	Evaluate the number of jobs in RUN state
numpend(jobArrayJobId, op num)	Evaluate the number of jobs in PEND state
numdone(jobArrayJobId, op num)	Evaluate the number of jobs in DONE state
numexit(jobArrayJobId, op num)	Evaluate the number of jobs in EXIT state
numended(jobArrayJobId, op num)	Evaluate the number of jobs in DONE and EXIT state
numhold(jobArrayJobId, op num)	Evaluate the number of jobs in PSUSP state
numstart(jobArrayJobId, op num)	Evaluate the number of jobs in RUN and SSUSP and USUSP state

2. Use one the following operators (`op`) combined with a positive integer (`num`) to build a condition:
`== | > | < | >= | <= | !=`

Optionally, an asterisk (*) can be used in place of `num` to mean all jobs submitted from the job array.

For example, to start a job named `myJob` when 100 or more elements in a job array with job ID 123 have completed successfully:

```
bsub -w "numdone(123, >= 100)" myJob
```

Controlling job arrays

About this task

You can control the whole array, all the jobs submitted from the job array, with a single command. LSF also provides the ability to control individual jobs and groups of jobs submitted from a job array. When issuing commands against a job array, use the job array job ID instead of the job array name. Job names are not unique in LSF, and issuing a command using a job array name may result in unpredictable behavior.

Most LSF commands allow operation on both the whole job array, individual jobs, and groups of jobs. These commands include **kill**, **stop**, **resume**, and **mod**.

Some commands only allow operation on individual jobs submitted from a job array. These commands include **top**, **bot**, and **switch**.

Procedure

- Control a whole array
- Control individual jobs
- Control groups of jobs

Control a whole array

Procedure

To control the whole job array, specify the command as you would for a single job using only the job ID. For example, to kill a job array with job ID 123:

```
bkill 123
```

Control individual jobs

Procedure

To control an individual job submitted from a job array, specify the command using the job ID of the job array and the index value of the corresponding job. The job ID and index value must be enclosed in quotes.

For example, to kill the 5th job in a job array with job ID 123:

```
bkill "123[5]"
```

Control groups of jobs

Procedure

To control a group of jobs submitted from a job array, specify the command as you would for an individual job and use **indexList** syntax to indicate the jobs.

For example, to kill jobs 1-5, 239, and 487 in a job array with job ID 123:

```
bkill "123[1-5, 239, 487]"
```

Re-queuing jobs in DONE state

About this task

Use **brequeue** to re-queue a job array. When the job is re-queued, it is assigned the PEND status and the job's new position in the queue is after other jobs of the same priority.

Procedure

To re-queue DONE jobs use the -d option of **brequeue**.

For example, the command **brequeue -J "myarray[1-10]" -d 123** re-queues jobs with job ID 123 and DONE status.

Note: **brequeue** is not supported across clusters.

Requeue Jobs in EXIT state

Procedure

To requeue EXIT jobs use the -e option of **brequeue**.

For example, the command **brequeue -J "myarray[1-10]" -e 123** requeues jobs with job ID 123 and EXIT status.

Re-queuing all jobs in an array regardless of job state

Procedure

A submitted job array can have jobs that have different job states. To re-queue all the jobs in an array regardless of any job's state, use the -a option of **brequeue**.

For example, the command **brequeue -J "myarray[1-10]" -a 123** re-queues all jobs in a job array with job ID 123 regardless of their job state.

Re-queuing RUN jobs to PSUSP state

Procedure

To re-queue RUN jobs to PSUSP state, use the -H option of **brequeue**.

For example, the command **brequeue -J "myarray[1-10]" -H 123** re-queues to PSUSP RUN status jobs with job ID 123.

Re-queuing jobs in RUN state

Procedure

To re-queue RUN jobs use the -r option of **brequeue**.

For example, the command **brequeue -J "myarray[1-10]" -r 123** re-queues jobs with job ID 123 and RUN status.

Job array job slot limit

The job array job slot limit is used to specify the maximum number of jobs submitted from a job array that is allowed to run at any one time. A job array allows a large number of jobs to be submitted with one command, potentially flooding a system, and job slot limits provide a way to limit the impact a job array may have on a system.

Job array job slot limits are specified using the following syntax:

```
bsub -J "job_array_name[index_list]%job_slot_limit" myJob
```

where:

%job_slot_limit

Specifies the maximum number of jobs allowed to run at any one time. The percent sign (%) must be entered exactly as shown. Valid values are positive integers less than the maximum index value of the job array.

If job array limits are defined in a multicluster environment, forwarded jobs (in pending and running states) are counted towards the defined limit.

- [Setting a job array slot limit at submission](#)

Setting a job array slot limit at submission

Procedure

Use the **bsub** command to set a job slot limit at the time of submission.

To set a job array job slot limit of 100 jobs for a job array of 1000 jobs:

```
bsub -J "job_array_name[1000]%100" myJob
```

Setting a job array slot limit after submission

Procedure

Use the **bmod** command to set a job slot limit after submission.

For example, to set a job array job slot limit of 100 jobs for an array with job ID 123:

```
bmod -J "%100" 123
```

Changing a job array job slot limit

About this task

Changing a job array job slot limit is the same as setting it after submission.

Procedure

Use the **bmod** command to change a job slot limit after submission.

For example, to change a job array job slot limit to 250 for a job array with job ID 123:

```
bmod -J "%250" 123
```

Viewing a job array job slot limit

Procedure

To view job array job slot limits use the -A and -l options of **bjobs**. The job array job slot limit is displayed in the Job Name field in the same format in which it was set.

For example, the following output displays the job array job slot limit of 100 for a job array with job ID 123:

```
bjobs -A -l 123
Job <123>, Job Name <myArray[1-1000]%100>, User <user1>, Project <default>, Sta
tus <PEND>, Queue <normal>, Job Priority <20>, Command <my
Job>
Wed Feb 29 12:34:56 2010: Submitted from host <hostA>, CWD <$HOME>;

COUNTERS:
NJOB  PEND  DONE  RUN  EXIT  SSUSP  USUSP  PSUSP
10      9      0      1      0      0      0      0
```

Fair share scheduling

fair share scheduling divides the processing power of the LSF cluster among users and queues to provide fair access to resources, so that no user or queue can monopolize the resources of the cluster and no queue will be starved.

To configure any kind of fair share scheduling, you should understand the following concepts:

- User share assignments
- Dynamic share priority
- Job dispatch order

You can configure fair share at either host level or queue level. If you require more control, you can implement hierarchical fair share. You can also set some additional restrictions when you submit a job.

Note: If you enable the RELAX_JOB_DISPATCH_ORDER parameter in the lsb.params file, which allows LSF to deviate from standard job prioritization policies, LSF might break the job dispatch order as specified by the fair share policy.

- [Fair share scheduling](#)
By default, LSF considers jobs for dispatch in the same order as they appear in the queue (which is not necessarily the order in which they are submitted to the queue). This is called first-come, first-served (FCFS) scheduling.
- [Ways to configure fair share](#)
- [User-based fair share](#)
User-based fair share lets you allocate resources to users in a hierarchical manner.
- [Use time decay and committed run time](#)
By default, as a job is running, the dynamic priority decreases gradually until the job has finished running, then increases immediately when the job finishes.
- [How fair share affects job dispatch order](#)
Within a queue, jobs are dispatched according to the queue's scheduling policy.
- [Host partition user-based fair share](#)
User-based fair share policies that are configured at the host level handle resource contention across multiple queues. You can define a different fair share policy for every host partition. If multiple queues use the host partition, a user has the same priority across multiple queues.

- [Queue-level user-based fair share](#)
User-based fair share policies configured at the queue level handle resource contention among users in the same queue. You can define a different fair share policy for every queue, even if they share the same hosts. A user's priority is calculated separately for each queue.
- [Queue-based fair share](#)
When a priority is set in a queue configuration, a high priority queue tries to dispatch as many jobs as it can before allowing lower priority queues to dispatch any job. Lower priority queues are blocked until the higher priority queue cannot dispatch any more jobs. However, it may be desirable to give some preference to lower priority queues and regulate the flow of jobs from the queue.
- [Slot allocation per queue](#)
- [Users affected by multiple fair share policies](#)
If you belong to multiple user groups, which are controlled by different fair share policies, each group probably has a different dynamic share priority at any given time. By default, if any one of these groups becomes the highest priority user, you could be the highest priority user in that group, and LSF would attempt to place your job.
- [Re-sizable jobs and fair share](#)
Re-sizable jobs submitting into fair share queues or host partitions are subject to fair share scheduling policies. The dynamic priority of the user who submitted the job is the most important criterion. LSF treats pending resize allocation requests as a regular job and enforces the fair share user priority policy to schedule them.
- [Parallel fair share](#)
LSF can consider the number of CPUs when using fair share scheduling with parallel jobs.
- [User share assignments](#)
Both queue-level and host partition fair share use the following syntax to define how shares are assigned to users or user groups
- [Dynamic user priority](#)
LSF calculates a dynamic user priority for individual users or for a group, depending on how the shares are assigned. The priority is dynamic because it changes as soon as any variable in formula changes. By default, a user's dynamic priority gradually decreases after a job starts, and the dynamic priority immediately increases when the job finishes.
- [Cross-queue user-based fair share](#)
User-based fair share policies configured at the queue level handle resource contention across multiple queues.
- [Slot allocation per queue](#)
- [Typical slot allocation scenarios](#)
- [Job count based fair share scheduling](#)
Job count based fair share scheduling uses the number of jobs in the fair share scheduling algorithm instead of the number of job slots.

Fair share scheduling

By default, LSF considers jobs for dispatch in the same order as they appear in the queue (which is not necessarily the order in which they are submitted to the queue). This is called first-come, first-served (FCFS) scheduling.

Fair share scheduling divides the processing power of the LSF cluster among users and queues to provide fair access to resources, so that no user or queue can monopolize the resources of the cluster and no queue will be starved.

If your cluster has many users competing for limited resources, the FCFS policy might not be enough. For example, one user could submit many long jobs at once and monopolize the cluster's resources for a long time, while other users submit urgent jobs that must wait in queues until all the first user's jobs are all done. To prevent this, use fair share scheduling to control how resources should be shared by competing users.

Fair sharing is not necessarily equal sharing: you can assign a higher priority to the most important users. If there are two users competing for resources, you can:

- Give all the resources to the most important user
- Share the resources so the most important user gets the most resources
- Share the resources so that all users have equal importance

Queue-level vs. host partition fair share

You can configure fair share at either the queue level or the host level. However, these types of fair share scheduling are mutually exclusive. You cannot configure queue-level fair share and host partition fair share in the same cluster.

If you want a user's priority in one queue to depend on their activity in another queue, you must use cross-queue fair share or host-level fair share.

Fair share policies

A fair share policy defines the order in which LSF attempts to place jobs that are in a queue or a host partition. You can have multiple fair share policies in a cluster, one for every different queue or host partition. You can also configure some queues or host partitions with fair share scheduling, and leave the rest using FCFS scheduling.

How fair share scheduling works

Each fair share policy assigns a fixed number of shares to each user or group. These shares represent a fraction of the resources that are available in the cluster. The most important users or groups are the ones with the most shares. Users who have no shares cannot run jobs in the queue or host partition.

A user's dynamic priority depends on their share assignment, the dynamic priority formula, and the resources their jobs have already consumed.

The order of jobs in the queue is secondary. The most important thing is the dynamic priority of the user who submitted the job. When fair share scheduling is used, LSF tries to place the first job in the queue that belongs to the user with the highest dynamic priority.

Ways to configure fair share

- [Chargeback fair share](#)
Chargeback fair share lets competing users share the same hardware resources according to a fixed ratio. Each user is entitled to a specified portion of the available resources.
- [Equal share](#)
Equal share balances resource usage equally between users.
- [Priority user and static priority fair share](#)
There are two ways to configure fair share so that a more important user's job always overrides the job of a less important user, regardless of resource use.
- [Host partition fair share](#)
Host partition fair share balances resource usage across the entire cluster according to one single fair share policy. Resources that are used in one queue affect job dispatch order in another queue.
- [GPU runtime fair share](#)
GPU runtime fair share lets you consider GPU run time and historical run time as weighting factors in the dynamic priority calculation.

Chargeback fair share

Chargeback fair share lets competing users share the same hardware resources according to a fixed ratio. Each user is entitled to a specified portion of the available resources.

If two users compete for resources, the most important user is entitled to more resources.

- [Configuring chargeback fair share](#)

Configuring chargeback fair share

Procedure

To configure chargeback fair share, put competing users in separate user groups and assign a fair number of shares to each group.

Example

About this task

Suppose that two departments contributed to the purchase of a large system. The engineering department contributed 70 percent of the cost, and the accounting department 30 percent. Each department wants to get their money's worth from the system.

Procedure

1. Define two user groups in `lsb.users`, one listing all the engineers, and one listing all the accountants.

```
Begin UserGroup
Group_Name    Group_Member
eng_users     (user6 user4)
acct_users    (user2 user5)
End UserGroup
```

2. Configure a host partition for the host, and assign the shares appropriately.

```
Begin HostPartition
HPART_NAME = big_servers
HOSTS = hostH
USER_SHARES = [eng_users, 7] [acct_users, 3]
End HostPartition
```

Equal share

Equal share balances resource usage equally between users.

- [Configuring equal share](#)

Configuring equal share

Procedure

To configure equal share, use the keyword `default` to define an equal share for every user.

```
Begin HostPartition
HPART_NAME = equal_share_partition
HOSTS = all
USER_SHARES = [default, 1]
End HostPartition
```

Priority user and static priority fair share

There are two ways to configure fair share so that a more important user's job always overrides the job of a less important user, regardless of resource use.

Priority user fair share

Dynamic priority is calculated as usual, but more important and less important users are assigned a drastically different number of shares, so that resource use has virtually no effect on the dynamic priority: the user with the overwhelming majority of shares always goes first. However, if two users have a similar or equal number of shares, their resource use still determines which of them goes first. This is useful for isolating a group of high-priority or low-priority users, while allowing other fair share policies to operate as usual most of the time.

Static priority fair share

Dynamic priority is no longer dynamic because resource use is ignored. The user with the most shares always goes first. This is useful to configure multiple users in a descending order of priority.

- [Configuring priority user fair share](#)
A queue is shared by key users and other users. Priority user fair sharing gives priority to important users, so their jobs override the jobs of other users. You can still use fair share policies to balance resources among each group of users.
- [Configuring static priority fair share](#)
Static priority fair share assigns resources to the user with the most shares. Resource usage is ignored.

Configuring priority user fair share

A queue is shared by key users and other users. Priority user fair sharing gives priority to important users, so their jobs override the jobs of other users. You can still use fair share policies to balance resources among each group of users.

About this task

If two users compete for resources, and one of them is a priority user, the priority user's job always runs first.

Procedure

1. Define a user group for priority users in `lsb.users`, naming it accordingly.
For example, `key_users`.
2. Configure fair share and assign the overwhelming majority of shares to the key users:

```
Begin Queue
QUEUE_NAME = production
FAIRSHARE = USER_SHARES[[key_users@, 2000] [others, 1]]
...
End Queue
```

In the preceding example, key users have 2000 shares each, while other users together have only one share. This makes it virtually impossible for other users' jobs to get dispatched unless none of the users in the `key_users` group has jobs waiting to run.

If you want the same fair share policy to apply to jobs from all queues, configure host partition fair share in a similar way.

Configuring static priority fair share

Static priority fair share assigns resources to the user with the most shares. Resource usage is ignored.

Procedure

To implement static priority fair share, edit the `lsb.params` file and set all the weighting factors that are used in the dynamic priority formula to 0 (zero):

- Set `CPU_TIME_FACTOR` to 0
- Set `RUN_TIME_FACTOR` to 0

- Set RUN_JOB_FACTOR to 0
- Set COMMITTED_RUN_TIME_FACTOR to 0
- Set FAIRSHARE_ADJUSTMENT_FACTOR to 0
- Set GPU_RUN_TIME_FACTOR to 0

This results in $\text{dynamic priority} = \text{number_shares} / 0.01$; that is, if the denominator in the dynamic priority calculation is less than 0.01, IBM® Spectrum LSF rounds up to 0.01.

Results

If two users compete for resources, the most important user's job always runs first.

Host partition fair share

Host partition fair share balances resource usage across the entire cluster according to one single fair share policy. Resources that are used in one queue affect job dispatch order in another queue.

If two users compete for resources, their dynamic share priority is the same in every queue.

- [Configuring host partition fair share](#)

Configuring host partition fair share

Procedure

Use the keyword all to configure a single partition that includes all the hosts in the cluster.

```
Begin HostPartition
HPART_NAME =GlobalPartition
HOSTS = all
USER_SHARES = [groupA@, 3] [groupB, 7] [default, 1]
End HostPartition
```

GPU runtime fair share

GPU runtime fair share lets you consider GPU run time and historical run time as weighting factors in the dynamic priority calculation.

- [Configuring GPU run time](#)

Configuring GPU run time

Procedure

1. Set a value for the GPU_RUN_TIME_FACTOR parameter for the queue in lsb.queues or for the cluster in lsb.params.
2. To enable historical GPU run time of finished jobs, specify `ENABLE_GPU_HIST_RUN_TIME=Y` for the queue in lsb.queues or for the cluster in lsb.params.
Enabling historical GPU time ensures that the user's priority does not increase significantly after a GPU job finishes.

Results

If you set the GPU run time factor and enabled the use of GPU historical run time, the dynamic priority is calculated according to the following formula:

$$\text{dynamic priority} = \text{number_shares} / (\text{cpu_time} * \text{CPU_TIME_FACTOR} + (\text{historical_run_time} + \text{run_time}) * \text{RUN_TIME_FACTOR} + (\text{committed_run_time} - \text{run_time}) * \text{COMMITTED_RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR} + \text{fairshare_adjustment}(\text{struct} * \text{shareAdjustPair}) * \text{FAIRSHARE_ADJUSTMENT_FACTOR}) + ((\text{historical_gpu_run_time} + \text{gpu_run_time}) * \text{ngpus_physical}) * \text{GPU_RUN_TIME_FACTOR}$$

gpu_run_time—The run time requested at GPU job submission with the **-gpu** option of **bsub**, the queue or application profile configuration with the GPU_REQ parameter, or the cluster configuration with the LSB_GPU_REQ parameter.

- For jobs that ask for exclusive use of a GPU, *gpu_run_time* is the same as the job's run time and *ngpus_physical* is the value of the requested *ngpus_physical* in the job's effective RES_REQ string.
- For jobs that ask for an exclusive host (with the **bsub -x** option), the *gpu_run_time* is the same as the job's run time and *ngpus_physical* is the number of GPUs on the execution host.
- For jobs that ask for an exclusive compute unit (**bsub -R "cu[excl]"** option), the *gpu_run_time* is the same as the job's run time and *ngpus_physical* is the number of GPUs or all the execution hosts in the compute unit.
- For jobs that ask for shared mode GPUs, the *gpu_run_time* is the job's run time divided by the number of jobs that are using the GPU, and *ngpus_physical* is the value of the requested *ngpus_physical* in the job's effective RES_REQ string.

historical_run_time— If ENABLE_GPU_HIST_RUN_TIME is defined in the lsb.params file, the *historical_run_time* is the same as the job's run time (measured in hours) of finished GPU jobs, and a decay factor from time to time based on HIST_HOURS in the lsb.params file (5 hours by default).

User-based fair share

User-based fair share lets you allocate resources to users in a hierarchical manner.

By default, when shares are assigned to a group, group members compete for resources according to FCFS policy. If you use hierarchical fair share, you control the way shares that are assigned collectively are divided among group members. If groups have subgroups, you can configure additional levels of share assignments, resulting in a multi-level share tree that becomes part of the fair share policy.

How hierarchical user-based fair share affects dynamic share priority

When you use hierarchical fair share, the dynamic share priority formula does not change, but LSF measures the resource consumption for all levels of the share tree. To calculate the dynamic priority of a group, LSF uses the resource consumption of all the jobs in the queue or host partition that belong to users in the group and all its subgroups, recursively.

How hierarchical user-based fair share affects job dispatch order

LSF uses the dynamic share priority of a user or group to find out which user's job to run next. If you use hierarchical fair share, LSF works through the share tree from the top level down, and compares the dynamic priority of users and groups at each level until the user with the highest dynamic priority is a single user, or a group that has no subgroups.

- [Configure hierarchical fair share](#)
To define a hierarchical fair share policy, configure the top-level share assignment in lsb.queues or lsb.hosts, as usual. Then, for any group of users affected by the fair share policy, configure a share tree in the **UserGroup** section of lsb.users. This specifies how shares assigned to the group, collectively, are distributed among the individual users or subgroups.
- [Configuring a share tree](#)
- [User share assignments](#)
Both queue-level and host partition fair share use the following syntax to define how shares are assigned to users or user groups

- [Dynamic user priority](#)

LSF calculates a dynamic user priority for individual users or for a group, depending on how the shares are assigned. The priority is dynamic because it changes as soon as any variable in formula changes. By default, a user's dynamic priority gradually decreases after a job starts, and the dynamic priority immediately increases when the job finishes.

Configure hierarchical fair share

To define a hierarchical fair share policy, configure the top-level share assignment in `lsb.queues` or `lsb.hosts`, as usual. Then, for any group of users affected by the fair share policy, configure a share tree in the **UserGroup** section of `lsb.users`. This specifies how shares assigned to the group, collectively, are distributed among the individual users or subgroups.

If shares are assigned to members of any group individually, using `@`, there can be no further hierarchical fair share within that group. The shares are assigned recursively to all members of all subgroups, regardless of further share distributions defined in `lsb.users`. The group members and members of all subgroups compete for resources according to FCFS policy.

You can choose to define a hierarchical share tree for some groups but not others. If you do not define a share tree for any group or subgroup, members compete for resources according to FCFS policy.

Configuring a share tree

Procedure

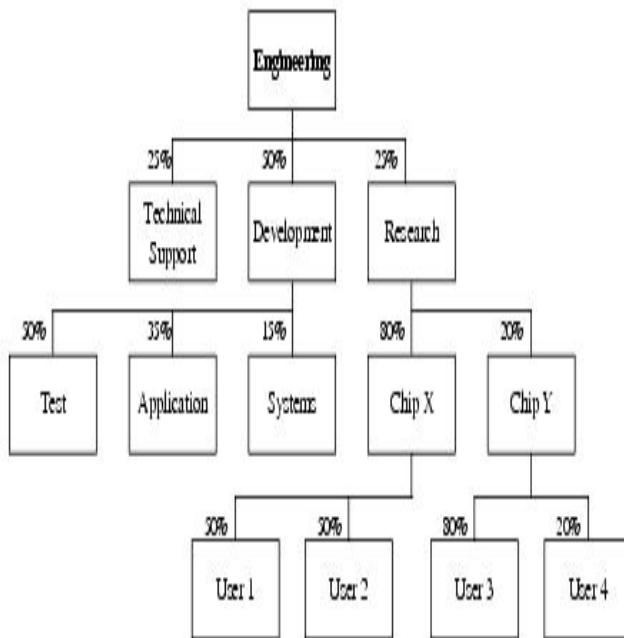
Group membership is already defined in the **UserGroup** section of `lsb.users`. To configure a share tree, use the **USER_SHARES** column to describe how the shares are distributed in a hierarchical manner. Use the following format.

```
Begin UserGroup
GROUP_NAME    GROUP_MEMBER    USER_SHARES
GroupB        (User1 User2)      ()
GroupC        (User3 User4)      ([User3, 3] [User4, 4])
GroupA        (GroupB GroupC User5) ([User5, 1] [default, 10])
```

- User groups must be defined before they can be used (in the **GROUP_MEMBER** column) to define other groups.
- Shares (in the **USER_SHARES** column) can only be assigned to user groups in the **GROUP_MEMBER** column.
- The keyword `all` refers to all users, not all user groups.
- Enclose the share assignment list in parentheses, as shown, even if you do not specify any user share assignments.

Example

An **Engineering** queue or host partition organizes users hierarchically, and divides the shares as shown; the actual number of shares assigned at each level do not matter:



The **Development** group gets the largest share (50%) of the resources in the event of contention. Shares that are assigned to the **Development** group can be further divided among the **Systems**, **Application**, and **Test** groups, which receive 15%, 35%, and 50%, respectively. At the lowest level, individual users compete for these shares as usual.

One way to measure a user's importance is to multiply their percentage of the resources at every level of the share tree. For example, **User1** is entitled to 10% of the available resources ($.50 \times .80 \times .25 = .10$) and **User3** is entitled to 4% ($.80 \times .20 \times .25 = .04$). However, if **Research** has the highest dynamic share priority among the 3 groups at the top level, and **ChipY** has a higher dynamic priority than **ChipX**, the next comparison is between **User3** and **User4**, so the importance of **User1** is not relevant. The dynamic priority of **User1** is not even calculated at this point.

User share assignments

Both queue-level and host partition fair share use the following syntax to define how shares are assigned to users or user groups

Syntax

[*user*, *number_shares*]

Enclose each user share assignment in square brackets, as shown. Separate multiple share assignments with a space between each set of square brackets.

user

Specify users of the queue or host partition. You can assign the shares:

- to a single user (specify *user_name*)
- to users in a group, individually (specify *group_name@*) or collectively (specify *group_name*)
- to users not included in any other share assignment, individually (specify the keyword *default*) or collectively (specify the keyword *others*)

By default, when resources are assigned collectively to a group, the group members compete for the resources according to FCFS scheduling. You can use hierarchical fair share to further divide the shares among the group members.

When resources are assigned to members of a group individually, the share assignment is recursive. Members of the group and of all subgroups always compete for the resources according to FCFS scheduling, regardless of hierarchical

fair share policies.

number_shares

Specify a positive integer representing the number of shares of cluster resources assigned to the user.

The number of shares assigned to each user is only meaningful when you compare it to the shares assigned to other users, or to the total number of shares. The total number of shares is just the sum of all the shares assigned in each share assignment.

Examples

```
[User1, 1] [GroupB, 1]
```

Assigns two shares: 1 to `User1`, and 1 to be shared by the users in `GroupB`. Each user in `GroupB` has equal importance. `User1` is as important as all the users in `GroupB` put together. In this example, it does not matter if the number of shares is one, six, or 600. As long as `User1` and `GroupB` are both assigned the same number of shares, the relationship stays the same.

```
[User1, 10] [GroupB@, 1]
```

If `GroupB` contains ten users, assigns 20 shares in total: ten to `User1`, and 1 to each user in `GroupB`. Each user in `GroupB` has equal importance. `User1` is ten times as important as any user in `GroupB`.

```
[User1, 10] [User2, 9] [others, 8]
```

Assigns 27 shares: ten to `User1`, none to `User2`, and eight to the remaining users, as a group. `User1` is slightly more important than `User2`. Each of the remaining users has equal importance.

- If there are three users in total, the single remaining user has all eight shares, and is almost as important as `User1` and `User2`.
- If there are 12 users in total, then ten users compete for those eight shares, and each of them is significantly less important than `User1` and `User2`.

```
[User1, 10] [User2, 6] [default, 4]
```

The relative percentage of shares held by a user will change, depending on the number of users who are granted shares by default.

- If there are three users in total, assigns 20 shares: ten to `User1`, six to `User2`, and four to the remaining user. `User1` has half of the available resources (ten shares out of 20).
- If there are 12 users in total, assigns 56 shares: ten to `User1`, six to `User2`, and 4 to each of the remaining ten users. `User1` has about a fifth of the available resources (ten shares out of 56).

Dynamic user priority

LSF calculates a dynamic user priority for individual users or for a group, depending on how the shares are assigned. The priority is dynamic because it changes as soon as any variable in formula changes. By default, a user's dynamic priority gradually decreases after a job starts, and the dynamic priority immediately increases when the job finishes.

How LSF calculates dynamic priority

By default, LSF calculates the dynamic priority for each user based on the following criteria:

- The number of shares assigned to the user
- The resources used by jobs belonging to the user:
 - Number of job slots reserved and in use
 - Run time of running jobs
 - Cumulative actual CPU time (not normalized), adjusted so that recently used CPU time is weighted more heavily than CPU time used in the distant past

If you enable additional functionality, the formula can also involve additional resources used by jobs belonging to the user:

- Decayed run time of running jobs
- Historical run time of finished jobs
- Committed run time, specified at job submission with the **-W** option of **bsub**, or in the queue with the RUNLIMIT parameter in lsb.queues
- Memory usage adjustment made by the fair share plug-in (libfairshareadjust.*).

How LSF measures fair share resource usage

LSF resource usage differently, depending on the type of fair share:

- For user-based fair share:
 - For queue-level fair share, LSF measures the resource consumption of all the user's jobs in the queue. This means a user's dynamic priority can be different in every queue.
 - For host partition fair share, LSF measures resource consumption for all the user's jobs that run on hosts in the host partition. This means a user's dynamic priority is the same in every queue that uses hosts in the same partition.
- For queue-based fair share, LSF measures the resource consumption of all jobs in each queue.

Default dynamic priority formula

By default, LSF calculates dynamic priority according to the following formula:

$$\text{dynamic priority} = \text{number_shares} / (\text{cpu_time} * \text{CPU_TIME_FACTOR} + \text{run_time} * \text{RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR} + (1 + \text{fwd_job_slots}) * \text{FWD_JOB_FACTOR} + \text{fairshare_adjustment} * \text{FAIRSHARE_ADJUSTMENT_FACTOR}) + ((\text{historical_gpu_run_time} + \text{gpu_run_time}) * \text{ngpus_physical}) * \text{GPU_RUN_TIME_FACTOR}$$

Note: The maximum value of dynamic user priority is 100 times the number of user shares (if the denominator in the calculation is less than 0.01, LSF rounds up to 0.01).

For *cpu_time*, *run_time*, and *job_slots*, LSF uses the total resource consumption of all the jobs in the queue or host partition that belong to the user or group.

number_shares

The number of shares assigned to the user.

cpu_time

The cumulative CPU time used by the user (measured in hours). LSF calculates the cumulative CPU time using the actual (not normalized) CPU time and a decay factor such that 1 hour of recently-used CPU time decays to 0.1 hours after an interval of time specified by HIST_HOURS in lsb.params (5 hours by default).

run_time

The total run time of running jobs (measured in hours).

job_slots

The number of job slots reserved and in use.

fairshare_adjustment

The adjustment calculated by the fair share adjustment plug-in (libfairshareadjust.*).

Configure the default dynamic priority

You can give additional weight to the various factors in the priority calculation by setting the following parameters for the queue in lsb.queues or for the cluster in lsb.params. When the queue value is not defined, the cluster-wide value from lsb.params is used.

- CPU_TIME_FACTOR
- RUN_TIME_FACTOR
- RUN_JOB_FACTOR
- FWD_JOB_FACTOR

- FAIRSHARE_ADJUSTMENT_FACTOR
- HIST_HOURS
- GPU_RUN_TIME_FACTOR

If you modify the parameters used in the dynamic priority formula, it affects every fair share policy in the cluster:

CPU_TIME_FACTOR

The CPU time weighting factor.
Default: 0.7

FWD_JOB_FACTOR

The forwarded job slots weighting factor when using the LSF multicluster capability.
Default: Not defined

RUN_TIME_FACTOR

The run time weighting factor.
Default: 0.7

RUN_JOB_FACTOR

The job slots weighting factor.
Default: 3

FAIRSHARE_ADJUSTMENT_FACTOR

The fair share plug-in (libfairshareadjust.*) weighting factor.
Default: 0

HIST_HOURS

Interval for collecting resource consumption history.
Default: 5

GPU_RUN_TIME_FACTOR

GPU run time weighting factor.
Default: 0

Customize the dynamic priority

In some cases the dynamic priority equation may require adjustments beyond the run time, CPU time, and job slot dependencies provided by default. The fair share adjustment plug-in is open source and can be customized once you identify specific requirements for dynamic priority.

All information used by the default priority equation (except the user shares) is passed to the fair share plug-in. In addition, the fair share plug-in is provided with current memory use over the entire cluster and the average memory that is allocated to a slot in the cluster.

Note: If you modify the parameters used in the dynamic priority formula, it affects every fair share policy in the cluster. The fair share adjustment plug-in (libfairshareadjust.*) is not queue-specific. Parameter settings passed to the fair share adjustment plug-in are those defined in lsb.params.

Example

Jobs assigned to a single slot on a host can consume host memory to the point that other slots on the hosts are left unusable. The default dynamic priority calculation considers job slots used, but doesn't account for unused job slots effectively blocked by another job.

The fair share adjustment plug-in example code provided by LSF is found in the examples directory of your installation, and implements a memory-based dynamic priority adjustment as follows:

```
fair share adjustment= (1+slots)*((used_memory /used_slots)/(slot_memory*THRESHOLD))
```

used_slots

The number of job slots in use by started jobs.

used_memory

The total memory in use by started jobs.

slot_memory

The average amount of memory that exists per slot in the cluster.

THRESHOLD

The memory threshold set in the fair share adjustment plug-in.

Use time decay and committed run time

By default, as a job is running, the dynamic priority decreases gradually until the job has finished running, then increases immediately when the job finishes.

In some cases this can interfere with fair share scheduling if two users who have the same priority and the same number of shares submit jobs at the same time.

To avoid these problems, you can modify the dynamic priority calculation by using one or more of the following weighting factors:

- Run time decay
- Historical run time decay
- Committed run time
- [Historical run time decay](#)
By default, historical run time does not affect the dynamic priority. You can configure LSF so that the user's dynamic priority increases *gradually* after a job finishes. After a job is finished, its run time is saved as the historical run time of the job and the value can be used in calculating the dynamic priority, the same way LSF considers historical CPU time in calculating priority. LSF applies a decaying algorithm to the historical run time to gradually increase the dynamic priority over time after a job finishes.
- [Run time decay](#)
In a cluster running jobs of varied length, a user running only short jobs may always have a higher priority than a user running a long job. This can happen when historical run time decay is applied, decreasing the impact of the completed short jobs but not the longer job that is still running. To correct this, you can configure LSF to decay the run time of a job that is still running in the same manner historical run time decays.
- [Committed run time weighting factor](#)
The committed run time is the run time requested at job submission with the **-W** option for the **bsub** command, or in the queue configuration with the RUNLIMIT parameter. By default, committed run time does not affect the dynamic priority.

Historical run time decay

By default, historical run time does not affect the dynamic priority. You can configure LSF so that the user's dynamic priority increases *gradually* after a job finishes. After a job is finished, its run time is saved as the historical run time of the job and the value can be used in calculating the dynamic priority, the same way LSF considers historical CPU time in calculating priority. LSF applies a decaying algorithm to the historical run time to gradually increase the dynamic priority over time after a job finishes.

- [Configuring historical run time](#)
- [How mbatchd reconfiguration and restart affects historical run time](#)
After restarting or reconfiguring the **mbatchd** daemon, the historical run time of finished jobs might be different, since it includes jobs that may have been cleaned from **mbatchd** before the restart. **mbatchd** restart only reads recently finished jobs from **lsb.events**, according to the value of **CLEAN_PERIOD** in **lsb.params**. Any jobs cleaned before restart are lost and are not included in the new calculation of the dynamic priority.

Configuring historical run time

Procedure

Specify `ENABLE_HIST_RUN_TIME=Y` for the queue in `lsb.queues` or for the cluster in `lsb.params`.
Historical run time is added to the calculation of the dynamic priority so that the formula becomes the following:

dynamic priority = *number_shares* / *A*

where *A* is (*cpu_time* * `CPU_TIME_FACTOR` + *run_time* * *number_CPUs* * `RUN_TIME_FACTOR` + (1 + *job_slots*) * `RUN_JOB_FACTOR` + *fairshare_adjustment*(*struc* * *shareAdjustPair*) * `FAIRSHARE_ADJUSTMENT_FACTOR` + (*historical_gpu_run_time* + *gpu_run_time*) * *ngpus_physical* * `GPU_RUN_TIME_FACTOR`)

historical_run_time—(measured in hours) of finished jobs accumulated in the user's share account file. LSF calculates the historical run time using the actual run time of finished jobs and a decay factor such that 1 hour of recently-used run time decays to 0.1 hours after an interval of time specified by `HIST_HOURS` in `lsb.params` (5 hours by default).

How `mbatchd` reconfiguration and restart affects historical run time

After restarting or reconfiguring the `mbatchd` daemon, the historical run time of finished jobs might be different, since it includes jobs that may have been cleaned from `mbatchd` before the restart. `mbatchd` restart only reads recently finished jobs from `lsb.events`, according to the value of `CLEAN_PERIOD` in `lsb.params`. Any jobs cleaned before restart are lost and are not included in the new calculation of the dynamic priority.

Example

The following fair share parameters are configured in `lsb.params`:

```
CPU_TIME_FACTOR = 0
RUN_JOB_FACTOR  = 0
RUN_TIME_FACTOR = 1
FAIRSHARE_ADJUSTMENT_FACTOR = 0
```

Note that in this configuration, only run time is considered in the calculation of dynamic priority. This simplifies the formula to the following:

dynamic priority = *number_shares* / (*run_time* * `RUN_TIME_FACTOR`)

Without the historical run time, the dynamic priority increases suddenly as soon as the job finishes running because the run time becomes zero, which gives no chance for jobs pending for other users to start.

When historical run time is included in the priority calculation, the formula becomes:

dynamic priority = *number_shares* / (*historical_run_time* + *run_time*) * `RUN_TIME_FACTOR`)

Now the dynamic priority increases gradually as the historical run time decays over time.

Run time decay

In a cluster running jobs of varied length, a user running only short jobs may always have a higher priority than a user running a long job. This can happen when historical run time decay is applied, decreasing the impact of the completed short jobs but not the longer job that is still running. To correct this, you can configure LSF to decay the run time of a job that is still running in the same manner historical run time decays.

Once a job is complete, the decayed run time is transferred to the historical run time where the decay continues. This equalizes the effect of short and long running jobs on user dynamic priority.

Note: Running `badadmin reconfig` or restarting `mbatchd` during a job's run time results in the decayed run time being recalculated. When a suspended job using run time decay is resumed, the decay time is based on the elapsed time.

- [Configuring run time decay](#)

Configuring run time decay

Procedure

1. Specify `HIST_HOURS` for the queue in `lsb.queues` or for the cluster in `lsb.params`.
2. Specify `RUN_TIME_DECAY=Y` for the queue in `lsb.queues` or for the cluster in `lsb.params`.
The run time used in the calculation of the dynamic priority so that the formula becomes the following:

$$\text{dynamic priority} = \text{number_shares} / A$$

where *A* is:

$$(\text{cpu_time} * \text{CPU_TIME_FACTOR} + \text{run_time} * \text{number_CPUs} * \text{RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR} + \text{fair_share_adjustment}(\text{struc} * \text{shareAdjustPair}) * \text{FAIRSHARE_ADJUSTMENT_FACTOR} + (\text{historical_gpu_run_time} + \text{gpu_run_time}) * \text{ngpus_physical} * \text{GPU_RUN_TIME_FACTOR})$$

Committed run time weighting factor

The committed run time is the run time requested at job submission with the **-W** option for the **bsub** command, or in the queue configuration with the `RUNLIMIT` parameter. By default, committed run time does not affect the dynamic priority.

While the job is running, the actual run time is subtracted from the committed run time. The user's dynamic priority decreases immediately to its lowest expected value, and is maintained at that value until the job finishes. Job run time is accumulated as usual, and historical run time, if any, is decayed.

When the job finishes, the committed run time is set to zero and the actual run time is added to the historical run time for future use. The dynamic priority increases gradually until it reaches its maximum value.

Providing a weighting factor in the run time portion of the dynamic priority calculation prevents a job dispatching burst, where one user monopolizes job slots because of the latency in computing run time.

Limitation

If you use queue-level fair share, and a running job has a committed run time, you should not switch that job to or from a fair share queue (using **bswitch**). The fair share calculations will not be correct.

Run time displayed by **bqueues** and **bhpart**

The run time displayed by **bqueues** and **bhpart** is the sum of the actual, accumulated run time and the historical run time, but does not include the committed run time.

- [Configuring committed run time](#)

Configuring committed run time

Procedure

Set a value for the `COMMITTED_RUN_TIME_FACTOR` parameter for the queue in `lsb.queues` or for the cluster in `lsb.params`. You should also specify a `RUN_TIME_FACTOR`, to prevent the user's dynamic priority from increasing as the run time increases.

If you have also enabled the use of historical run time, the dynamic priority is calculated according to the following formula:

```
dynamic priority = number_shares / (cpu_time * CPU_TIME_FACTOR + (historical_run_time +
run_time) * RUN_TIME_FACTOR + (committed_run_time - run_time) * COMMITTED_RUN_TIME_FACTOR + (1
+ job_slots) * RUN_JOB_FACTOR + fair_share_adjustment(struct*
shareAdjustPair)*FAIRSHARE_ADJUSTMENT_FACTOR) + ((historical_gpu_run_time + gpu_run_time) *
ngpus_physical) * GPU_RUN_TIME_FACTOR
```

where *committed_run_time* is the run time requested at job submission with the **-W** option of **bsub**, or in the queue configuration with the RUNLIMIT parameter. This calculation measures the committed run time in hours.

In the calculation of a user's dynamic priority, COMMITTED_RUN_TIME_FACTOR determines the relative importance of the committed run time in the calculation. If the **-W** option of **bsub** is not specified at job submission and a RUNLIMIT has not been set for the queue, the committed run time is not considered.

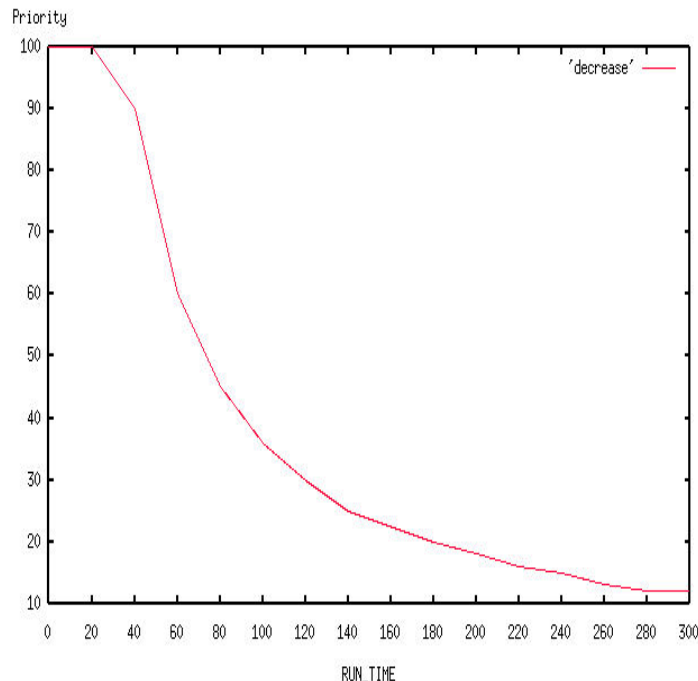
COMMITTED_RUN_TIME_FACTOR can be any positive value between 0.0 and 1.0. The default value set in lsb.params is 0.0. As the value of COMMITTED_RUN_TIME_FACTOR approaches 1.0, more weight is given to the committed run time in the calculation of the dynamic priority.

Example

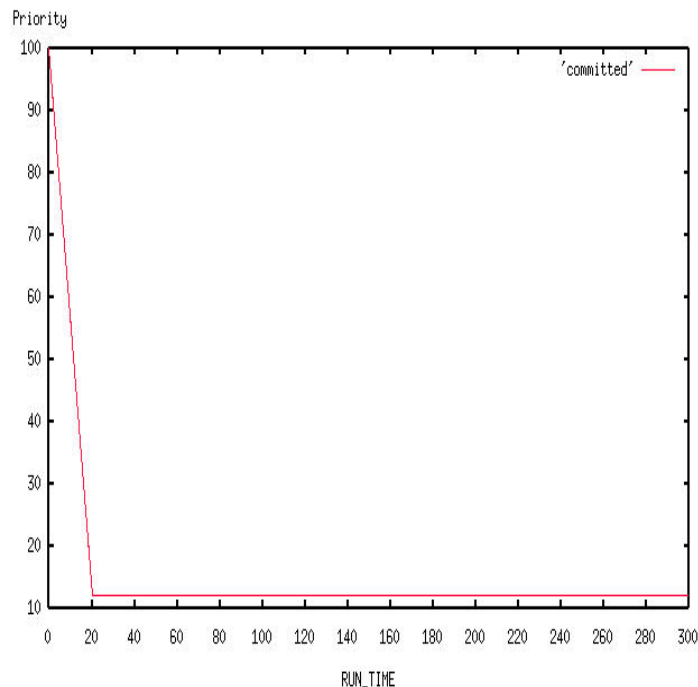
The following fair share parameters are configured in lsb.params:

```
CPU_TIME_FACTOR = 0
RUN_JOB_FACTOR = 0
RUN_TIME_FACTOR = 1
FAIRSHARE_ADJUSTMENT_FACTOR = 0
GPU_RUN_TIME_FACTOR = 0
COMMITTED_RUN_TIME_FACTOR = 1
```

Without a committed run time factor, dynamic priority for the job owner drops gradually while a job is running:



When a committed run time factor is included in the priority calculation, the dynamic priority drops as soon as the job is dispatched, rather than gradually dropping as the job runs:



How fair share affects job dispatch order

Within a queue, jobs are dispatched according to the queue's scheduling policy.

- For FCFS queues, the dispatch order depends on the order of jobs in the queue (which depends on job priority and submission time, and can also be modified by the job owner).
- For fair share queues, the dispatch order depends on dynamic share priority, then order of jobs in the queue (which is not necessarily the order in which they are submitted to the queue).

A user's priority gets higher when they use less than their fair share of the cluster's resources. When a user has the highest priority, LSF considers one of their jobs first, even if other users are ahead of them in the queue.

If there are only one user's jobs pending, and you do not use hierarchical fair share, then there is no resource contention between users, so the fair share policies have no effect and jobs are dispatched as usual.

Job dispatch order among queues of equivalent priority

The order of dispatch depends on the order of the queues in the queue configuration file. The first queue in the list is the first to be scheduled.

Jobs in a fair share queue are always considered as a group, so the scheduler attempts to place all jobs in the queue before beginning to schedule the next queue.

Jobs in an FCFS queue are always scheduled along with jobs from other FCFS queues of the same priority (as if all the jobs belonged to the same queue).

Example

In a cluster, queues A, B, and C are configured in that order and have equal queue priority.

Jobs with equal job priority are submitted to each queue in this order: C, B, A, B, and A.

- If all queues are FCFS queues, order of dispatch is C, B, A, B, and A (queue A is first; queues B and C are the same priority as A; all jobs are scheduled in FCFS order).

- If all queues are fair share queues, order of dispatch is AA, BB, and C (queue A is first; all jobs in the queue are scheduled; then queue B, then C).
- If A and C are fair share, and B is FCFS, order of dispatch is AA, B, B, and C (queue A jobs are scheduled according to user priority; then queue B jobs are scheduled in FCFS order; then queue C jobs are scheduled according to user priority)
- If A and C are FCFS, and B is fair share, order of dispatch is C, A, A, and BB (queue A is first; queue A and C jobs are scheduled in FCFS order, then queue B jobs are scheduled according to user priority)
- If any of these queues uses cross-queue fair share, the other queues must also use cross-queue fair share and belong to the same set, or they cannot have the same queue priority.

Host partition user-based fair share

User-based fair share policies that are configured at the host level handle resource contention across multiple queues. You can define a different fair share policy for every host partition. If multiple queues use the host partition, a user has the same priority across multiple queues.

To run a job on a host that has fair share, users must have a share assignment (USER_SHARES in the **HostPartition** section of `lsb.hosts`). Even cluster administrators cannot submit jobs to a fair share host if they do not have a share assignment.

- [Configuring host partition fair share scheduling](#)

Configuring host partition fair share scheduling

Procedure

To configure host partition fair share, define a host partition in `lsb.hosts`. Use the following format:

```
Begin HostPartition
HPART_NAME = Partition1
HOSTS = hostA hostB ~hostC
USER_SHARES = [groupA@, 3] [groupB, 7] [default, 1]
End HostPartition
```

- A host cannot belong to multiple partitions.
- Optional: Use the reserved host name *all* to configure a single partition that applies to all hosts in a cluster.
- Optional: Use the not operator (~) to exclude hosts or host groups from the list of hosts in the host partition.
- Hosts in a host partition cannot participate in queue-based fair share.
Hosts that are not included in any host partition are controlled by FCFS scheduling policy instead of fair share scheduling policy.

Queue-level user-based fair share

User-based fair share policies configured at the queue level handle resource contention among users in the same queue. You can define a different fair share policy for every queue, even if they share the same hosts. A user's priority is calculated separately for each queue.

To submit jobs to a fair share queue, users must be allowed to use the queue (USERS in the `lsb.queues` file) and must have a share assignment (a value of **fairshare** in the `lsb.queues` file). Even cluster and queue administrators cannot submit jobs to a fair share queue if they do not have a share assignment.

If the default user group set in DEFAULT_USER_GROUP (in the `lsb.params` file) does not have shares assigned in a fair share queue, jobs can still run from the default user group, and are charged to the highest priority account the user can access in the queue. The default user group should have shares assigned in most fair share queues to ensure jobs run smoothly.

Job submitted with a user group (**bsub -G**) which is no longer valid when the job runs charge the default user group (if defined) or the highest priority account the user can access in the queue (if no default user group is defined). In such cases **bjobs -l** output shows the submission user group, along with the updated SAAP (share attribute account path).

By default, user share accounts are created for users in each user group, whether they have active jobs or not. When many user groups in the fair share policy have all as a member, the memory used creating user share accounts on startup of the **mbatchd** daemon can be noticeable. Limit the number of share accounts created to active users (and all members of the default user group) by setting `LSB_SACCT_ONE_UG=Y` in the `lsf.conf` file.

- [Configuring queue-level fair share](#)
 - [Cross-queue user-based fair share](#)
- User-based fair share policies configured at the queue level handle resource contention across multiple queues.

Configuring queue-level fair share

Procedure

To configure a fair share queue, define fair share in `lsb.queues` and specify a share assignment for all users of the queue:

```
fair share = USER_SHARES[[user, number_shares]...]
```

- You must specify at least one user share assignment.
- Enclose the list in square brackets, as shown.
- Enclose each user share assignment in square brackets, as shown.

Cross-queue user-based fair share

User-based fair share policies configured at the queue level handle resource contention across multiple queues.

Apply the same fair share policy to several queues

With cross-queue fair share, the same user-based fair share policy can apply to several queues at the same time. You define the fair share policy in a *parent_queue* and list *child_queues* to which the same fair share policy applies; child queues inherit the same fair share policy as your parent queue. For job scheduling purposes, this is equivalent to having one queue with one fair share tree.

In this way, if a user submits jobs to different queues, user priority is calculated by taking into account all the jobs the user has submitted across the defined queues.

To submit jobs to a fair share queue, users must be allowed to use the queue (`USERS` in `lsb.queues`) and must have a share assignment (`FAIRSHARE` in `lsb.queues`). Even cluster and queue administrators cannot submit jobs to a fair share queue if they do not have a share assignment.

User and queue priority

By default, a user has the same priority across the parent and child queues. If the same user submits several jobs to these queues, user priority is calculated by taking into account all the jobs the user has submitted across the parent-child set.

If `DISPATCH_ORDER=QUEUE` is set in the parent queue, jobs are dispatched according to queue priorities first, then user priority. This avoids having users with higher fair share priority getting jobs dispatched from low-priority queues.

Jobs from users with lower fair share priorities who have pending jobs in higher priority queues are dispatched before jobs in lower priority queues. Jobs in queues having the same priority are dispatched according to user priority.

Queues that are not part of the ordered cross-queue fair share can have any priority. Their priority can fall within the priority range of cross-queue fair share queues and they can be inserted between two queues using the same fair share tree.

- [Configuring cross-queue fair share](#)
- [Control job dispatch order in cross-queue fair share](#)

Configuring cross-queue fair share

About this task

- FAIRSHARE must be defined in the parent queue. If it is also defined in the queues that are listed in FAIRSHARE_QUEUES, it will be ignored.
- Cross-queue fair sharing can be defined more than once within lsb.queues. You can define several sets of parent-child queues. However, a queue cannot belong to more than one parent-child set. For example, you can define:
 - In parent queue `normal`: **FAIRSHARE_QUEUES=short**
 - In parent queue `priority`: **FAIRSHARE_QUEUES= night owners**
 You cannot, however, define `night`, `owners`, or `priority` as children in the `normal` queue; or `normal`, `short` as children in the `priority` queue; or `short`, `night`, `owners` as parent queues of their own.
- Cross-queue fair sharing cannot be used with host partition fair sharing. It is part of queue-level fair sharing.

Procedure

1. Decide to which queues in your cluster cross-queue fair sharing will apply.
For example, in your cluster you may have the queues `normal`, `priority`, `short`, and you want cross-queue fair sharing to apply only to `normal`, and `short`.

2. Define fair share policies in your parent queue.

In the queue you want to be the parent, for example `normal`, define the following in `lsb.queues`:

- FAIRSHARE and specify a share assignment for all users of the queue.
- FAIRSHARE_QUEUES and list child queues to which the defined fair share policy will also apply
- PRIORITY to indicate the priority of the queue.

```
Begin Queue
QUEUE_NAME      = queue1
PRIORITY        = 30
NICE             = 20
FAIRSHARE       = USER_SHARES[[user1,100] [default,1]]
FAIRSHARE_QUEUES = queue2 queue3
DESCRIPTION     = For normal low priority jobs, running only if hosts are lightly loaded.
End Queue
```

3. In all the child queues listed in FAIRSHARE_QUEUES, define all queue values as desired.
For example:

```
Begin Queue
QUEUE_NAME      = queue2
PRIORITY        = 40
NICE            = 20
UJOB_LIMIT      = 4
PJOB_LIMIT      = 2
End Queue

Begin Queue
QUEUE_NAME      = queue3
PRIORITY        = 50
NICE            = 10
PREEMPTION      = PREEMPTIVE
QJOB_LIMIT      = 10
UJOB_LIMIT      = 1
PJOB_LIMIT      = 1
End Queue
```

Control job dispatch order in cross-queue fair share

DISPATCH_ORDER parameter (lsb.queues)

Use DISPATCH_ORDER=QUEUE in the parent queue to define an ordered cross-queue fair share set. DISPATCH_ORDER indicates that jobs are dispatched according to the order of queue priorities, not user fair share priority.

Priority range in cross-queue fair share

By default, the range of priority defined for queues in cross-queue fair share cannot be used with any other queues. The priority of queues that are not part of the cross-queue fair share cannot fall between the priority range of cross-queue fair share queues.

For example, you have four queues: `queue1`, `queue2`, `queue3`, and `queue4`. You configure cross-queue fair share for `queue1`, `queue2`, and `queue3`, and assign priorities of 30, 40, 50 respectively. The priority of `queue4` (which is not part of the cross-queue fair share) cannot fall between 30 and 50, but it can be any number up to 29 or higher than 50. It does not matter if `queue4` is a fair share queue or FCFS queue.

If DISPATCH_ORDER=QUEUE is set in the parent queue, queues that are not part of the ordered cross-queue fair share can have any priority. Their priority can fall within the priority range of cross-queue fair share queues and they can be inserted between two queues using the same fair share tree. In the example above, `queue4` can have any priority, including a priority falling between the priority range of the cross-queue fair share queues (30-50).

Jobs from equal priority queues

- If two or more non-fair share queues have the same priority, their jobs are dispatched first-come, first-served based on submission time or job ID as if they come from the same queue.
- If two or more fair share queues have the same priority, jobs are dispatched in the order the queues are listed in `lsb.queues`.

Queue-based fair share

When a priority is set in a queue configuration, a high priority queue tries to dispatch as many jobs as it can before allowing lower priority queues to dispatch any job. Lower priority queues are blocked until the higher priority queue cannot dispatch any more jobs. However, it may be desirable to give some preference to lower priority queues and regulate the flow of jobs from the queue.

Queue-based fair sharing allows flexible slot allocation per queue as an alternative to absolute queue priorities by enforcing a soft job slot limit on a queue. This allows you to organize the priorities of your work and tune the number of jobs dispatched from a queue so that no single queue monopolizes cluster resources, leaving other queues waiting to dispatch jobs.

You can balance the distribution of job slots among queues by configuring a ratio of jobs waiting to be dispatched from each queue. LSF then attempts to dispatch a certain percentage of jobs from each queue, and does not attempt to drain the highest priority queue entirely first.

When queues compete, the allocated slots per queue are kept within the limits of the configured share. If only one queue in the pool has jobs, that queue can use all the available resources and can span its usage across all hosts it could potentially run jobs on.

Manage pools of queues

You can configure your queues into a pool, which is a named group of queues using the same set of hosts. A pool is entitled to a slice of the available job slots. You can configure as many pools as you need, but each pool must use the same set of hosts. There can be queues in the cluster that do not belong to any pool yet share some hosts that are used by a pool.

How LSF allocates slots for a pool of queues

During job scheduling, LSF orders the queues within each pool based on the shares the queues are entitled to. The number of running jobs (or job slots in use) is maintained at the percentage level that is specified for the queue. When a queue has no pending jobs, leftover slots are redistributed to other queues in the pool with jobs pending.

The total number of slots in each pool is constant; it is equal to the number of slots in use plus the number of free slots to the maximum job slot limit configured either in `lsb.hosts (MAX)` or in `lsb.resources` for a host or host group. The accumulation of slots in use by the queue is used in ordering the queues for dispatch.

Job limits and host limits are enforced by the scheduler. For example, if LSF determines that a queue is eligible to run 50 jobs, but the queue has a job limit of 40 jobs, no more than 40 jobs will run. The remaining 10 job slots are redistributed among other queues belonging to the same pool, or make them available to other queues that are configured to use them.

Accumulated slots in use

As queues run the jobs allocated to them, LSF accumulates the slots each queue has used and decays this value over time, so that each queue is not allocated more slots than it deserves, and other queues in the pool have a chance to run their share of jobs.

Interaction with other scheduling policies

- Queues participating in a queue-based fair share pool cannot be preemptive or pre-emptable.
- You should not configure slot reservation (`SLOT_RESERVE`) in queues that use queue-based fair share.
- Cross-queue user-based fair share (`FAIRSHARE_QUEUES`) can undo the dispatching decisions of queue-based fair share. Cross-queue user-based fair share queues should not be part of a queue-based fair share pool.
- When `SLOT_RESERVE` and `BACKFILL` are defined (in `lsb.queues`) for the same queue, jobs in the queue cannot backfill using slots reserved by other jobs in the same queue.

Examples

Three queues using two hosts each with maximum job slot limit of 6 for a total of 12 slots to be allocated:

- `queue1` shares 50% of slots to be allocated = $2 * 6 * 0.5 = 6$
slots
- `queue2` shares 30% of slots to be allocated = $2 * 6 * 0.3 = 3.6 = 4$
slots
- `queue3` shares 20% of slots to be allocated = $2 * 6 * 0.2 = 2.4 = 3$
slots; however, since the total cannot be more than 12, `queue3` is actually allocated only 2 slots.

Four queues using two hosts each with maximum job slot limit of 6 for a total of 12 slots; `queue4` does not belong to any pool:

- `queue1` shares 50% of slots to be allocated = $2 * 6 * 0.5 = 6$
- `queue2` shares 30% of slots to be allocated = $2 * 6 * 0.3 = 3.6 = 4$
- `queue3` shares 20% of slots to be allocated = $2 * 6 * 0.2 = 2.4 = 2$
- `queue4` shares no slots with other queues

`queue4` causes the total number of slots to be less than the total free and in use by the `queue1`, `queue2`, and `queue3` that do belong to the pool. It is possible that the pool may get all its shares used up by `queue4`, and jobs from the pool will remain pending.

`queue1`, `queue2`, and `queue3` belong to one pool, `queue6`, `queue7`, and `queue8` belong to another pool, and `queue4` and `queue5` do not belong to any pool:

LSF orders the queues in the two pools from higher-priority queue to lower-priority queue (`queue1` is highest and `queue8` is lowest):

`queue1 -> queue2 -> queue3 -> queue6 -> queue7 -> queue8`

If the queue belongs to a pool, jobs are dispatched from the highest priority queue first. Queues that do not belong to any pool (`queue4` and `queue5`) are merged into this ordered list according to their priority, but LSF dispatches as many

jobs from the non-pool queues as it can:

```
queue1 -> queue2 -> queue3 -> queue4 -> queue5 -> queue6 -> queue7 -> queue8
```

Slot allocation per queue

Configure as many pools as you need in `lsb.queues`.

Note: This feature is deprecated and might be removed in a future version of LSF.

SLOT_SHARE parameter

The `SLOT_SHARE` parameter represents the percentage of running jobs (job slots) in use from the queue. `SLOT_SHARE` must be greater than zero and less than or equal to 100.

The sum of `SLOT_SHARE` for all queues in the pool does not need to be 100%. It can be more or less, depending on your needs.

SLOT_POOL parameter

The `SLOT_POOL` parameter is the name of the pool of job slots the queue belongs to. A queue can only belong to one pool. All queues in the pool must share the same set of hosts.

MAX_SLOTS_IN_POOL parameter

The optional parameter `MAX_SLOTS_IN_POOL` sets a limit on the number of slots available for a slot pool. This parameter is defined in the first queue of the slot pool in `lsb.queues`.

USE_PRIORITY_IN_POOL parameter

The optional parameter `USE_PRIORITY_IN_POOL` enables LSF scheduling to allocate any unused slots in the pool to jobs based on the job priority across the queues in the slot pool. This parameter is defined in the first queue of the slot pool in `lsb.queues`.

Host job slot limit

The hosts that are used by the pool must have a maximum job slot limit, configured either in `lsb.hosts` (`MXJ`) or `lsb.resources` (`HOSTS` and `SLOTS`).

- [Configuring slot allocation per queue](#)
- [Typical slot allocation scenarios](#)
- [Configuring slot allocation per queue](#)

Configuring slot allocation per queue

Procedure

1. For each queue that uses queue-based fair share, define the following in `lsb.queues`:
 - a. `SLOT_SHARE`
 - b. `SLOT_POOL`
2. Optional: Define the following in `lsb.queues` for each queue that uses queue-based fair share:
 - a. `HOSTS` to list the hosts that can receive jobs from the queue

If no hosts are defined for the queue, the default is all hosts.

Tip: Hosts for queue-based fair share cannot be in a host partition.

b. PRIORITY to indicate the priority of the queue.

3. For each host used by the pool, define a maximum job slot limit, either in lsb.hosts (MXJ) or lsb.resources (HOSTS and SLOTS).

Configure two pools

The following example configures pool **A** with three queues, with different shares, using the hosts in host group **groupA**:

```
Begin Queue
QUEUE_NAME = queue1
PRIORITY   = 50
SLOT_POOL  = poolA
SLOT_SHARE = 50
HOSTS      = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue2
PRIORITY   = 48
SLOT_POOL  = poolA
SLOT_SHARE = 30
HOSTS      = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue3
PRIORITY   = 46
SLOT_POOL  = poolA
SLOT_SHARE = 20
HOSTS      = groupA
...
End Queue
```

The following configures a pool named **poolB**, with three queues with equal shares, using the hosts in host group **groupB**, setting a maximum number of slots for the pool (MAX_SLOTS_IN_POOL) and enabling a second round of scheduling based on job priority across the queues in the pool (USE_PRIORITY_IN_POOL):

```
Begin Queue
QUEUE_NAME = queue4
PRIORITY   = 44
SLOT_POOL  = poolB
SLOT_SHARE = 30
HOSTS      = groupB
MAX_SLOTS_IN_POOL=128
USE_PRIORITY_IN_POOL=Y
...
End Queue

Begin Queue
QUEUE_NAME = queue5
PRIORITY   = 43
SLOT_POOL  = poolB
SLOT_SHARE = 30
HOSTS      = groupB
...
End Queue

Begin Queue
QUEUE_NAME = queue6
PRIORITY   = 42
SLOT_POOL  = poolB
SLOT_SHARE = 30
HOSTS      = groupB
...
End Queue
```

Typical slot allocation scenarios

3 queues with SLOT_SHARE 50%, 30%, 20%, with 15 job slots

This scenario has three phases:

1. All three queues have jobs running, and LSF assigns the number of slots to queues as expected: 8, 5, 2. Though queue *Genova* deserves 3 slots, the total slot assignment must be 15, so *Genova* is allocated only 2 slots:

bqueues											
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
Roma	50	Open:Active	-	-	-	-	1000	992	8	0	
Verona	48	Open:Active	-	-	-	-	995	990	5	0	
Genova	48	Open:Active	-	-	-	-	996	994	2	0	

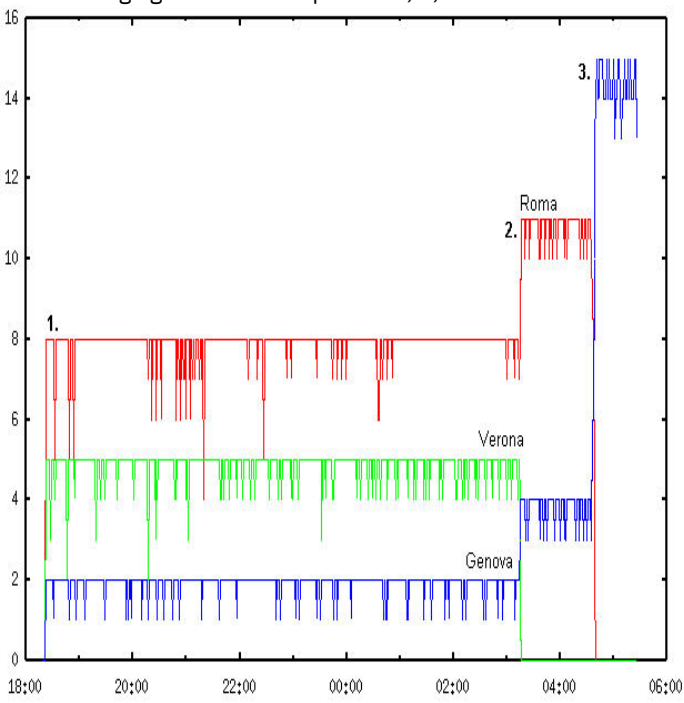
2. When queue *Verona* has done its work, queues *Roma* and *Genova* get their respective shares of 8 and 3. This leaves 4 slots to be redistributed to queues according to their shares: 50% (2 slots) to *Roma*, 20% (1 slot) to *Genova*. The one remaining slot is assigned to queue *Roma* again:

bqueues											
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
Roma	50	Open:Active	-	-	-	-	231	221	11	0	
Verona	48	Open:Active	-	-	-	-	0	0	0	0	
Genova	48	Open:Active	-	-	-	-	496	491	4	0	

3. When queues *Roma* and *Verona* have no more work to do, *Genova* can use all the available slots in the cluster:

bqueues											
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
Roma	50	Open:Active	-	-	-	-	0	0	0	0	
Verona	48	Open:Active	-	-	-	-	0	0	0	0	
Genova	48	Open:Active	-	-	-	-	475	460	15	0	

The following figure illustrates phases 1, 2, and 3:



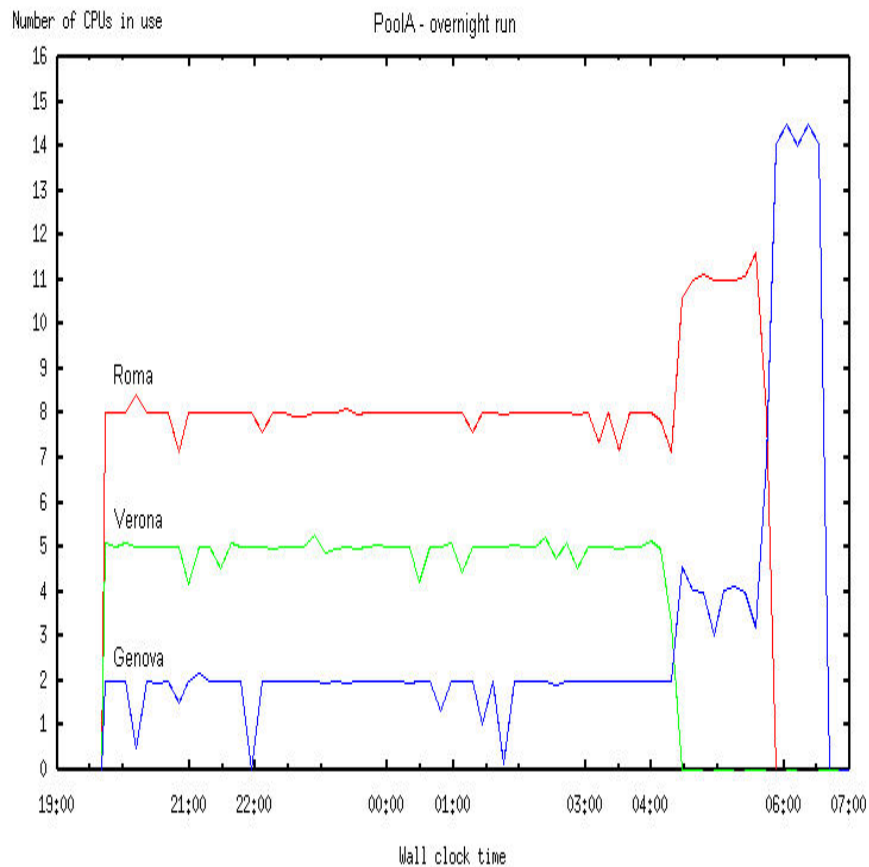
2 pools, 30 job slots, and 2 queues out of any pool

- *poolA* uses 15 slots and contains queues *Roma* (50% share, 8 slots), *Verona* (30% share, 5 slots), and *Genova* (20% share, 2 remaining slots to total 15).

- poolB with 15 slots containing queues Pisa (30% share, 5 slots), Venezia (30% share, 5 slots), and Bologna (30% share, 5 slots).
- Two other queues Milano and Parma do not belong to any pool, but they can use the hosts of poolB. The queues from Milano to Bologna all have the same priority.

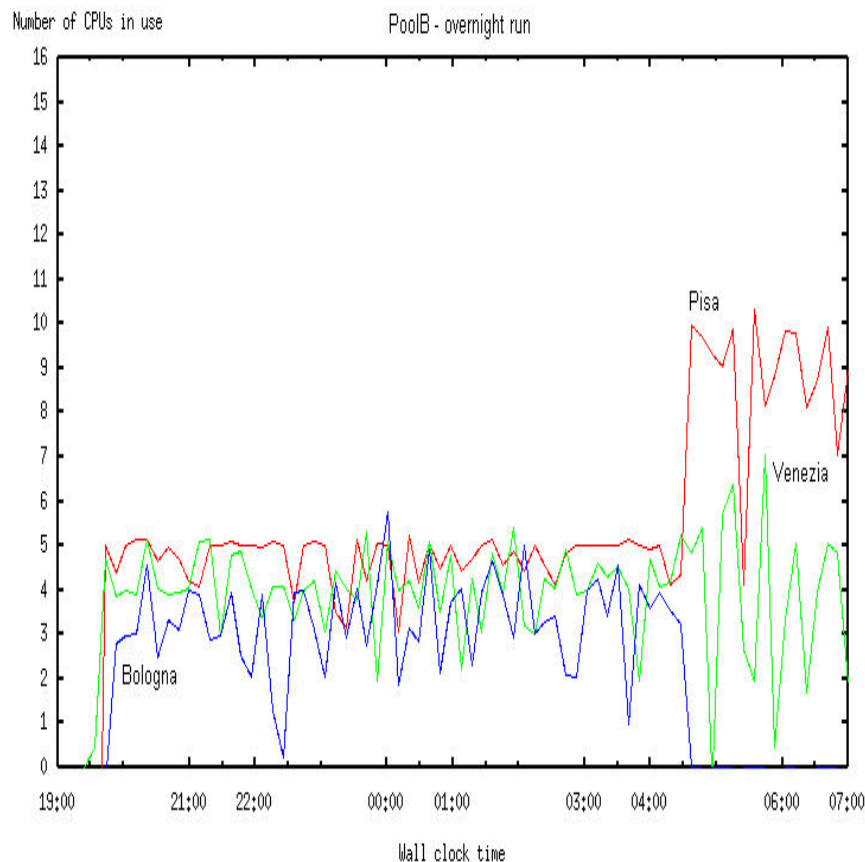
The queues Milano and Parma run very short jobs that get submitted periodically in bursts. When no jobs are running in them, the distribution of jobs looks like this:

QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open:Active	-	-	-	-	1000	992	8	0
Verona	48	Open:Active	-	-	-	-	1000	995	5	0
Genova	48	Open:Active	-	-	-	-	1000	998	2	0
Pisa	44	Open:Active	-	-	-	-	1000	995	5	0
Milano	43	Open:Active	-	-	-	-	2	2	0	0
Parma	43	Open:Active	-	-	-	-	2	2	0	0
Venezia	43	Open:Active	-	-	-	-	1000	995	5	0
Bologna	43	Open:Active	-	-	-	-	1000	995	5	0



When Milano and Parma have jobs, their higher priority reduces the share of slots free and in use by Venezia and Bologna:

QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open:Active	-	-	-	-	992	984	8	0
Verona	48	Open:Active	-	-	-	-	993	990	3	0
Genova	48	Open:Active	-	-	-	-	996	994	2	0
Pisa	44	Open:Active	-	-	-	-	995	990	5	0
Milano	43	Open:Active	-	-	-	-	10	7	3	0
Parma	43	Open:Active	-	-	-	-	11	8	3	0
Venezia	43	Open:Active	-	-	-	-	995	995	2	0
Bologna	43	Open:Active	-	-	-	-	995	995	2	0



Round-robin slot distribution: 13 queues and 2 pools

- Pool `poolA` has 3 hosts each with 7 slots for a total of 21 slots to be shared. The first 3 queues are part of the pool `poolA` sharing the CPUs with proportions 50% (11 slots), 30% (7 slots) and 20% (3 remaining slots to total 21 slots).
- The other 10 queues belong to pool `poolB`, which has 3 hosts each with 7 slots for a total of 21 slots to be shared. Each queue has 10% of the pool (3 slots).

The initial slot distribution looks like this:

bqueues											
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
Roma	50	Open:Active	-	-	-	-	15	6	11	0	
Verona	48	Open:Active	-	-	-	-	25	18	7	0	
Genova	47	Open:Active	-	-	-	-	460	455	3	0	
Pisa	44	Open:Active	-	-	-	-	264	261	3	0	
Milano	43	Open:Active	-	-	-	-	262	259	3	0	
Parma	42	Open:Active	-	-	-	-	260	257	3	0	
Bologna	40	Open:Active	-	-	-	-	260	257	3	0	
Sora	40	Open:Active	-	-	-	-	261	258	3	0	
Ferrara	40	Open:Active	-	-	-	-	258	255	3	0	
Napoli	40	Open:Active	-	-	-	-	259	256	3	0	
Livorno	40	Open:Active	-	-	-	-	258	258	0	0	
Palermo	40	Open:Active	-	-	-	-	256	256	0	0	
Venezia	4	Open:Active	-	-	-	-	255	255	0	0	

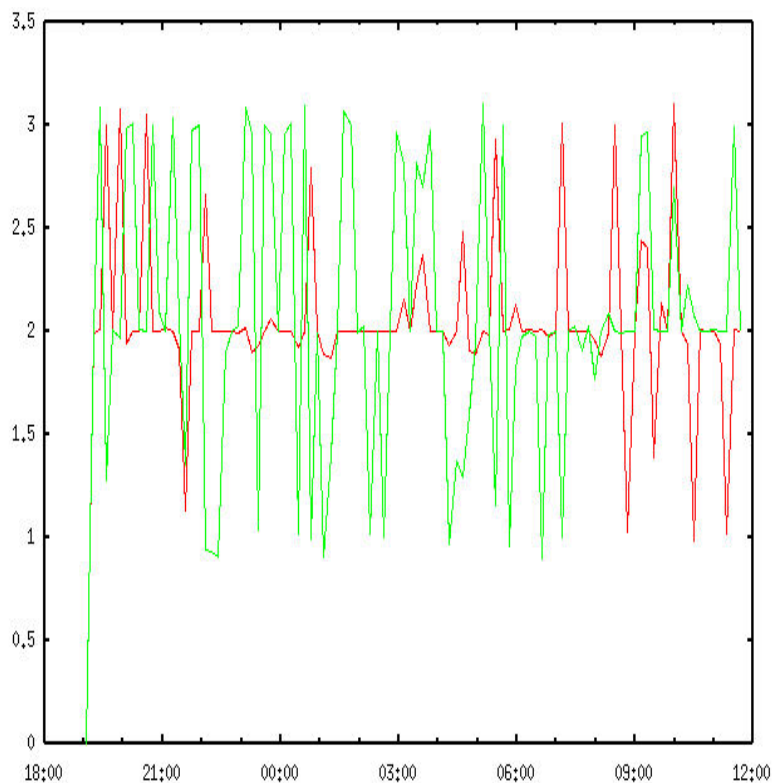
Initially, queues `Livorno`, `Palermo`, and `Venezia` in `poolB` are not assigned any slots because the first 7 higher priority queues have used all 21 slots available for allocation.

As jobs run and each queue accumulates used slots, LSF favors queues that have not run jobs yet. As jobs finish in the first 7 queues of `poolB`, slots are redistributed to the other queues that originally had no jobs (queues `Livorno`, `Palermo`, and `Venezia`). The total slot count remains 21 in all queues in `poolB`.

bqueues											
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	

Roma	50	Open:Active	-	-	-	-	15	6	9	0
Verona	48	Open:Active	-	-	-	-	25	18	7	0
Genova	47	Open:Active	-	-	-	-	460	455	5	0
Pisa	44	Open:Active	-	-	-	-	263	261	2	0
Milano	43	Open:Active	-	-	-	-	261	259	2	0
Parma	42	Open:Active	-	-	-	-	259	257	2	0
Bologna	40	Open:Active	-	-	-	-	259	257	2	0
Sora	40	Open:Active	-	-	-	-	260	258	2	0
Ferrara	40	Open:Active	-	-	-	-	257	255	2	0
Napoli	40	Open:Active	-	-	-	-	258	256	2	0
Livorno	40	Open:Active	-	-	-	-	258	256	2	0
Palermo	40	Open:Active	-	-	-	-	256	253	3	0
Venezia	4	Open:Active	-	-	-	-	255	253	2	0

The following figure illustrates the round-robin distribution of slot allocations between queues Livorno and Palermo:



How LSF rebalances slot usage

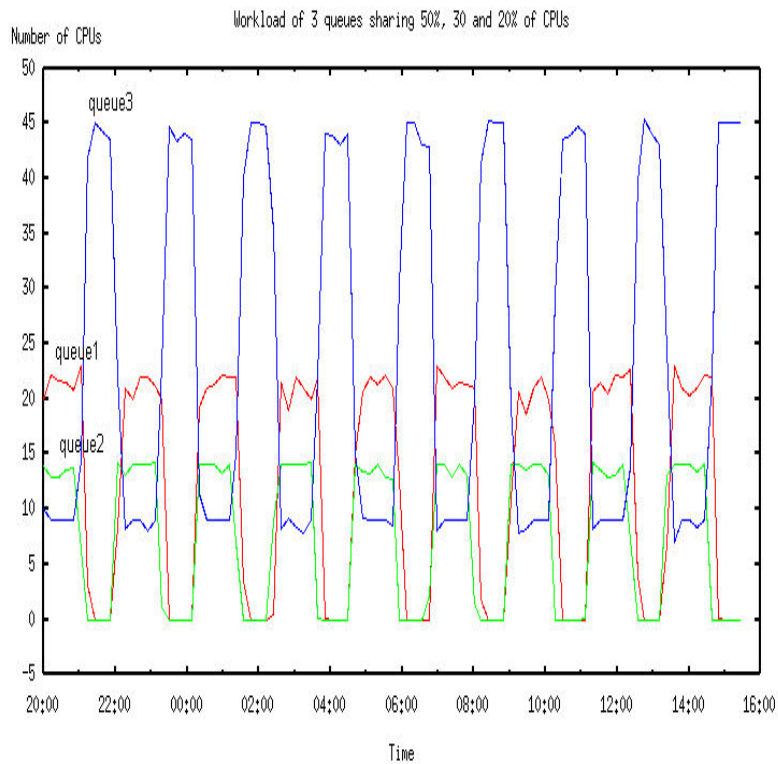
In the following examples, job runtime is not equal, but varies randomly over time.

3 queues in one pool with 50%, 30%, 20% shares

A pool configures 3 queues:

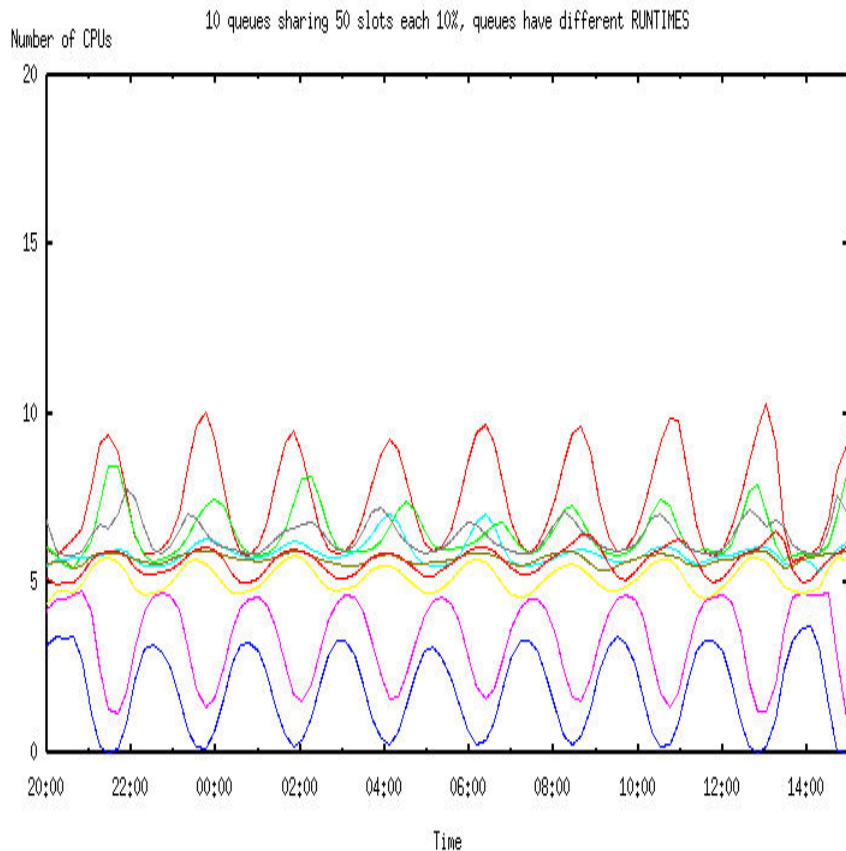
- queue1 50% with short-running jobs
- queue2 20% with short-running jobs
- queue3 30% with longer running jobs

As queue1 and queue2 finish their jobs, the number of jobs in queue3 expands, and as queue1 and queue2 get more work, LSF rebalances the usage:



10 queues sharing 10% each of 50 slots

In this example, *queue1* (the curve with the highest peaks) has the longer running jobs and so has less accumulated slots in use over time. LSF accordingly rebalances the load when all queues compete for jobs to maintain a configured 10% usage share.



Users affected by multiple fair share policies

If you belong to multiple user groups, which are controlled by different fair share policies, each group probably has a different dynamic share priority at any given time. By default, if any one of these groups becomes the highest priority user, you could be the highest priority user in that group, and LSF would attempt to place your job.

To restrict the number of fair share policies that will affect your job, submit your job and specify a single user group that your job will belong to, for the purposes of fair share scheduling. LSF will not attempt to dispatch this job unless the group you specified is the highest priority user. If you become the highest priority user because of some other share assignment, another one of your jobs might be dispatched, but not this one.

- [Submitting a job and specify a user group](#)
Associate a job with a user group for fair share scheduling.

Submitting a job and specify a user group

Associate a job with a user group for fair share scheduling.

Procedure

Use **bsub -G** and specify a group that you belong to.

For example, `User1` shares resources with `groupA` and `groupB`. `User1` is also a member of `groupA`, but not any other groups.

`User1` submits a job: **bsub sleep 100**.

By default, the job could be considered for dispatch if either `User1` or `GroupA` has highest dynamic share priority.

`User1` submits a job and associates the job with `GroupA`:

```
bsub -G groupA sleep 100
```

If `User1` is the highest priority user, this job will not be considered.

- `User1` can only associate the job with a group of which they are a member.
- `User1` cannot associate the job with his individual user account because **bsub -G** only accepts group names.

Example with hierarchical fair share

In the share tree, `User1` shares resources with `GroupA` at the top level. `GroupA` has 2 subgroups, `B` and `C`. `GroupC` has 1 subgroup, `GroupD`. `User1` also belongs to `GroupB` and `GroupC`.

`User1` submits a job:

```
bsub sleep 100
```

By default, the job could be considered for dispatch if either `User1`, `GroupB`, or `GroupC` has highest dynamic share priority.

`User1` submits a job and associates the job with `GroupB`:

```
bsub -G groupB sleep 100
```

If `User1` or `GroupC` is the highest priority user, this job will not be considered.

- `User1` cannot associate the job with `GroupC`, because `GroupC` includes a subgroup.
- `User1` cannot associate the job with his individual user account because **bsub -G** only accepts group names.

Re-sizable jobs and fair share

Re-sizable jobs submitting into fair share queues or host partitions are subject to fair share scheduling policies. The dynamic priority of the user who submitted the job is the most important criterion. LSF treats pending resize allocation requests as a regular job and enforces the fair share user priority policy to schedule them.

The dynamic priority of users depends on the following:

- Their share assignment
- The slots their jobs are currently consuming
- The resources their jobs consumed in the past
- The adjustment made by the fair share plug-in (`libfairshareadjust.*`)

Re-sizable job allocation changes affect the user priority calculation if the `RUN_JOB_FACTOR` or `FAIRSHARE_ADJUSTMENT_FACTOR` is greater than zero. Resize add requests increase number of slots in use and decrease user priority. Resize release requests decrease number of slots in use, and increase user priority. The faster a re-sizable job grows, the lower the user priority is, the less likely a pending allocation request can get more slots.

Note: The effect of re-sizable job allocation changes when the `Fairshare_adjustment_factor` is greater than zero, and depends on the user-defined fair share adjustment plug-in (`libfairshareadjust.*`). After job allocation changes, running the `bqueues` and `bhpart` display updated user priority.

Guaranteed resource pools

Guaranteed resource pools provide a minimum resource guarantee to a group of users or other consumers.

Resource pools can optionally lend guaranteed resources that are not in use. During job scheduling, the order of job scheduling does not change, but some jobs have access to additional guaranteed resources. After the guaranteed resources are used, jobs run outside the guarantee following whatever other scheduling features are configured.

Note: Hosts that are not ready for dispatched jobs are not assigned to the guaranteed resource pool. This include hosts that are in `unavail` or `unreach` status, or hosts that are closed by the administrator.

- [About guaranteed resources](#)
Use guaranteed resources when you want LSF to reserve some amount of resources for a group of jobs.
- [Configuration overview of guaranteed resource pools](#)
- [Submitting jobs to use guarantees](#)
For a job to access guaranteed resources, it must belong to a service class. A job in a service class can use resources that are guaranteed to that service class.
- [Package guarantees](#)
A package comprises some number of slots and some amount of memory all on a single host.
- [Adding consumers to a guaranteed resource pool](#)

About guaranteed resources

Use guaranteed resources when you want LSF to reserve some amount of resources for a group of jobs.

LSF allows for guarantees of the following resources:

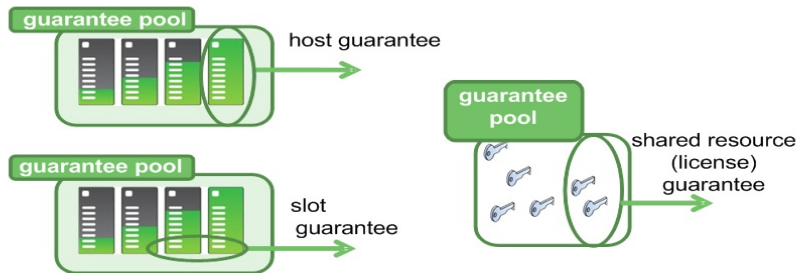
- Whole hosts
- Slots
- Packages, composed of a number of slots and some amount of memory together on a host
- Licenses managed by License Scheduler

LSF uses service classes in order to group jobs for the purpose of providing guarantees. In the context of guarantees, a service class can be thought of as simply a job container. A job can be submitted to a service class with the `bsub -sla` option. You can configure access controls on a service class to control which jobs are allowed to use the service class. As well, you can configure LSF to automatically associate jobs with a service class that meet the access control criteria. A job can belong to at most one service class.

A guarantee policy requires you to specify the following:

- Resource pool: The pool is specified by a type of resource (whole hosts, slots, packages, or licenses). Also, for host-based resources (hosts, slots, and packages) you may specify the set hosts from which the resources are to be reserved.
- Guarantees: These are the amounts of the resource in the pool that should be reserved for each service class.

Note that a service class can potentially have guarantees in multiple pools.



Prior to scheduling jobs, LSF determines the number of free resources in each pool, and the number of resources that must be reserved in order to honor guarantees. Then, LSF considers jobs for dispatch according to whatever job prioritization policies are configured (queue priority, fair share, job priority). LSF will limit job access to the resources in order to try to honor the guarantees made for the service classes.

Optionally, a guarantee policy can be configured such that resources not needed immediately for guarantees can be borrowed by other jobs. This allows LSF to maximize utilization of the pool resources, while ensure that specific groups of jobs can get minimum amounts of resources when needed.

Note that a guarantee policy will not affect job prioritization directly. Rather, it works by limiting the number of resources in a pool that a given job can use, based on the job's service class. The advantage of this approach is that guarantee policies can be combined with any job prioritization policy in LSF.

Normally, a pool will have a greater number of resources than the number of resources guaranteed from the pool to service classes. Resources in a pool in excess of what is required for guarantees can potentially be used by any job, regardless of service class, or even by jobs that are not associated with a service class.

Configuration overview of guaranteed resource pools

Basic service class configuration

Service classes are configured in `lsb.serviceclasses`. At a minimum, for each service class to be used in a guarantee policy, you must specify the following parameters:

- NAME = `service_class_name`: This is the name of the service class.
- GOALS = `[GUARANTEE]`: To distinguish from other types of service class, you must give the guarantee goal.

Optionally, your service class can have a description. Use the `DESCRIPTION` parameter.

The following is an example of a basic service class configuration:

```
Begin ServiceClass
NAME = myServiceClass
GOALS = [GUARANTEE]
DESCRIPTION = Example service class.
End ServiceClass
```

Once a service class is configured, you can submit jobs to this service class with the **bsub -sla** submission option:

```
bsub -sla myServiceClass ./a.out
```

The service class only defines the container for jobs. In order to complete the guarantee policy, you must also configure the pool. This is done in the `GuaranteedResourcePool` section of `lsb.resources`.

Basic guarantee policy configuration

At minimum, for `GuaranteedResourcePool` sections you need to provide values for the following parameters:

- **NAME** = `pool_name`: The name of the guarantee policy/pool.
- **TYPE** = `slots` | `hosts` | `package[slots=num_slots:mem=mem_amount]` | `resource[rsrc_name]`
 - The resources that compose the pool.
 - Package means that each unit guaranteed is composed of a number of slots, and some amount of memory together on the same host.
 - resource must be a License Scheduler managed resource.
- **DISTRIBUTION** = `[service_class, amount[%]]` ...
 - Describes the number of resources in the pool deserved by each service class.
 - A percentage guarantee means percentage of the guaranteed resources in the pool.

Optionally, you can also include a description of a `GuaranteedResourcePool` using the `DESCRIPTION` parameter.

The following is an example of a guaranteed resource pool configuration:

```
Begin GuaranteedResourcePool
NAME = myPool
Type = slots
DISTRIBUTION = [myServiceClass, 10] [yourServiceClass, 15]
DESCRIPTION = Example guarantee policy.
End GuaranteedResourcePool
```

Controlling access to a service class

You can control which jobs are allowed into a service class by setting the following parameter in the `ServiceClass` section:

```
ACCESS_CONTROL = [QUEUES[ queue ...]] [USERS[ user_name] [user_group] ...]] [FAIRSHARE_GROUPS[user_group ...]]
[APPS[app_name ...]] [PROJECTS[proj_name ...]] [LIC_PROJECTS[license_proj ...]]
```

Where:

- **QUEUES**: restricts access based on queue
- **USERS**: restricts access based on user
- **FAIRSHARE_GROUPS**: restricts access based on `bsub -G` option
- **APPS**: restricts access based on `bsub -app` option
- **PROJECTS**: restricts access based on `bsub -P` option
- **LIC_PROJECTS**: restricts access based on `bsub -Lp` option

When `ACCESS_CONTROL` is not configured for a service class, any job can be submitted to the service class with the `-sla` option. If `ACCESS_CONTROL` is configured and a job is submitted to the service class, but the job does not meet the access control criteria of the service class, then the submission is rejected.

The following example shows a service class that only accepts jobs from the priority queue (from user `joe`):

```
Begin ServiceClass
NAME = myServiceClass
GOALS = [GUARANTEE]
ACCESS_CONTROL = QUEUES[priority] USERS[joe]
DESCRIPTION = Example service class.
End ServiceClass
```

Have LSF automatically put jobs in service classes

A job can be associated with a service class by using the **bsub -sla** option to name the service class. You can configure a service class so that LSF will automatically try to put the job in the service class if the job meets the access control criteria. Use

the following parameter in the ServiceClass definition:

`AUTO_ATTACH=Y`

When a job is submitted without a service class explicitly specified (that is, the **bsub -sla** option is not specified) then LSF will consider the service classes with `AUTO_ATTACH=Y` and put the job in the first such service class for which the job meets the access control criteria. Each job can be associated with at most one service class.

The following is an example of a service class that automatically accepts jobs from user `joe` in queue priority:

```
Begin ServiceClass
NAME = myServiceClass
GOALS = [GUARANTEE]
ACCESS_CONTROL = QUEUES[priority] USERS[joe]
AUTO_ATTACH = Y
DESCRIPTION = Example service class.
End ServiceClass
```

Restricting the set of hosts in a guaranteed resource pool

Each host in the cluster can potentially belong to at most one pool of type, slots, hosts or package. To restrict the set of hosts that can belong to a pool, use the following parameters:

- `RES_SELECT = select_string`
- `HOSTS = host | hostgroup ...`

The syntax for `RES_SELECT` is the same as in **bsub -R "select[...]"**.

When LSF starts up, it goes through the hosts and assigns each host to a pool that will accept the host, based on the pool's `RES_SELECT` and `HOSTS` parameters. If multiple pools will accept the host, then the host will be assigned to the first pool according to the configuration order of the pools.

The following is an example of a guaranteed resource policy on hosts of type `x86_64` from host group `myHostGroup`:

```
Begin GuaranteedResourcePool
NAME = myPool
TYPE = slots
RES_SELECT = type==X86_64
HOSTS = myHostGroup
DISTRIBUTION = [myServiceClass, 10] [yourServiceClass, 15]
End GuaranteedResourcePool
```

Loaning resources from a pool

At the beginning of each scheduling session, LSF partitions the hosts in each guarantee package pool into owner hosts and shared hosts. Owner hosts are only for owners who have not exceeded the guarantee, while shared hosts are available for general use. LSF ensures that there are enough owner hosts to honor the configured guarantees so that the total number of packages on all owned hosts is at least the number of packages that are configured for owners.

When LSF schedules, it tries to reserve sufficient resources from the pool in order to honor guarantees. A job is only allowed on an owner host if the job belongs to a service class and the requested resources are not greater than its guaranteed number of packages within a pool. Jobs can use shared hosts without restriction. By default, if these reserved resources cannot be used immediately to satisfy guarantees, they are left idle.

Optionally, you can configure loaning to allow other jobs to use these resources when they are not needed immediately for guarantees. If you enable loaning, jobs that are permitted to loan from the pool can use the owner hosts, but these jobs have lower priority access to these hosts than the owners. If you do not enable loaning, service class jobs that request more than the number of guaranteed resources are not allowed to dispatch jobs on owner hosts.

To enable loaning, use the following parameter in the pool:

```
LOAN_POLICIES = QUEUES[all | [!]queue_name ...] [IDLE_BUFFER[amount[%]]] [DURATION[minutes]] [CLOSE_ON_DEMAND]
```

Where:

- **QUEUES**[all | queue_name ...]
 - This is the only required keyword.
 - Specifies which queues are allowed to loan from the pool.
 - As more queues are permitted to loan, this can potentially degrade scheduling performance, so be careful about adding queues if scheduling performance is a concern.
 - Specify an exclamation point (!) before the queue name for that queue to ignore any **IDLE_BUFFER** and **DURATION** policies when deciding whether a job in the queue can borrow unused guaranteed resources.
 - **IDLE_BUFFER**[amount[%]]
 - Without this keyword, LSF will potentially loan out all the resources in the pool to non-owners (that is, those jobs without guarantees) when you enable loan policies, and there may never be a free package. Guaranteed jobs may starve (if resource reservation is not used). So **IDLE_BUFFER** can be used as an alternative to resource reservation in such cases.
 - When **IDLE_BUFFER** is set, then as long as there are unused guarantees, LSF will try to keep idle the amount of resources specified in **IDLE_BUFFER**. These idle resources can only be used to honor guarantees. Whenever the number of free resources in the pool drops below the **IDLE_BUFFER** amount, LSF stops loaning resources from the pool.
 - With **IDLE_BUFFER**, LSF maintains an idle buffer. The number kept idle is: $\text{MIN}(\text{IDLE_BUFFER}, \text{amount needed for unused guarantees})$.
 - For example, suppose that a service class owns 100% of a pool and **IDLE_BUFFER** is 10. Initially, LSF will loan out all but 10 of the resources. If the service class then occupies those 10 resources, LSF will stop loaning to non-guaranteed jobs until more than 10 resources free up (as jobs finish).
 - This policy is ignored for any queues that have a preceding exclamation point (!) in its queue name in the **QUEUES** keyword specification when deciding whether a job in the queue can borrow unused guaranteed resources.
- Note: The **RETAIN** keyword is deprecated in LSF, Version 10.1.0 Fix Pack 10. Use **IDLE_BUFFER** instead of **RETAIN**.
- **DURATION**[minutes]
 - Specifies that only jobs with runtime (-W) or expected runtime (-We) less than the given number of minutes are permitted loans from the pool.
 - Means that if later there is demand from a service class with a guarantee in the pool, the service class will not have to wait longer than the **DURATION** before it is able to have its guarantee met.
 - This policy is ignored for any queues that have a preceding exclamation point (!) in its queue name in the **QUEUES** keyword specification when deciding whether a job in the queue can borrow unused guaranteed resources.
 - **CLOSE_ON_DEMAND**
 - Tells LSF that loaning should be disabled whenever there are pending jobs belonging to service classes with guarantees in the pool.
 - This is a very conservative policy. It should generally only be used when the service classes with guarantees in the pool have workload submitted to them only infrequently.

The following is an example of a guarantee package policy that loans resources to jobs in queue short, but keeps sufficient resources for 10 packages unavailable for loaning so it can honor guarantees immediately when there is demand from the service classes:

```
Begin GuaranteedResourcePool
NAME = myPool
TYPE = package[slots=1:mem=1024]
DISTRIBUTION = [myServiceClass, 10] [yourServiceClass, 15]
LOAN_POLICIES = QUEUES[short] IDLE_BUFFER[10]
End GuaranteedResourcePool
```

Configuring a high priority queue to ignore guarantees

In some cases, you would like guarantees to apply to batch workload. However, for some high priority interactive or administrative workloads, you would like to get jobs running as soon as possible, without regard to guarantees.

You can configure a queue to ignore guarantee policies by setting the following parameter in the queue definition in `lsb.queues`:

```
SLA_GUARANTEES_IGNORE=Y
```

This parameter essentially allows the queue to violate any configured guarantee policies. The queue can take any resources that should be reserved for guarantees. As such, queues with this parameter set should have infrequent or limited workload.

The following example shows how to configure a high priority interactive queue to ignore guarantee policies:

```
Begin Queue
QUEUE_NAME = interactive
PRIORITY = 100
SLA_GUARANTEES_IGNORE = Y
DESCRIPTION = A high priority interactive queue that ignores all guarantee policies.
End Queue
```

Best practices for configuring guaranteed resource pools

- In each guarantee pool, hosts should be relatively homogeneous in terms of the resources that will be available to the jobs.
- Each job with a guarantee should ideally be able to fit within a single unit of the guaranteed resources.
 - In a slot type pool, each job with a guarantee should require only a single slot to run. Otherwise, multiple slots may be reserved on different hosts and the job may not run.
 - In a package type pool, each job should require only a single package.
- For each guarantee policy, you must give the list of queues able to loan from the pool. For each queue able to loan, LSF must try scheduling from the queue twice during each scheduling session. This can potentially degrade scheduling performance. If scheduling performance is a concern, be sure to limit the number of queues able to loan.
- When configuring the RES_SELECT parameter for a pool, use only static resources (such as maxmem) instead of dynamically changing resources (such as mem).

Submitting jobs to use guarantees

For a job to access guaranteed resources, it must belong to a service class. A job in a service class can use resources that are guaranteed to that service class.

Interactions with guarantee policies

About this task

There are two ways a job can be associated with a service class:

- You can use the **bsub -sla** option to explicitly associate a job with a service class.
- You can submit a job without the **-sla** option, and LSF will put the job in the first service class (by configuration order) with AUTO_ATTACH=Y, such that the job meets the service class access control criteria.

For example, you can submit a job to service class **myServiceClass**, as follows:

```
bsub -sla myServiceClass ./a.out
```

A guarantee pool of host-based resources (slots, hosts, package) includes only hosts in the following states:

- ok
- closed_Busy
- closed_Excl
- closed_cu_Excl
- closed_Full

Hosts in other states are temporarily excluded from the pool, and any SLA jobs running on hosts in other states are not counted towards the guarantee.

Advance reservation

Hosts within an advance reservation are excluded from guaranteed resource pools.

Compute units

Configuring guaranteed resource pools and compute units with hosts in common is not recommended. If such configuration is required, do not submit jobs with compute unit requirements using the maxcus, balance, or excl keywords.

Queue-based fair share

During loan scheduling, shares between queues are not preserved.

Exclusive jobs

Using exclusive jobs with slot-type guaranteed resource pools is not recommended. Instead, use host-type pools.

Multicluster

Leased hosts can be used in a guaranteed resource pool by including a host group with remote hosts in the HOSTS parameter.

Preemption

Guarantee SLA jobs can only be preempted by queues with `SLA_GUARANTEES_IGNORE=Y`. If a queue does not have this parameter set, jobs in this queue cannot trigger preemption of an SLA job. If an SLA job is suspended (for example, by a **bstop**), jobs in queues without the parameter being set cannot make use of the slots released by the suspended job.

Jobs scheduled using loaned resources cannot trigger preemption.

Guarantee SLA jobs can preempt other jobs, and can use preemption to meet guarantees. Normally, jobs attached to guarantee-type SLAs cannot be preempted even if they are running outside any guarantees or outside any pools in which they have guarantees. The exception to this is when you set the parameter `SLA_GUARANTEES_IGNORE=y` in a preemptive queue to allow the queue to preempt jobs attached to guarantee SLAs.

Chunk jobs

Jobs running on loaned resources cannot be chunked.

Forced jobs (**brun**)

Jobs that are forced to run using **brun** can use resources regardless of guarantees.

Resource duration

Duration for the memory **rusage** string is ignored for jobs that are running in package type guarantee pools.

Package guarantees

A package comprises some number of slots and some amount of memory all on a single host.

Administrators can configure an service class of a number of packages for jobs of a particular class. A package has all the slot and memory resources for a single job of that class to run. Each job running in a guarantee pool must occupy the whole multiple of packages. You should define a package size based on the resource requirement of the jobs for which you made the guarantees.

Configuring guarantee package policies

Guarantee policies (pools) are configured in `lsb.resources`. For package guarantees, these policies specify:

- A set (pool) of hosts
- The resources in a package
- How many packages to reserve for each set of service classes
- Policies for loaning out reserved resources that are not immediately needed

Configuration is done the same as for a slot or host guarantee policy, with a **GuaranteedResourcePool** section in `lsb.resources`. The main difference being that the `TYPE` parameter is used to express the package resources. The following example is a guarantee package pool defined in `lsb.resources`:

```
Begin GuaranteedResourcePool
NAME = example_pool
TYPE = package[slots=1:mem=1000]
HOSTS = hgroup1
RES_SELECT = mem > 16000
DISTRIBUTION = ([sc1, 25%] [sc2, 25%] [sc3, 30%])
End GuaranteedResourcePool
```

A package does not necessarily require both slots and memory. Setting `TYPE=package [slots=1]` gives essentially the same result as a slot pool. It may be useful to have only slots in a package (and not mem) in order to provide guarantees for parallel jobs that require multiple CPUs on a single host, where memory is not an important resource. It is likely not useful to configure guarantees of only memory without slots, although the feature supports this.

Each host can belong to at most one slot/host/package guarantee pool. At **mbatchd** startup time, it will go through hosts one by one. For each host, **mbatchd** will go through the list of guarantee pools in configuration order, and assign the host to the first pool for which the job meets the RES_SELECT and HOSTS criteria.

Total packages of a pool

The total packages of a pool is intended to represent the number of packages that can be supplied by the pool if there are no jobs running in the pool. This total is used for:

- Display purposes – **bresources** displays the total for each pool, as well as showing the pool status as overcommitted when the number guaranteed in the pool exceeds the total.
- Determining the actual number of packages to reserve when guarantees are given as percentages instead of absolute numbers.

LSF calculates the total packages of a pool by summing over all hosts in the pool, the total package each host. Hosts that are currently unavailable are not considered to be part of a pool. On each host in a pool, the total contributed by the host is the number of packages that fit into the MXJ and total memory of the host. For the purposes of computing the total packages of the host, **mbatchd** estimates the total memory for LSF jobs as the minimum of:

- The total slots of the host (MXJ), and
- The maximum memory of the host; that is, maxmem as reported by the **lshosts** command.

The total packages on a host is the number of packages that can fit into the total slots and maxmem of the host. This way, the memory occupied by processes on the host that do not belong to LSF jobs does not count toward the total packages for the host. Even if you kill all the memory occupied by jobs on the host, LSF jobs might not use memory all the way to maxmem.

Memory on a host can be used by processes outside of LSF jobs. Even when no jobs are running on a host, the number of free packages on the host is less than the total packages of the host. The free packages are computed from the available slots and available memory.

Currently available packages in a pool

So that LSF knows how many packages to reserve during scheduling, LSF must track the number of available packages in each package pool. The number of packages available on a host in the pool is equal to the number of packages that fit into the free resources on the host. The available packages of a pool is simply this amount summed over all hosts in the pool.

For example, suppose there are 5 slots and 5 GB of memory free on the host. Each package contains 2 slots and 2 GB of memory. Therefore, 2 packages are currently available on the host.

Hosts in other states are temporarily excluded from the pool, and any SLA jobs running on hosts in other states are not counted towards the guarantee.

Adding consumers to a guaranteed resource pool

About this task

Change the DISTRIBUTION of a guaranteed resource pool in `lsb.resources` using live reconfiguration.

Procedure

Run **bconf addmember gpool=pool_name "DISTRIBUTION=([SLA, share])"**

For example, for the existing `lsb.resources` configuration:

```
...
Begin GuaranteedResourcePool
NAME=my_pool
DISTRIBUTION=([SLA1, 10] [SLA2, 30])
...
End GuaranteedResourcePool
...
```

Add another SLA and share:

```
bconf addmember gpool=my_pool "DISTRIBUTION=([SLA3, 10])"
bconf: Request for gpool <my_pool> accepted
```

Once accepted by **bconf**, the new share definition appears in **bqueue -l** output:

```
bresources -gl my_pool
GUARANTEED RESOURCE POOL: my_pool
TYPE: slots
DISTRIBUTION: [SLA1,10] [SLA2,30] [SLA3,10]
...
```

Note: An SLA is neither a user group nor a host group. Do not use **bconf** to update an SLA. For more about guaranteed resource pools see [About guaranteed resources](#)

Reserving memory and license resources

Use LSF to reserve memory and license resources for high-throughput workload.

- [Memory reservation for pending jobs](#)
By default, the rusage string reserves resources for running jobs. Because resources are not reserved for pending jobs, some memory-intensive jobs could be pending indefinitely because smaller jobs take the resources immediately before the larger jobs can start running. The more memory a job requires, the worse the problem is.
- [Reserving license resources](#)
Use LSF to reserve license resources for high-throughput workload.

Memory reservation for pending jobs

By default, the rusage string reserves resources for running jobs. Because resources are not reserved for pending jobs, some memory-intensive jobs could be pending indefinitely because smaller jobs take the resources immediately before the larger jobs can start running. The more memory a job requires, the worse the problem is.

Memory reservation for pending jobs solves this problem by reserving memory as it becomes available until the total required memory specified on the rusage string is accumulated and the job can start. Use memory reservation for pending jobs if memory-intensive jobs often compete for memory with smaller jobs in your cluster.

- [Reserving host memory for pending jobs](#)
- [Enabling memory reservation for sequential jobs](#)
- [Configuring lsb.queues](#)
- [Using memory reservation for pending jobs](#)
- [How memory reservation for pending jobs works](#)

Reserving host memory for pending jobs

Procedure

Use the RESOURCE_RESERVE parameter in lsb.queues to reserve host memory for pending jobs.

The amount of memory reserved is based on the currently available memory when the job is pending. Reserved memory expires at the end of the time period represented by the number of dispatch cycles specified by the value of MAX_RESERVE_TIME set on the RESOURCE_RESERVE parameter.

Enabling memory reservation for sequential jobs

Procedure

Add the LSF scheduler plug-in module name for resource reservation (schmod_reserve) to the lsb.modules file:

```
Begin PluginModule
SCH_PLUGIN          RB_PLUGIN          SCH_DISABLE_PHASES
schmod_default      ()                  ()
schmod_reserve      ()                  ()
schmod_preemption   ()                  ()
End PluginModule
```

Configuring lsb.queues

Procedure

Set the RESOURCE_RESERVE parameter in a queue defined in lsb.queues.

If both RESOURCE_RESERVE and SLOT_RESERVE are defined in the same queue, job slot reservation and memory reservation are both enabled and an error is displayed when the cluster is reconfigured. SLOT_RESERVE is ignored.

Example queues

The following queue enables memory reservation for pending jobs:

```
Begin Queue
QUEUE_NAME = reservation
DESCRIPTION = For resource reservation
PRIORITY=40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
End Queue
```

Using memory reservation for pending jobs

Procedure

Use the rusage string in the **-R** option to **bsub** or the RES_REQ parameter in lsb.queues to specify the amount of memory required for the job. Submit the job to a queue with RESOURCE_RESERVE configured.

Note:

Compound resource requirements do not support use of the || operator within the component rusage simple resource requirements, multiple -R options, or the cu section.

How memory reservation for pending jobs works

Amount of memory reserved

The amount of memory reserved is based on the currently available memory when the job is pending. For example, if LIM reports that a host has 300 MB of memory available, the job submitted by the following command:

```
bsub -R "rusage[mem=400]" -q reservation my_job
```

will be pending and reserve the 300 MB of available memory. As other jobs finish, the memory that becomes available is added to the reserved memory until 400 MB accumulates, and the job starts.

No memory is reserved if no job slots are available for the job because the job could not run anyway, so reserving memory would waste the resource.

Only memory is accumulated while the job is pending; other resources specified on the rusage string are only reserved when the job is running. Duration and decay have no effect on memory reservation while the job is pending.

How long memory is reserved (MAX_RESERVE_TIME)

Reserved memory expires at the end of the time period represented by the number of dispatch cycles specified by the value of MAX_RESERVE_TIME set on the RESOURCE_RESERVE parameter. If a job has not accumulated enough memory to start by the time MAX_RESERVE_TIME expires, it releases all its reserved memory so that other pending jobs can run. After the reservation time expires, the job cannot reserve slots or memory for one scheduling session, so other jobs have a chance to be dispatched. After one scheduling session, the job can reserve available resources again for another period that is specified by MAX_RESERVE_TIME.

Examples

lsb.queues

The following queues are defined in lsb.queues:

```
Begin Queue  
QUEUE_NAME = reservation  
DESCRIPTION = For resource reservation  
PRIORITY=40  
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]  
End Queue
```

Assumptions

Assume one host in the cluster with 10 CPUs and 1 GB of free memory currently available.

Sequential jobs

Each of the following sequential jobs requires 400 MB of memory and runs for 300 minutes.

Job 1:

```
bsub -W 300 -R "rusage[mem=400]" -q reservation myjob1
```

The job starts running, using 400M of memory and one job slot.

Job 2:

Submitting a second job with same requirements yields the same result.

Job 3:

Submitting a third job with same requirements reserves one job slot, and reserves all free memory, if the amount of free memory is between 20 MB and 200 MB (some free memory may be used by the operating system or other software.)

Reserving license resources

Use LSF to reserve license resources for high-throughput workload.

Administering IBM Spectrum LSF parallel workload

Learn how to submit, monitor, and control parallel workload in your LSF cluster. Configure scheduling policies that reserve resources to enable efficient execution of large parallel jobs.

- [Running parallel jobs](#)
LSF provides a generic interface to parallel programming packages so that any parallel package can be supported by writing shell scripts or wrapper programs.
- [Advance reservation](#)
Advance reservations ensure access to specific hosts or slots during specified times. During the time that an advance reservation is active only users or groups associated with the reservation have access to start new jobs on the reserved hosts or slots.
- [Fair share scheduling](#)
fair share scheduling divides the processing power of the LSF cluster among users and queues to provide fair access to resources, so that no user or queue can monopolize the resources of the cluster and no queue will be starved.
- [Job checkpoint and restart](#)
Optimize resource usage with job checkpoint and restart to stop jobs and then restart them from the point at which they stopped.
- [Job migration for checkpoint-able and re-runnable jobs](#)
Use job migration to move checkpoint-able and re-runnable jobs from one host to another. Job migration makes use of job checkpoint and restart so that a migrated checkpoint-able job restarts on the new host from the point at which the job stopped on the original host.
- [Re-sizable jobs](#)
Re-sizable jobs can use the number of tasks that are available at any time and can grow or shrink during the job run time by requesting extra tasks if required or release tasks that are no longer needed.

Running parallel jobs

LSF provides a generic interface to parallel programming packages so that any parallel package can be supported by writing shell scripts or wrapper programs.

- [How LSF runs parallel jobs](#)
When LSF runs a job, the LSB_HOSTS variable is set to the names of the hosts running the batch job. For a parallel batch job, LSB_HOSTS contains the complete list of hosts that LSF has allocated to that job.
- [Preparing your environment to submit parallel jobs to LSF](#)
- [Submitting a parallel job](#)
- [Starting parallel tasks with LSF utilities](#)
For simple parallel jobs you can use LSF utilities to start parts of the job on other hosts. Because LSF utilities handle signals transparently, LSF can suspend and resume all components of your job without additional programming.
- [Job slot limits for parallel jobs](#)
A job slot is the basic unit of processor allocation in LSF. A sequential job uses one job slot. A parallel job that has N components (tasks) uses N job slots, which can span multiple hosts.
- [Specify a minimum and maximum number of tasks](#)
By default, when scheduling a parallel job, the number of slots allocated on each host will not exceed the number of CPUs on that host even though host MXJ is set greater than number of CPUs. When submitting a parallel job, you can also specify a minimum number and a maximum number of tasks.
- [Restricting job size requested by parallel jobs](#)
Specifying a list of allowed job sizes (number of tasks) in queues or application profiles enables LSF to check the requested job sizes when submitting, modifying, or switching jobs.
- [About specifying a first execution host](#)
In general, the first execution host satisfies certain resource requirements that might not be present on other available hosts.
- [Compute units](#)
Compute units are similar to host groups, with the added feature of granularity allowing the construction of cluster-wide structures that mimic network architecture. Job scheduling using compute unit resource requirements optimizes job placement based on the underlying system architecture, minimizing communications bottlenecks. Compute units are

especially useful when running communication-intensive parallel jobs spanning several hosts. Compute units encode cluster network topology for jobs with a lot of communication between processes. For example, compute units can help minimize network latency and take advantage of fast interconnects by placing all job tasks in the same rack instead of making several network hops.

- [Control processor allocation across hosts](#)

Sometimes you need to control how the selected processors for a parallel job are distributed across the hosts in the cluster.

- [Run parallel processes on homogeneous hosts](#)

Parallel jobs run on multiple hosts. If your cluster has heterogeneous hosts some processes from a parallel job may for example, run on Solaris. However, for performance reasons you may want all processes of a job to run on the same type of host instead of having some processes run on one type of host and others on another type of host.

- [Limit the number of processors allocated](#)

Use the TASKLIMIT parameter in lsb.queues or lsb.applications to limit the number of tasks that can be allocated to a parallel job.

- [Limit the number of allocated hosts](#)

Use the HOSTLIMIT_PER_JOB parameter in lsb.queues to limit the number of hosts that a job can use. For example, if a user submits a parallel job using `bsub -n 1,4096 -R "span[ptile=1]"`, this job requests 4096 hosts from the cluster. If you specify a limit of 20 hosts per job, a user submitting a job requesting 4096 hosts will only be allowed to use 20 hosts.

- [Reserve processors](#)

- [Reserve memory for pending parallel jobs](#)

- [Backfill scheduling](#)

By default, a reserved job slot cannot be used by another job. To make better use of resources and improve performance of LSF, you can configure backfill scheduling.

- [How deadline constraint scheduling works for parallel jobs](#)

Deadline constraint scheduling is enabled by default.

- [Optimized preemption of parallel jobs](#)

You can configure preemption for parallel jobs to reduce the number of jobs suspended in order to run a large parallel job.

- [Controlling CPU and memory affinity](#)

IBM® Spectrum LSF can schedule jobs that are affinity aware. This allows jobs to take advantage of different levels of processing units (NUMA nodes, sockets, cores, and threads). Affinity scheduling is supported only on Linux and Power 7 and Power 8 hosts. Affinity scheduling is supported in LSF Standard Edition and LSF Advanced Edition. Affinity scheduling is not supported on LSF Express Edition.

- [Processor binding for LSF job processes](#)

Processor binding for LSF job processes takes advantage of the power of multiple processors and multiple cores to provide hard processor binding functionality for sequential LSF jobs and parallel jobs that run on a single host.

- [Running parallel jobs with blaunch](#)

Learn how to configure and use the **blaunch** command for launching parallel and distributed applications within LSF. Task geometry allows for flexibility in how tasks are grouped for execution on system nodes. A typical LSF parallel job launches its tasks across multiple hosts. By default you can enforce limits on the total resources used by all the tasks in the job.

- [Running MPI workload through IBM Parallel Environment Runtime Edition](#)

IBM Spectrum LSF integrates with the IBM Parallel Environment Runtime Edition (IBM PE Runtime Edition) program product - Version 1.3 or later to run PE jobs through the IBM Parallel Operating Environment (POE). The integration enables network-aware scheduling, allowing an LSF job to specify network resource requirements, collect network information, and schedule the job according to the requested network resources.

How LSF runs parallel jobs

When LSF runs a job, the LSB_HOSTS variable is set to the names of the hosts running the batch job. For a parallel batch job, LSB_HOSTS contains the complete list of hosts that LSF has allocated to that job.

LSF starts one controlling process for the parallel batch job on the first host in the host list. It is up to your parallel application to read the LSB_HOSTS environment variable to get the list of hosts, and start the parallel job components on all the other allocated hosts.

For running large parallel jobs, use `LSB_MCPU_HOSTS`. The format for this parameter is `LSB_MCPU_HOSTS="host_nameA num_processors1 host_nameB num_processors2..."`

LSF provides a generic interface to parallel programming packages so that any parallel package can be supported by writing shell scripts or wrapper programs.

Preparing your environment to submit parallel jobs to LSF

Getting the host list

Some applications can take this list of hosts directly as a command line parameter. For other applications, you may need to process the host list.

Example

The following example shows a **/bin/sh** script that processes all the hosts in the host list, including identifying the host where the job script is executing.

```
#!/bin/sh
# Process the list of host names in LSB_HOSTS

for host in $LSB_HOSTS ; do
  handle_host $host
done
```

Parallel job scripts

Each parallel programming package has different requirements for specifying and communicating with all the hosts used by a parallel job. LSF is not tailored to work with a specific parallel programming package. Instead, LSF provides a generic interface so that any parallel package can be supported by writing shell scripts or wrapper programs.

You can modify these scripts to support more parallel packages.

- [Using a job starter](#)

Using a job starter

About this task

You can configure the script into your queue as a job starter, and then all users can submit parallel jobs without having to type the script name.

Procedure

To see if your queue already has a job starter defined, run **bqueues -l**.

Submitting a parallel job

About this task

LSF can allocate more than one slot to run a job and automatically keeps track of the job status, while a parallel job is running.

When submitting a parallel job that requires multiple slots, you can specify the exact number of slots to use.

Procedure

1. To submit a parallel job, use **bsub -n** and specify the number of slots the job requires.
2. To submit jobs based on the number of available job slots instead of the number of CPUs, use `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`.
For example:

```
bsub -n 4 myjob
```

The job **myjob** submits as a parallel job. The job is started when four job slots are available.

Note: When `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the resource requirement string keyword `ncpus` refers to the number of slots instead of the number of CPUs however **lshosts** output will continue to show `ncpus` as defined by `EGO_DEFINE_NCPUS` in `lsf.conf`.

Starting parallel tasks with LSF utilities

For simple parallel jobs you can use LSF utilities to start parts of the job on other hosts. Because LSF utilities handle signals transparently, LSF can suspend and resume all components of your job without additional programming.

Running parallel tasks with **lsgrun**

The simplest parallel job runs an identical copy of the executable on every host. The **lsgrun** command takes a list of host names and runs the specified task on each host. The **lsgrun -p** command specifies that the task should be run in parallel on each host.

Example

This example submits a job that uses **lsgrun** to run **myjob** on all the selected hosts in parallel:

```
bsub -n 10 'lsgrun -p -m "$LSB_HOSTS" myjob'  
Job <3856> is submitted to default queue <normal>.
```

For more complicated jobs, you can write a shell script that runs **lsrun** in the background to start each component.

Running parallel tasks with the **blaunch** distributed application framework

Most MPI implementations and many distributed applications use **rsh** and **ssh** as their task launching mechanism. The **blaunch** command provides a drop-in replacement for **rsh** and **ssh** as a transparent method for launching parallel and distributed applications within LSF.

Similar to the **lsrun** command, **blaunch** transparently connects directly to the RES and **sbatchd** on the remote host, and subsequently creates and tracks the remote tasks, and provides the connection back to LSF. There is no need to insert `pam` or `taskstarter` into the **rsh** or **ssh** calling sequence, or configure any wrapper scripts.

Important: You cannot run **blaunch** directly from the command line.

blaunch only works within an LSF job; it can only be used to launch tasks on remote hosts that are part of a job allocation. It cannot be used as a standalone command. On success **blaunch** exits with 0.

Windows: **blaunch** is supported on Windows 2000 or later with the following exceptions:

- Only the following signals are supported: SIGKILL, SIGSTOP, SIGCONT.
- The `-n` option is not supported.
- `CMD . EXE /C <user command line>` is used as intermediate command shell when: `-no-shell` is not specified

- `CMD.EXE /C` is not used when `-no-shell` is specified.
- Windows User Account Control must be configured correctly to run jobs.

Submitting jobs with **blaunch**

Use **bsub** to call **blaunch**, or to invoke a job script that calls **blaunch**. The **blaunch** command assumes that **bsub -n** implies one remote task per job slot.

The **blaunch** syntax is:

```
blaunch [-n] [-u host_file | -z host_name ... | host_name] [-use-login-shell | -no-shell] command [argument ...]
```

```
blaunch [-h | -V]
```

The following are some examples of **blaunch** usage:

- Submit a parallel job:

```
bsub -n 4 blaunch myjob
```

- Submit a job to an application profile

```
bsub -n 4 -app pjob blaunch myjob
```

Job slot limits for parallel jobs

A job slot is the basic unit of processor allocation in LSF. A sequential job uses one job slot. A parallel job that has N components (tasks) uses N job slots, which can span multiple hosts.

By default, running and suspended jobs count against the job slot limits for queues, users, hosts, and processors that they are associated with.

With processor reservation, job slots that are reserved by pending jobs also count against all job slot limits.

When backfilling occurs, the job slots used by backfill jobs count against the job slot limits for the queues and users, but not hosts or processors. This means when a pending job and a running job occupy the same physical job slot on a host, both jobs count towards the queue limit, but only the pending job counts towards host limit.

Specify a minimum and maximum number of tasks

By default, when scheduling a parallel job, the number of slots allocated on each host will not exceed the number of CPUs on that host even though `host MXJ` is set greater than number of CPUs. When submitting a parallel job, you can also specify a minimum number and a maximum number of tasks.

If you specify a maximum and minimum number of tasks, the job can start if the minimum number of processors are available, but it always tries to use up to the maximum number of processors, depending on how many processors are available at the time. Once the job starts running, no more processors are allocated to it even though more may be available later on.

Jobs that request fewer tasks than the minimum `TASKLIMIT` defined for the queue or application profile to which the job is submitted, or more tasks than the maximum `TASKLIMIT` are rejected. If the job requests minimum and maximum tasks, the maximum requested cannot be less than the minimum `TASKLIMIT`, and the minimum requested cannot be more than the maximum `TASKLIMIT`.

If `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the job specifies a maximum and minimum number of job slots instead of tasks. LSF ignores the number of CPUs constraint during parallel job scheduling and only schedules based on slots.

If `PARALLEL_SCHED_BY_SLOT` is not defined for a re-sizable job, individual allocation requests are constrained by the number of CPUs during scheduling. However, the final re-sizable job allocation may not agree. For example, if an automatically re-

sizable job requests 1 to 4 tasks, on a 2 CPU, 4 slot box, an automatically re-sizable job eventually will use up to 4 slots.

Syntax

```
bsub -n min_task[,max_task]
```

Example

```
bsub -n 4,16 myjob
```

At most, 16 processors can be allocated to this job. If there are less than 16 processors eligible to run the job, this job can still be started as long as the number of eligible processors is greater than or equal to 4.

Restricting job size requested by parallel jobs

Specifying a list of allowed job sizes (number of tasks) in queues or application profiles enables LSF to check the requested job sizes when submitting, modifying, or switching jobs.

About this task

Certain applications may yield better performance with specific job sizes (for example, the power of two, so that the job sizes are x^2), or some sites may want to run all job sizes to generate high cluster resource utilization. The `JOB_SIZE_LIST` parameter in `lsb.queues` or `lsb.applications` allows you to define a discrete list of allowed job sizes for the specified queues or application profiles.

LSF rejects jobs requesting job sizes that are not in this list, or jobs requesting a range of job sizes. The first job size in this list is the default job size, which is the job size assigned to jobs that do not explicitly request a job size. The rest of the list can be defined in any order.

For example, if the job size list for the `queue1` queue allows 2, 4, 8, and 16 tasks, and you submit a parallel job requesting 10 tasks in this queue (`bsub -q queue1 -n 10 ...`), that job is rejected because the job size of 10 is not explicitly allowed in the list. To assign a default job size of 4, specify 4 as the first value in the list, and job submissions that do not request a job size are automatically assigned a job size of 4 (`JOB_SIZE_LIST=4 2 8 16`).

When using resource requirements to specify job size, the request must specify a single fixed job size and not multiple values or a range of values:

- When using compound resource requirements with `-n` (that is, both `-n` and `-R` options), ensure that the compound resource requirement matches the `-n` value, which must match a value in the job size list.
- When using compound resource requirements without `-n`, the compound resource requirement must imply a fixed job size number, and the implied total job size must match a value in the job size list.
- When using alternative resource requirements, each of the alternatives must request a fixed job size number, and all alternative values must match the values in the job size list.

For example, the job size list for the `normal` queue allows 2, 4, and 8 tasks, with 2 as the default (`JOB_SIZE_LIST=2 4 8`). For the resource requirement `"2*{-}+{-}"`, the last term (`{-}`) does not contain a fixed number of tasks, so this compound resource requirement is rejected in any queue that has a job size list.

- For the following job submission with the compound resource requirement:

```
bsub -R "2*{-}+{-}" -q normal myjob
```

This job submission is rejected because the compound resource requirement does not contain a fixed number of tasks.

- For the following job submission with the compound resource requirement:

```
bsub -n 4 -R "2*{-}+{-}" -q normal myjob
```

This job submission is accepted because `-n 4` requests a fixed number of tasks, even though the compound resource requirement does not.

- For the following job submission with compound and alternative resource requirements:
`bsub -R "{2*{-}+{-}}||{4*{-}}" -q normal myjob`

This job submission is rejected for specifying a range of values because the first alternative (`2*{-}+{-}`) does not imply a fixed job size.

- For the following job submission with compound and alternative resource requirements for the `interactive` queue:
`bsub -R "{2*{-}+{-}}||{4*{-}}" -q interactive -H myjob`

This job submission is accepted because the `interactive` queue does not have a job size list. However, if you try to modify or switch this job to any queue or application profile with a job size list, and the job has not yet started, the request is rejected. For example, if this job has job ID 123 and is not started, the following request is rejected because the `normal` queue has a job size list:

```
bswitch normal 123
```

Similarly, if the `app1` application profile has the same job size list as the `normal` queue, the following request is also rejected:

```
bmod -app app1 123
```

When defined in both a queue (`lsb.queues`) and an application profile (`lsb.applications`), the job size request must satisfy both requirements. In addition, `JOB_SIZE_LIST` overrides any `TASKLIMIT` (formerly `PROCLIMIT`) parameters defined at the same level. Job size requirements do not apply to queues and application profiles with no job size lists, nor do they apply to other levels of job submissions (that is, host level or cluster level job submissions).

Specify a job size list for queues and application profiles as follows:

Procedure

1. Log on as root or the LSF administrator on any host in the cluster.
2. Define the `JOB_SIZE_LIST` parameter for the specific application profiles (in `lsb.applications`) or queues (in `lsb.queues`).

```
JOB_SIZE_LIST=default_size [size ...]
```

For example,

- `lsb.applications`:

```
Begin Application
NAME = app1
...
JOB_SIZE_LIST=4 2 8 16
...
End Application
```

- `lsb.queues`:

```
Begin Queue
QUEUE_NAME = queue1
...
JOB_SIZE_LIST=4 2 8 16
...
End Queue
```

3. Save the changes to modified the configuration files.
4. Use **badadmin ckconfig** to check the new queue definition. If any errors are reported, fix the problem and check the configuration again.
5. Run **badadmin reconfig** to reconfigure `mbatchd`.

About specifying a first execution host

In general, the first execution host satisfies certain resource requirements that might not be present on other available hosts.

By default, LSF selects the first execution host dynamically according to the resource availability and host load for a parallel job. Alternatively, you can specify one or more first execution host candidates so that LSF selects one of the candidates as the first execution host.

When a first execution host is specified to run the first task of a parallel application, LSF does not include the first execution host or host group in a job resize allocation request.

- [Specifying a first execution host](#)

Specifying a first execution host

Procedure

To specify one or more hosts, host groups, or compute units as first execution host candidates, add the (!) symbol after the host name.

You can specify first execution host candidates at job submission, or in the queue definition.

- [Rules](#)
Follow these guidelines when you specify first execution host candidates:

Job level

Procedure

1. Use the **-m** option of **bsub**:

```
bsub -n 32 -m "hostA! hostB hostgroup1! hostC" myjob
```

The scheduler selects either hostA or a host defined in hostgroup1 as the first execution host, based on the job's resource requirements and host availability.

2. In a multicluster environment, insert the (!) symbol after the cluster name, as shown in the following example:

```
bsub -n 2 -m "host2@cluster2! host3@cluster2" my_parallel_job
```

Queue level

About this task

The queue-level specification of first execution host candidates applies to all jobs submitted to the queue.

Procedure

Specify the first execution host candidates in the list of hosts in the HOSTS parameter in lsb.queues:

```
HOSTS = hostA! hostB hostgroup1! hostC
```

Rules

Follow these guidelines when you specify first execution host candidates:

- If you specify a host group or compute unit, you must first define the host group or compute unit in the file lsb.hosts.
- Do not specify a dynamic host group as a first execution host.
- Do not specify `all`, `allremote`, `others`, or a host partition as a first execution host.
- Do not specify a preference (+) for a host identified by (!) as a first execution host candidate.
- For each parallel job, specify enough regular hosts to satisfy the CPU requirement for the job. Once LSF selects a first execution host for the current job, the other first execution host candidates become unavailable to the current job.
- You cannot specify first execution host candidates when you use the **brun** command.

If the first execution host is incorrect at job submission, the job is rejected. If incorrect configurations exist on the queue level, warning messages are logged and displayed when LSF starts, restarts, or is reconfigured.

Job chunking

Specifying first execution host candidates affects job chunking. For example, the following jobs have different job requirements, and are not placed in the same job chunk:

```
bsub -n 2 -m "hostA! hostB hostC" myjob
bsub -n 2 -m "hostA hostB hostC" myjob
bsub -n 2 -m "hostA hostB! hostC" myjob
```

The requirements of each job in this example are:

- Job 1 must start on hostA
- Job 2 can start and run on hostA, hostB, or hostC
- Job 3 must start on hostB

For job chunking, all jobs must request the same hosts *and* the same first execution hosts (if specified). Jobs that specify a host preference must all specify the same preference.

Resource reservation

If you specify first execution host candidates at the job or queue level, LSF tries to reserve a job slot on the first execution host. If LSF cannot reserve a first execution host job slot, it does not reserve slots on any other hosts.

Compute units

If compute units resource requirements are used, the compute unit containing the first execution host is given priority:

```
bsub -n 64 -m "hg! cu1 cu2 cu3 cu4" -R "cu[pref=config]" myjob
```

In this example the first execution host is selected from the host group `hg`. Next, in the job's allocation list are any appropriate hosts from the same compute unit as the first execution host. Finally, remaining hosts are grouped by compute unit, with compute unit groups appearing in the same order as in the ComputeUnit section of `lsb.hosts`.

Compound resource requirements

If compound resource requirements are being used, the resource requirements specific to the first execution host should appear first:

```
bsub -m "hostA! hg12" -R "1*{select[type==X86_64]rusage[licA=1]} + {select[type==any]}" myjob
```

In this example the first execution host must satisfy: `select[type==X86_64]rusage[licA=1]`

Compute units

Compute units are similar to host groups, with the added feature of granularity allowing the construction of cluster-wide structures that mimic network architecture. Job scheduling using compute unit resource requirements optimizes job placement based on the underlying system architecture, minimizing communications bottlenecks. Compute units are especially useful when running communication-intensive parallel jobs spanning several hosts. Compute units encode cluster network topology for jobs with a lot of communication between processes. For example, compute units can help minimize network latency and take advantage of fast interconnects by placing all job tasks in the same rack instead of making several network hops.

Resource requirement strings can specify compute units requirements such as running a job exclusively (`excl`), spreading a job evenly over multiple compute units (`balance`), or choosing compute units based on other criteria.

Compute unit configuration

To enforce consistency, compute unit configuration has the following requirements:

- Hosts and host groups appear in the finest granularity compute unit type, and nowhere else.
- Hosts appear in the membership list of at most one compute unit of the finest granularity.
- All compute units of the same type have the same type of compute units (or hosts) as members.

Tip: Configure each individual host as a compute unit to use the compute unit features for host level job allocation.

Where to use compute units

LSF compute units can be used in defining the following parameters in LSF configuration files:

- EXCLUSIVE in lsb.queues for the compute unit type allowed for the queue.
- HOSTS in lsb.queues for the hosts on which jobs from this queue can be run.
- RES_REQ in lsb.queues for queue compute unit resource requirements.
- RES_REQ in lsb.applications for application profile compute unit resource requirements.
- [Control job locality using compute units](#)
Compute units are groups of hosts laid out by the LSF administrator and configured to mimic the network architecture, minimizing communications overhead for optimal placement of parallel jobs. Different granularity of compute units provide the flexibility to configure an extensive cluster accurately and run larger jobs over larger compute units.
- [Wildcard and special characters to define names in compute units](#)
You can use special characters when defining compute unit members under the MEMBER column to specify hosts, host groups, and compute units. These are useful to define several names in a single entry such as a range of hosts, or for all names with a certain text string.
- [Define condensed compute units](#)
You can define condensed compute units to display information for its hosts as a summary for the entire group, including the slot usage for each compute unit. This is useful because it allows you to see statistics of the compute unit as a whole instead of having to add up the data yourself. This allows you to better plan the distribution of jobs submitted to the hosts and compute units in your cluster.
- [Import external host groups \(egroup\)](#)
When the membership of a compute unit changes frequently, or when the compute unit contains a large number of members, you can use an external executable called **egroup** to retrieve a list of members rather than having to configure the membership manually. You can write a site-specific egroup executable that retrieves compute unit names and the hosts that belong to each group, and compute units of the finest granularity can contain egroups as members.
- [Use compute units with advance reservation](#)
When running exclusive compute unit jobs (with the resource requirement cu[excl]), an advance reservation can affect hosts outside the advance reservation but in the same compute unit.

Control job locality using compute units

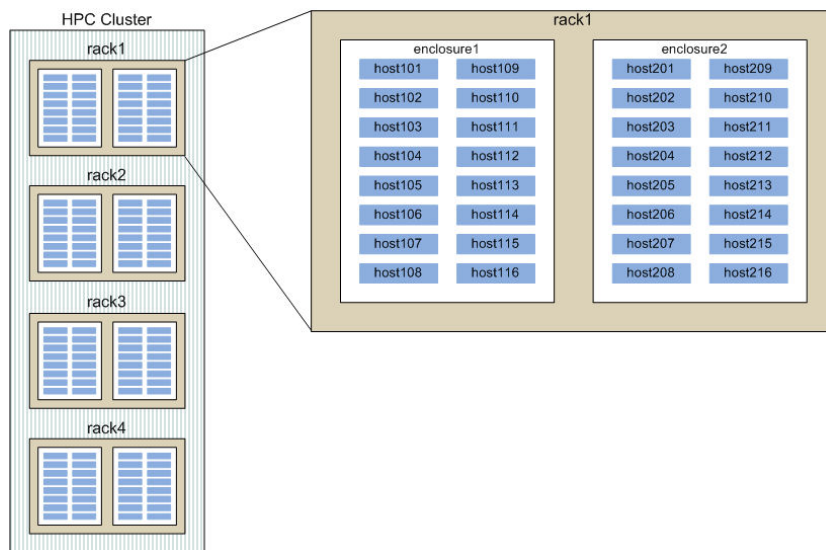
Compute units are groups of hosts laid out by the LSF administrator and configured to mimic the network architecture, minimizing communications overhead for optimal placement of parallel jobs. Different granularity of compute units provide the flexibility to configure an extensive cluster accurately and run larger jobs over larger compute units.

Resource requirement keywords within the compute unit section can be used to allocate resources throughout compute units in manner analogous to host resource allocation. Compute units then replace hosts as the basic unit of allocation for a job.

High performance computing clusters running large parallel jobs spread over many hosts benefit from using compute units. Communications bottlenecks within the network architecture of a large cluster can be isolated through careful configuration of compute units. Using compute units instead of hosts as the basic allocation unit, scheduling policies can be applied on a large scale.

Note:

Configure each individual host as a compute unit to use the compute unit features for host level job allocation.



As indicated in the picture, two types of compute units have been defined in the parameter `COMPUTE_UNIT_TYPES` in `lsb.params`:

```
COMPUTE_UNIT_TYPES= enclosure! rack
```

! indicates the default compute unit type. The first type listed (`enclosure`) is the finest granularity and the only type of compute unit containing hosts and host groups. Coarser granularity `rack` compute units can only contain `enclosures`.

The hosts have been grouped into compute units in the **ComputeUnit** section of `lsb.hosts` as follows (some lines omitted):

```
Begin ComputeUnit
NAME      MEMBER      CONDENSED  TYPE
enclosure1 (host1[01-16])    Y      enclosure
...
enclosure8 (host8[01-16]) Y      enclosure
rack1      (enclosure[1-2])  Y      rack
rack2      (enclosure[3-4]) Y      rack
rack3      (enclosure[5-6]) Y      rack
rack4      (enclosure[7-8]) Y      rack
End ComputeUnit
```

This example defines 12 compute units, all of which have condensed output:

- `enclosure1` through `enclosure8` are the finest granularity, and each contain 16 hosts.
- `rack1`, `rack2`, `rack3`, and `rack4` are the coarsest in granularity, and each contain 2 enclosures.

Syntax

The **cu** string supports the following syntax:

`cu[balance]`

All compute units used for this job should contribute the same number of slots (to within one slot). It provides a balanced allocation over the fewest possible compute units.

`cu[pref=bestfit]`

The job will be placed to span as few compute units as possible (given the current resource availability) while preferring to use already occupied resources for the job, in order to try to reduce fragmentation in the cluster. Do not use with the `balance` keyword.

`cu[pref=config]`

Compute units for this job are considered in the order that they appear in the `lsb.hosts` configuration file. This is the default value.

`cu[pref=minavail]`

Compute units with the fewest available slots are considered first for this job. It is useful for smaller jobs (both sequential and parallel) since this minimizes the possible fragmentation of compute units, leaving whole compute units free for larger jobs.

`cu[pref=maxavail]`

Compute units with the most available slots are considered first for this job.

cu[maxcus=*number*]

Maximum number of compute units the job can run across.

cu[usablecuslots=*number*]

All compute units used for this job should contribute the same minimum *number* of slots. At most the final allocated compute unit can contribute fewer than *number* slots.

cu[type=*cu_type*]

Type of compute unit being used, where *cu_type* is one of the types defined by COMPUTE_UNIT_TYPES in lsb.params.

The default is the compute unit type listed first in lsb.params.

cu[excl]

Compute units used exclusively for the job. Must be enabled by EXCLUSIVE in lsb.queues.

Continuing with the example shown above, assume lsb.queues contains the parameter definition EXCLUSIVE=CU[rack] and that the slots available for each compute unit are shown under **MAX** in the condensed display from **bhosts**, where

HOST_NAME refers to the compute unit:

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
enclosure1	ok	-	64	34	34	0	0	0
enclosure2	ok	-	64	54	54	0	0	0
enclosure3	ok	-	64	46	46	0	0	0
enclosure4	ok	-	64	44	44	0	0	0
enclosure5	ok	-	64	45	45	0	0	0
enclosure6	ok	-	64	44	44	0	0	0
enclosure7	ok	-	32	0	0	0	0	0
enclosure8	ok	-	64	0	0	0	0	0
rack1	ok	-	128	88	88	0	0	0
rack2	ok	-	128	90	90	0	0	0
rack3	ok	-	128	89	89	0	0	0
rack4	ok	-	128	0	0	0	0	0

Based on the 12 configured compute units, jobs can be submitted with a variety of compute unit requirements.

Use compute units

1. bsub -R "cu[]" -n 64 ./app

This job is restricted to compute units of the default type `enclosure`. The default **pref=config** applies, with compute units considered in configuration order. The job runs on 30 slots in `enclosure1`, 10 slots in `enclosure2`, 18 slots in `enclosure3`, and 6 slots in `enclosure4` for a total of 64 slots.

2. Compute units can be considered in order of most free slots or fewest free slots, where free slots include any slots available and not occupied by a running job.

bsub -R "cu[pref=minavail]" -n 32 ./app

This job is restricted to compute units of the default type `enclosure` in the order **pref=minavail**. Compute units with the fewest free slots are considered first. The job runs on 10 slots in `enclosure2`, 18 slots in `enclosure3` and 4 slots in `enclosure5` for a total of 32 slots.

3. bsub -R "cu[type=rack:pref=maxavail]" -n 64 ./app

This job is restricted to compute units of the default type `enclosure` in the order **pref=maxavail**. Compute units with the most free slots are considered first. The job runs on 64 slots in `enclosure8`.

Localized allocations

Jobs can be run over a limited number of compute units using the **maxcus** keyword.

1. bsub -R "cu[pref=maxavail:maxcus=1]" ./app

This job is restricted to a single enclosure, and compute units with the most free slots are considered first. The job requirements are satisfied by `enclosure8` which has 64 free slots.

2. bsub -n 64 -R "cu[maxcus=3]" ./app

This job requires a total of 64 slots over 3 enclosures or less. Compute units are considered in configuration order. The job requirements are satisfied by the following allocation:

Compute unit	Free slots
--------------	------------

Compute unit	Free slots
enclosure1	30
enclosure3	18
enclosure4	16

Balanced slot allocations

Balanced allocations split jobs evenly between compute units, which increases the efficiency of some applications.

1. `bsub -n 80 -R "cu[balance:maxcus=4]" ./app`

This job requires a balanced allocation over the fewest possible compute units of type `enclosure` (the default type), with a total of 80 slots. Since none of the configured enclosures have 80 slots, 2 compute units with 40 slots each are used, satisfying the **maxcus** requirement to use 4 compute units or less.

The keyword **pref** is not included so the default order of **pref=config** is used. The job requirements are satisfied by 27 slots in `enclosure1`, 27 slots in `enclosure7`, and 26 slots in `enclosure8`, for a total of 80 slots.

2. `bsub -n 64 -R "cu[balance:type=rack:pref=maxavail]" ./app`

This job requires a balanced allocation over the fewest possible compute units of type `rack`, with a total of 64 slots. Compute units with the most free slots are considered first, in the order `rack4`, `rack1`, `rack3`, `rack2`. The job requirements are satisfied by `rack4`.

3. `bsub -n "40,80" -R "cu[balance:pref=minavail]" ./app`

This job requires a balanced allocation over compute units of type `rack`, with a range of 40 to 80 slots. Only the minimum number of slots is considered when a range is specified along with keyword **balance**, so the job needs 40 slots. Compute units with the fewest free slots are considered first.

Because **balance** uses the fewest possible compute units, racks with 40 or more slots are considered first, namely `rack1` and `rack4`. The rack with the fewest available slots is then selected, and all job requirements are satisfied by 40 slots in `enclosure8`.

Balanced host allocations

Using **balance** and **ptile** together within the requirement string results in a balanced host allocation over compute units, and the same number of slots from each host. The final host may provide fewer slots if required.

- `bsub -n 64 -R "cu[balance] span[ptile=4]" ./app`

This job requires a balanced allocation over the fewest possible compute units of type `enclosure`, with a total of 64 slots. Each host used must provide 4 slots. Since `enclosure8` has 64 slots available over 16 hosts (4 slots per host), it satisfies the job requirements.

Had `enclosure8` not satisfied the requirements, other possible allocations in order of consideration (fewest compute units first) include:

Number of compute units	Number of hosts
2	8+8
3	5+5+6
4	4+4+4+4
5	3+3+3+4

Minimum slot allocations

Minimum slot allocations result in jobs spreading over fewer compute units, and ignoring compute units with few hosts available.

1. `bsub -n 45 -R "cu[usablecuslots=10:pref=minavail]" ./app`

This job requires an allocation of at least 10 slots in each enclosure, except possibly the last one. Compute units with the fewest free slots are considered first. The requirements are satisfied by a slot allocation of:

Compute unit	Number of slots
enclosure2	10
enclosure5	19
enclosure4	16

2. `bsub -n "1,140" -R "cu[usablecuslots=20]" ./app`

This job requires an allocation of at least 20 slots in each enclosure, except possibly the last one. Compute units are considered in configuration order and as close to 140 slots are allocated as possible. The requirements are satisfied by an allocation of 140 slots, where only the last compute unit has fewer than 20 slots allocated as follows:

Compute unit	Number of slots
enclosure1	30
enclosure4	20
enclosure6	20
enclosure7	64
enclosure2	6

Exclusive compute unit jobs

Because `EXCLUSIVE=CU[rack]` in `lsb.queues`, jobs may use compute units of type `rack` or finer granularity type **enclosure** exclusively. Exclusive jobs lock all compute units they run in, even if not all slots are being used by the job. Running compute unit exclusive jobs minimizes communications slowdowns resulting from shared network bandwidth.

1. `bsub -R "cu[excl:type=enclosure]" ./app`

This job requires exclusive use of an enclosure with compute units considered in configuration order. The first enclosure not running any jobs is `enclosure7`.

2. Using **excl** with **usablecuslots**, the job avoids compute units where a large portion of the hosts are unavailable.

```
bsub -n 90 -R "cu[excl:usablecuslots=12:type=enclosure]" ./app
```

This job requires exclusive use of compute units, and will not use a compute unit if fewer than 12 slots are available. Compute units are considered in configuration order. In this case the job requirements are satisfied by 64 slots in `enclosure7` and 26 slots in `enclosure8`.

3. `bsub -R "cu[excl]" ./app`

This job requires exclusive use of a rack with compute units considered in configuration order. The only rack not running any jobs is `rack4`.

Reservation

Compute unit constraints such as keywords **maxcus**, **balance**, and **excl** can result in inaccurately predicted start times from default LSF resource reservation. Time-based resource reservation provides a more accurate pending job predicted start time. When calculating job a time-based predicted start time, LSF considers job scheduling constraints and requirements, including job topology and resource limits, for example.

Host-level compute units

Configuring each individual host as a compute unit allows you to use the compute unit features for host level job allocation. Consider an example where one type of compute units has been defined in the parameter `COMPUTE_UNIT_TYPES` in `lsb.params`:

```
COMPUTE_UNIT_TYPES= host!
```

The hosts have been grouped into compute hosts in the **ComputeUnit** section of `lsb.hosts` as follows:

```
Begin ComputeUnit
NAME  MEMBER  TYPE
h1    host1    host
h2    host2    host
```

```
...
h50    host50    host
End ComputeUnit
```

Each configured compute unit of default type `host` contains a single host.

Order host allocations

Using the compute unit keyword `pref`, hosts can be considered in order of most free slots or fewest free slots, where free slots include any slots available and not occupied by a running job:

1. `bsub -R "cu[]" ./app`
Compute units of default type `host`, each containing a single host, are considered in configuration order.
2. `bsub -R "cu[pref=minavail]" ./app`
Compute units of default type `host` each contain a single host. Compute units with the fewest free slots are considered first.
3. `bsub -n 20 -R "cu[pref=maxavail]" ./app`
Compute units of default type `host` each contain a single host. Compute units with the most free slots are considered first. A total of 20 slots are allocated for this job.

Limit hosts in allocations

Use the compute unit keyword `maxcus` to specify the maximum number of hosts allocated to a job can be set:

- `bsub -n 12 -R "cu[pref=maxavail:maxcus=3]" ./app`
Compute units of default type `host` each contain a single host. Compute units with the most free slots are considered first. This job requires an allocation of 12 slots over at most 3 hosts.

Balanced slot allocations

Using the compute unit keyword `balance`, jobs can be evenly distributed over hosts:

1. `bsub -n 9 -R "cu[balance]" ./app`
Compute units of default type `host`, each containing a single host, are considered in configuration order. Possible balanced allocations are:

Compute units	Hosts	Number of slots per host
1	1	9
2	2	4, 5
3	3	3, 3, 3
4	4	2, 2, 2, 3
5	5	2, 2, 2, 2, 1
6	6	2, 2, 2, 1, 1, 1
7	7	2, 2, 1, 1, 1, 1, 1
8	8	2, 1, 1, 1, 1, 1, 1, 1
9	9	1, 1, 1, 1, 1, 1, 1, 1, 1

2. `bsub -n 9 -R "cu[balance:maxcus=3]" ./app`
Compute units of default type `host`, each containing a single host, are considered in configuration order. Possible balanced allocations are 1 host with 9 slots, 2 hosts with 4 and 5 slots, or 3 hosts with 3 slots each.

Minimum slot allocations

Using the compute unit keyword `usablecuslots`, hosts are only considered if they have a minimum number of slots free and usable for this job:

1. `bsub -n 16 -R "cu[usablecuslots=4]" ./app`

Compute units of default type `host`, each containing a single host, are considered in configuration order. Only hosts with 4 or more slots available and not occupied by a running job are considered. Each host (except possibly the last host allocated) must contribute at least 4 slots to the job.

2. `bsub -n 16 -R "rusage[mem=1000] cu[usablecuslots=4]" ./app`

Compute units of default type `host`, each containing a single host, are considered in configuration order. Only hosts with 4 or more slots available, not occupied by a running job, and with 1000 memory units are considered. A host with 10 slots and 2000 units of memory, for example, will only have 2 slots free that satisfy the memory requirements of this job.

Best fit for job placement

Some users may want LSF to place jobs with an optimal placement with respect to compute units, without having to specify different requirements for different jobs. For this purpose, LSF has the "bestfit" value for the `pref` option. For example, `bsub -R "cu [pref=bestfit]" ...`

When this algorithm is used, LSF places the job to span as few compute units as possible (given the current resource availability) while preferring to use already occupied resources for the job, in order to try to reduce fragmentation in the cluster. This is done by considering the underlying network topology of the cluster, as specified in LSF Compute Units (CUs).

The "bestfit" value is different from other compute unit placement algorithms in that it considers multiple levels of the compute unit hierarchy, if applicable. It also allows both large and small jobs to use the same compute unit requirements. Ultimately, it simplifies the specification of compute unit requirements.

The `[pref=bestfit]` option can be used together with:

- `maxcus`
- `type=<type>`
- `usablecuslots=<num>`
- `excl`
- compute unit preference `-m` option

The `[pref=bestfit]` algorithm works in two phases.

Examples of the usage of `[pref=bestfit]` and how the algorithm finds the best fit of compute units:

- `bsub -R "cu[excl:pref=bestfit]" -n 10 myjob`
An exclusive job on a best fit of compute units
- `bsub -R "cu[pref=bestfit:usablecuslots=5]" -n 15 myjob`
Best-fit job placement with minimum slot allocations for the default level and below.
- `bsub -m "cu1+10 cu3+1" -R "cu[pref=bestfit]" -n 10 myjob`
Best-fit job placement while taking host preference into consideration. Host preference has a higher priority.
- `bsub -R "cu[pref=bestfit:maxcus=2:type=switch]" -n 10 myjob`
Best-fit job placement with specified `maxcus` for the default level and below. For the second phase, LSF considers the lowest `maxcus` among the user-specified value and the number of compute units that is calculated in the first phase.
- `bsub -R "8*{select[LN]} + {cu[pref=bestfit]}" myjob`
A job with a compound resource requirement.

Specify the compute unit order with the host preference feature

Before LSF 10.1, the compute unit order was determined only by the compute unit `pref` policies (`cu[pref=config | maxavail | minavail]`). Host preference (specified by `-m` or `HOSTS` in `lsb.queues`) only affected the host order within each compute unit. In LSF 10.1, considering the customer's requirement to specify compute unit order more flexibly, this behavior has been changed.

Currently, LSF allows use of the host preference to specify compute unit order along with `cu[pref=config | maxavail | minavail]` policy. The following example illustrates use of the `-m` preference to specify the compute unit's order as: `cu1>cu2>cu3>cu4`.

```
bsub -n 2 -m "cu1+10 cu2+5 cu3+1 cu4" -R "cu[]" ./app
```

Host preference works along with `cu[pref=config | maxavail | minavail]` in the following manner to determine compute unit order:

1. LSF calculates the compute unit preference according to the host preference, taking the highest preference of hosts within the compute unit as the compute unit preference. For example:

- a. In the following example, in which `h1 h2` belong to `cu1` and `h3 h4` belong to `cu2`, according to the candidate host preference, LSF determines that the `cu1` preference is 10 and the `cu2` preference is 0.

```
bsub -n 2 -m "h1+10 others" -R "cu[pref=minavail]" ./app
```

- b. In the following example, in which `h1 h2` belong to `cu1` and `h3 h4` belong to `cu2`, according to the candidate host preference, LSF determines that the `cu1` preference is 10 and `cu2` preference is 9.

```
bsub -n 2 -m "h1+10 h2+1 h3+9 h4+9" -R "cu[pref=minavail]" ./app
```

2. LSF determines the compute unit order as follows:

- a. When the compute unit preference calculated in step 1, above, differs, LSF orders the compute unit by the compute unit preference, considering the compute unit with the higher preference first. For example:
 - i. Because `cu1` has a higher preference than `cu2` in the following example, LSF first considers `cu1`, then `cu2`, without regard for the `cu[pref=config | maxavail | minavail]` policy.

```
bsub -n 2 -m "cu1+10 cu2" -R "cu[pref=minavail]" ./app
```

- ii. `cu[pref=maxvail]` and `cu[pref=config]` also follow this policy.

- b. When the compute unit preference calculated in step 1, above, is the same, LSF orders the compute unit using `cu[pref=config | maxavail | minavail]`. For example:

- i. Because all compute units in the following example do not have the preference, LSF uses standard logic to determine compute unit order. `pref=minavail` takes affect.

```
bsub -n 2 -R "cu[pref=minavail]" ./app
```

- ii. Because all compute units in the following example have the same preference, LSF uses standard logic to determine compute unit order. `pref=minavail` takes affect.

```
bsub -n 2 -m "cu1+10 cu2+10" -R "cu[pref=minavail]" ./app
```

- iii. `cu[pref=maxvail]` and `cu[pref=config]` also follow this policy.

3. After LSF determines the compute unit order, the LSF scheduler adjusts the candidate host list according to the compute unit order.

For resource reservation, the host preference is considered when determining the resource on which the compute unit is reserved first. The default, `pref=config`, however, is always used.

The first-execution host works with the compute unit feature as follows:

- When mandatory first-execution hosts are used together with the compute unit feature, the compute unit that contains the first-execution host is given first preference among compute units. The remaining compute units are ordered according to the calculated compute unit preference. As previously in LSF, exactly one of the candidate first-execution hosts can be used in the job's allocation.

Note:

1. Compute unit policy `pref=config|maxavail|minavail` does not affect the order of compute units specified as first-execution hosts.

This means that even when under the following circumstances:

- a. You use the following options:

```
-m "cu1! cu2! cu3!" -R "cu[pref=minavail]"
```

- b. `cu1` has 15 free slots.
- c. `cu2` has 10 free slots.
- d. `cu3` has 20 free slots.

LSF does not reorder the first-execution host list under the compute unit `cu[pref=config|maxavail|minavail]` policy.

2. When using host preference to determine compute unit preference, the host specified as first-execution host is not considered. Namely, LSF counts only the preference of hosts that are not specified as first-execution hosts. For example, let `cu1`, `cu2`, `cu3`, `cu4` denote compute units, all of the same type. Let `hg` denote a host group containing one host from each compute unit.

The user submits the following job:

```
bsub -n 64 -m "hg! cu1+1 cu2+2 cu3+3 cu4+4" -R "cu[pref=config]" ./app
```

When the job is dispatched, exactly one host from `hg` appears in the job's allocation list. (This host should appear in the first position of the list.) Next in the list are zero or more hosts from the same compute unit that as the first-execution host. The remaining hosts from the other compute units appear grouped by compute unit, with the groups themselves appearing in order, according to the high-low preference of the compute unit. For example:

- a. If `h1` from `cu1` is selected as the first-execution host, the final compute unit order would be `cu1>cu4>cu3>cu2`
- b. If `h2` from `cu2` is selected as the first-execution host, the final compute unit order would be `cu2>cu4>cu3>cu1`
- c. If `h3` from `cu3` is selected as the first-execution host, the final compute unit order would be `cu3>cu4>cu2>cu1`
- d. If `h4` from `cu4` is selected as the first-execution host, the final compute unit order would be `cu4>cu3>cu2>cu1`

- [Configuring compute units](#)

Configuring compute units

Procedure

1. Log in as the LSF administrator to any host in the cluster.
2. Open `lsb.params`.
3. Add the `COMPUTE_UNIT_TYPES` parameter if it does not already exist and list your compute unit types in order of granularity (finest first).
`COMPUTE_UNIT_TYPES=enclosure rack cabinet`
4. Save your changes.
5. Open `lsb.hosts`.
6. Add the `ComputeUnit` section if it does not exist.

```
Begin ComputeUnit  
NAME           MEMBER           TYPE  
encl1          (hostA hg1)           enclosure  
encl2          (hostC hostD)         enclosure  
encl3          (hostE hostF)         enclosure  
encl4          (hostG hg2)           enclosure  
rack1          (encl1 encl2)         rack  
rack2          (encl3 encl4)         rack  
cab1          (rack1 rack2)         cabinet  
End ComputeUnit
```

7. Enter a compute unit name under the `NAME` column.
External compute units must be defined in the **egroup** executable.
8. Specify hosts or host groups in the `MEMBER` column of the finest granularity compute unit type. Specify compute units in the `MEMBER` column of coarser compute unit types.
(Optional) To tell LSF that the compute unit members of a finest granularity compute unit should be retrieved using **egroup**, put an exclamation mark (!) in the `MEMBER` column.
9. Specify the type of compute unit in the `TYPE` column.
10. Save your changes.
11. Run **badadmin ckconfig** to check the compute unit definition. If any errors are reported, fix the problem and check the configuration again.
12. Run **badadmin mbdrestart** to apply the new configuration.
To view configured compute units, run **bmgroup -cu**.

Wildcard and special characters to define names in compute units

You can use special characters when defining compute unit members under the MEMBER column to specify hosts, host groups, and compute units. These are useful to define several names in a single entry such as a range of hosts, or for all names with a certain text string.

When defining host, host group, and compute unit members of compute units, you can use string literals and the following special characters:

- Use a tilde (~) to exclude specified hosts, host groups, or compute units from the list. The tilde can be used in conjunction with the other special characters listed below. The following example matches all hosts in `group12` except for `hostA`, and `hostB`:

```
... (group12 ~hostA ~hostB)
```

- Use an asterisk (*) as a wildcard character to represent any number of characters. The following example matches all hosts beginning with the text string "hostC" (such as `hostCa`, `hostC1`, or `hostCZ1`):

```
... (hostC*)
```

- Use square brackets with a hyphen (`[integer1 - integer2]`) to define a range of non-negative integers at the end of a name. The first integer must be less than the second integer. The following example matches all hosts from `hostD51` to `hostD100`:

```
... (hostD[51-100])
```

- Use square brackets with commas (`[integer1, integer2 ...]`) to define individual non-negative integers at the end of a name. The following example matches `hostD101`, `hostD123`, and `hostD321`:

```
... (hostD[101,123,321])
```

- Use square brackets with commas and hyphens (such as `[integer1 - integer2, integer3, integer4 - integer5]`) to define different ranges of non-negative integers at the end of a name. The following example matches all hosts from `hostD1` to `hostD100`, `hostD102`, all hosts from `hostD201` to `hostD300`, and `hostD320`:

```
... (hostD[1-100,102,201-300,320])
```

Restrictions

You cannot use more than one set of square brackets in a single compute unit definition.

The following example is *not* correct:

```
... (hostA[1-10]B[1-20] hostC[101-120])
```

The following example is correct:

```
... (hostA[1-20] hostC[101-120])
```

The keywords `all`, `allremote`, `all@cluster`, `other` and `default` cannot be used when defining compute units.

Define condensed compute units

You can define condensed compute units to display information for its hosts as a summary for the entire group, including the slot usage for each compute unit. This is useful because it allows you to see statistics of the compute unit as a whole instead of having to add up the data yourself. This allows you to better plan the distribution of jobs submitted to the hosts and compute units in your cluster.

To define condensed compute units, add a CONDENSE column to the ComputeUnit section. Under this column, enter **Y** to define a condensed host group or **N** to define an noncondensed host group, as shown in the following:

```
Begin ComputeUnit
NAME      CONDENSE  MEMBER                                TYPE
enclA     Y          (hostA hostB hostD)                  enclosure
enclB     N          (hostC hostE)                        enclosure
End HostGroup
```

The following commands display condensed host information:

- **bhosts**
- **bhosts -w**
- **bjobs**
- **bjobs -w**

Use **bmgroup -l** to see whether host groups are condensed or not.

Import external host groups (egroup)

When the membership of a compute unit changes frequently, or when the compute unit contains a large number of members, you can use an external executable called **egroup** to retrieve a list of members rather than having to configure the membership manually. You can write a site-specific egroup executable that retrieves compute unit names and the hosts that belong to each group, and compute units of the finest granularity can contain egroups as members.

For information about how to use the external host and user groups feature, see [External host and user groups](#).

Use compute units with advance reservation

When running exclusive compute unit jobs (with the resource requirement `cu[excl]`), an advance reservation can affect hosts outside the advance reservation but in the same compute unit.

- An exclusive compute unit job dispatched to a host inside the advance reservation will lock the entire compute unit, including any hosts outside the advance reservation.
- An exclusive compute unit job dispatched to a host outside the advance reservation will lock the entire compute unit, including any hosts inside the advance reservation.

Ideally all hosts belonging to a compute unit should be inside or outside of an advance reservation.

Control processor allocation across hosts

Sometimes you need to control how the selected processors for a parallel job are distributed across the hosts in the cluster.

You can control this at the job level or at the queue level. The queue specification is ignored if your job specifies its own locality.

Specify parallel job locality at the job level

By default, LSF does allocate the required processors for the job from the available set of processors.

A parallel job may span multiple hosts, with a specifiable number of processes allocated to each host. A job may be scheduled on to a single multiprocessor host to take advantage of its efficient shared memory, or spread out on to multiple hosts to take advantage of their aggregate memory and swap space. Flexible spanning may also be used to achieve parallel I/O.

You are able to specify “select all the processors for this parallel batch job on the same host”, or “do not choose more than *n* processors on one host” by using the `span` section in the resource requirement string (**`bsub -R` or `RES_REQ`** in the queue definition in `lsb.queues`).

If `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the `span` string is used to control the number of job slots instead of processors.

Syntax

The `span` string supports the following syntax:

`span[hosts=1]`

Indicates that all the processors allocated to this job must be on the same host.

`span[ptile=value]`

Indicates the number of processors on each host that should be allocated to the job, where *value* is one of the following:

- Default `ptile` value, specified by *n* processors. In the following example, the job requests 4 processors on each available host, regardless of how many processors the host has:

```
span[ptile=4]
```

- Predefined `ptile` value, specified by '!'. The following example uses the predefined maximum job slot limit `lsb.hosts` (MXJ per host type/model) as its value:

```
span[ptile='!']
```

Tip:

If the host or host type/model does not define MXJ, the default predefined `ptile` value is 1.

- Predefined `ptile` value with optional multiple `ptile` values, per host type or host model:
 - For host type, you must specify `same[type]` in the resource requirement. In the following example, the job requests 8 processors on a host of type `HP`, and 2 processors on a host of type `LINUX`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host types:
- For host model, you must specify `same[model]` in the resource requirement. In the following example, the job requests 4 processors on hosts of model `PC1133`, and 2 processors on hosts of model `PC233`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host models:

```
span[ptile='!',HP:8,LINUX:2] same[type]
```

```
span[ptile='!',PC1133:4,PC233:2] same[model]
```

`span[hosts=-1]`

Disables `span` setting in the queue. LSF allocates the required processors for the job from the available set of processors.

For example,

```
bsub -q super -R "span[hosts=-1]" -n 5 sleep 180
```

Specify multiple `ptile` values

In a `span` string with multiple `ptile` values, you must specify a predefined default value (`ptile='!'`) and either host model or host type.

You can specify both type and model in the `span` section in the resource requirement string, but the `ptile` values must be the same type.

If you specify `same[type:model]`, you *cannot* specify a predefined `ptile` value (!) in the `span` section.

Restriction:

Under bash 3.0, the exclamation mark (!) is not interpreted correctly by the shell. To use predefined `ptile` value (`ptile='!'`), use the `+H` option to disable '!' style history substitution in bash (`sh +H`).

LINUX and HP are both host types and can appear in the same `span` string. The following `span` string is valid:

```
same[type] span[ptile='!',LINUX:2,HP:4]
```

PC233 and PC1133 are both host models and can appear in the same `span` string. The following `span` string is valid:

```
same[model] span[ptile='!',PC233:2,PC1133:4]
```

You cannot mix host model and host type in the same `span` string. The following `span` strings are *not* correct:

```
span[ptile='!',LINUX:2,PC1133:4] same[model]  
span[ptile='!',LINUX:2,PC1133:4] same[type]
```

The `LINUX` host type and `PC1133` host model cannot appear in the same `span` string.

Multiple `ptile` values for a host type

For host type, you must specify `same[type]` in the resource requirement. For example:

```
span[ptile='!',HP:8,SOL:8,LINUX:2] same[type]
```

The job requests 8 processors on a host of type `HP` or `SOL`, and 2 processors on a host of type `LINUX`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host types.

Multiple `ptile` values for a host model

For host model, you must specify `same[model]` in the resource requirement. For example:

```
span[ptile='!',PC1133:4,PC233:2] same[model]
```

The job requests 4 processors on hosts of model `PC1133`, and 2 processors on hosts of model `PC233`, and the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host models.

Examples

```
bsub -n 4 -R "span[hosts=1]" myjob
```

Runs the job on a host that has at least 4 processors currently eligible to run the 4 components of this job.

```
bsub -n 4 -R "span[ptile=2]" myjob
```

Runs the job on 2 hosts, using 2 processors on each host. Each host may have more than 2 processors available.

```
bsub -n 4 -R "span[ptile=3]" myjob
```

Runs the job on 2 hosts, using 3 processors on the first host and 1 processor on the second host.

```
bsub -n 4 -R "span[ptile=1]" myjob
```

Runs the job on 4 hosts, even though some of the 4 hosts may have more than one processor currently available.

```
bsub -n 4 -R "type==any same[type] span[ptile='!',LINUX:2,HP:4]" myjob
```

Submits `myjob` to request 4 processors running on 2 hosts of type `LINUX` (2 processors per host), or a single host of type `HP`, or for other host types, the predefined maximum job slot limit in `lsb.hosts` (MXJ).

```
bsub -n 16 -R "type==any same[type] span[ptile='!',HP:8,SOL:8,LINUX:2]" myjob
```

Submits `myjob` to request 16 processors on 2 hosts of type `HP` or `SOL` (8 processors per hosts), or on 8 hosts of type `LINUX` (2 processors per host), or the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host types.

```
bsub -n 4 -R "same[model] span[ptile='!',PC1133:4,PC233:2]" myjob
```

Submits `myjob` to request a single host of model `PC1133` (4 processors), or 2 hosts of model `PC233` (2 processors per host), or the predefined maximum job slot limit in `lsb.hosts` (MXJ) for other host models.

Specify parallel job locality at the queue level

The queue may also define the locality for parallel jobs using the `RES_REQ` parameter.

Run parallel processes on homogeneous hosts

Parallel jobs run on multiple hosts. If your cluster has heterogeneous hosts some processes from a parallel job may for example, run on Solaris. However, for performance reasons you may want all processes of a job to run on the same type of host instead of having some processes run on one type of host and others on another type of host.

You can use the same section in the resource requirement string to indicate to LSF that processes are to run on one type or model of host. You can also use a custom resource to define the criteria for homogeneous hosts.

Run all parallel processes on the same host type

```
bsub -n 4 -R"select[type==HP6 || type==SOL11] same[type]" myjob
```

Allocate 4 processors on the same host type: either HP, or Solaris 11, but not both.

Run all parallel processes on the same host type and model

```
bsub -n 6 -R"select[type==any] same[type:model]" myjob
```

Allocate 6 processors on any host type or model as long as all the processors are on the same host type and model.

Run all parallel processes on hosts in the same high-speed connection group

```
bsub -n 12 -R "select[type==any && (hgconnect==hg1 |  
| hgconnect==hg2 || hgconnect==hg3)] same[hgconnect:type]" myjob
```

For performance reasons, you want to have LSF allocate 12 processors on hosts in high-speed connection group hg1, hg2, or hg3, but not across hosts in hg1, hg2 or hg3 at the same time. You also want hosts that are chosen to be of the same host type.

This example reflects a network in which network connections among hosts in the same group are high-speed, and network connections between host groups are low-speed.

In order to specify this, you create a custom resource `hgconnect` in `lsf.shared`.

```
Begin Resource
RESOURCENAME  TYPE    INTERVAL    INCREASING    RELEASE    DESCRIPTION
hgconnect     STRING    ()          ()            ()         (OS release)
...
End Resource
```

In the `lsf.cluster.cluster_name` file, identify groups of hosts that share high-speed connections.

```
Begin ResourceMap
RESOURCENAME  LOCATION
hgconnect     (hg1@[hostA hostB] hg2@[hostD hostE] hg3@[hostF hostG hostX])
End ResourceMap
```

If you want to specify the same resource requirement at the queue level, define a custom resource in `lsf.shared` as in the previous example, map hosts to high-speed connection groups in `lsf.cluster.cluster_name`, and define the following queue in `lsb.queues`:

```
Begin Queue
QUEUE_NAME = My_test
PRIORITY = 30
NICE = 20 RES REQ = "select[mem > 1000 && type==any && (hgconnect==hg1 ||  
hgconnect==hg2 || hgconnect==hg3)] same[hgconnect:type]"
DESCRIPTION = either hg1 or hg2 or hg3
End Queue
```

This example allocates processors on hosts that:

- Have more than 1000 MB in memory
- Are of the same host type
- Are in high-speed connection group hg1 or hg2 or hg3

Limit the number of processors allocated

Use the TASKLIMIT parameter in lsb.queues or lsb.applications to limit the number of tasks that can be allocated to a parallel job.

Syntax

`TASKLIMIT = [minimum_limit [default_limit]] maximum_limit`

All limits must be positive numbers greater than or equal to 1 that satisfy the following relationship:

$1 \leq \text{minimum} \leq \text{default} \leq \text{maximum}$

You can specify up to three limits in the TASKLIMIT parameter:

If you specify ...	Then ...
One limit	It is the maximum task limit. The minimum and default limits are set to 1.
Two limits	The first is the minimum task limit, and the second is the maximum. The default is set equal to the minimum. The minimum must be less than or equal to the maximum.
Three limits	The first is the minimum task limit, the second is the default task limit, and the third is the maximum. The minimum must be less than the default and the maximum.

How TASKLIMIT affects submission of parallel jobs

The -n option of **bsub** specifies the number of tasks to be used by a parallel job, subject to the task limits of the queue or application profile.

Jobs that specify fewer tasks than the minimum TASKLIMIT or more tasks than the maximum TASKLIMIT are rejected.

If a default value for TASKLIMIT is specified, jobs submitted without specifying -n use the default number of TASKLIMIT. If the queue or application profile has only minimum and maximum values for TASKLIMIT, the number of tasks is equal to the minimum value. If only a maximum value for TASKLIMIT is specified, or no TASKLIMIT is specified, the number of processors is equal to 1.

Incorrect task limits are ignored, and a warning message is displayed when LSF is reconfigured or restarted. A warning message is also logged to the mbatchd log file when LSF is started.

Change TASKLIMIT

If you change the TASKLIMIT parameter, the new task limit does not affect running jobs. Pending jobs with no task requirements use the new default TASKLIMIT value. If the pending job does not satisfy the new task limits, it remains in PEND state, and the pending reason changes to the following:

Job no longer satisfies TASKLIMIT configuration

If the TASKLIMIT specification is incorrect (for example, too many parameters), a reconfiguration error message is issued. Reconfiguration proceeds and the incorrect TASKLIMIT is ignored.

Resizable jobs

Resizable job allocation requests obey the TASKLIMIT definition in both application profiles and queues. When the maximum job task request is greater than the maximum task definition in TASKLIMIT, LSF chooses the minimum value of both. For example, if a job asks for `-n 1, 4`, but TASKLIMIT is defined as `2 2 3`, the maximum task request for the job is 3 rather than 4.

Automatic queue selection

When you submit a parallel job without specifying a queue name, LSF automatically selects the most suitable queue from the queues listed in the DEFAULT_QUEUE parameter in lsb.params or the LSB_DEFAULTQUEUE environment variable. Automatic queue selection takes into account any maximum and minimum TASKLIMIT values for the queues available for automatic selection.

If you specify `-n min_task,max_task`, but do not specify a queue, the first queue that satisfies the task requirements of the job is used. If no queue satisfies the task requirements, the job is rejected.

For example, queues with the following TASKLIMIT values are defined in lsb.queues:

- queueA with TASKLIMIT=1 1 1
- queueB with TASKLIMIT=2 2 2
- queueC with TASKLIMIT=4 4 4
- queueD with TASKLIMIT=8 8 8
- queueE with TASKLIMIT=16 16 16

In lsb.params: DEFAULT_QUEUE=queueA queueB queueC queueD queueE

For the following jobs:

```
bsub -n 8 myjob
```

LSF automatically selects queueD to run myjob.

```
bsub -n 5 myjob
```

Job myjob fails because no default queue has the correct number of processors.

Maximum task limit

TASKLIMIT is specified in the default queue in lsb.queues as:

```
TASKLIMIT = 3
```

The maximum number of tasks that can be allocated for this queue is 3.

Example	Description
<code>bsub -n 2 myjob</code>	The job myjob has 2 tasks.
<code>bsub -n 4 myjob</code>	The job myjob is rejected from the queue because it requires more than the maximum number of tasks configured for the queue (3).
<code>bsub -n 2,3 myjob</code>	The job myjob runs on 2 or 3 processors.
<code>bsub -n 2,5 myjob</code>	The job myjob runs on 2 or 3 processors, depending on how many slots are currently available on the host.
<code>bsub myjob</code>	No default or minimum is configured, so the job myjob runs on 1 processor.

Minimum and maximum task limits

TASKLIMIT is specified in lsb.queues as:

```
TASKLIMIT = 3 8
```


The minimum number of tasks that can be allocated for this queue is 3 and the maximum number of tasks that can be allocated for this queue is 8.

Example	Description
<code>bsub -n 5 myjob</code>	The job <code>myjob</code> has 5 tasks.
<code>bsub -n 2 myjob</code>	The job <code>myjob</code> is rejected from the queue because the number of processors requested is less than the minimum number of processors configured for the queue (3).
<code>bsub -n 4,5 myjob</code>	The job <code>myjob</code> runs on 4 or 5 processors.
<code>bsub -n 2,6 myjob</code>	The job <code>myjob</code> runs on 3 to 6 processors.
<code>bsub -n 4,9 myjob</code>	The job <code>myjob</code> runs on 4 to 8 processors.
<code>bsub myjob</code>	The default number of processors is equal to the minimum number (3). The job <code>myjob</code> runs on 3 processors.

Minimum, default, and maximum task limits

TASKLIMIT is specified in `lsb.queues` as:

TASKLIMIT = 4 6 9

- Minimum number of tasks that can be allocated for this queue is 4
- Default number of tasks for the queue is 6
- Maximum number of tasks that can be allocated for this queue is 9

Example	Description
<code>bsub myjob</code>	Because a default number of tasks is configured, the job <code>myjob</code> runs on 6 processors.

Limit the number of allocated hosts

Use the `HOSTLIMIT_PER_JOB` parameter in `lsb.queues` to limit the number of hosts that a job can use. For example, if a user submits a parallel job using `bsub -n 1,4096 -R "span[ptile=1]"`, this job requests 4096 hosts from the cluster. If you specify a limit of 20 hosts per job, a user submitting a job requesting 4096 hosts will only be allowed to use 20 hosts.

Syntax

`HOSTLIMIT_PER_JOB = integer`

Specify the maximum number of hosts that a job can use. If the number of hosts requested for a parallel job exceeds this limit, the parallel job will pend.

How HOSTLIMIT_PER_JOB affects submission of parallel jobs

`span[ptile=value]` resource requirements

If a parallel job is submitted with the `span[ptile=processors_per_host]` resource requirement, the exact number of hosts requested is known (by dividing the number of processors by the processors per host). The job is rejected if the number of hosts requested exceeds the `HOSTLIMIT_PER_JOB` value. Other commands that specify a `span[ptile=processors_per_host]` resource requirement (such as **bmod**) are also subjected to this per-job host limit.

Compound resource requirements

If there is any part of the compound resource requirement that does not have a `ptile` specification, that part is considered to have a minimum of one host requested (before multiplying) when calculating the number of hosts requested.

For example:

- `2*{span[ptile=1]}+3*{-}` is considered to have a minimum of three hosts requested because the last part uses at least three hosts.
- `2*{-}+3*{-}+4*{-}` is considered to have a minimum of three hosts requested.

Alternative resource requirements

The smallest calculated number of hosts for all sets of resource requirements is used to compare to requested number of hosts with the per-job host limit. Any sets of resource requirements containing compound resource requirements, are calculated as compound resource requirements (that is, if there is any part of the compound resource requirement that does not have a `ptile` specification, that part is considered to have a minimum of one host requested, before multiplying, when calculating the number of hosts requested).

If the number of hosts requested in a parallel job is unknown during the submission stage, the per-job host limit does not apply and the job submission is accepted.

The per-job host limit is verified during resource allocation. If the per-job host limit is exceeded and the minimum number of requested hosts cannot be satisfied, the parallel job will pend.

This parameter does not stop the parallel job from resuming even if the job's host allocation exceeds the per-job host limit specified in this parameter.

If a parallel job is submitted under a range of the number of slots (`bsub -n "min, max"`), the per-job host limit applies to the minimum number of requested slots. That is, if the minimum number of requested slots is satisfied under the per-job host limit, the job submission is accepted.

Note: If you do not use a `ptile` specification in your resource requirements, LSF may have a false scheduling failure (that is, LSF may fail to find an allocation for a parallel job), even if a valid allocation exists. This occurs due to the computational complexity of finding an allocation with complex resource and limit relationships.

For example, hostA has two slots available, hostB and hostC have four slots available, and hostD has eight slots available, and `HOSTLIMIT_PER_JOB=2`. If you submit a job that requires ten slots and no `ptile` specification, the scheduler will determine that selecting hostA, hostB, and hostC will satisfy the requirements, but since this requires three hosts, the job will pend. This is a false scheduling failure because selecting hostA and hostD would satisfy this requirement.

To avoid false scheduling failure when `HOSTLIMIT_PER_JOB` is specified, submit jobs with the `ptile` resource requirement or add `order[slots]` to the resource requirements.

Reserve processors

About processor reservation

When parallel jobs have to compete with sequential jobs for job slots, the slots that become available are likely to be taken immediately by a sequential job. Parallel jobs need multiple job slots to be available before they can be dispatched. If the cluster is always busy, a large parallel job could be pending indefinitely. The more processors a parallel job requires, the worse the problem is.

Processor reservation solves this problem by reserving job slots as they become available, until there are enough reserved job slots to run the parallel job.

You might want to configure processor reservation if your cluster has a lot of sequential jobs that compete for job slots with parallel jobs.

How processor reservation works

Processor reservation is disabled by default.

If processor reservation is enabled, and a parallel job cannot be dispatched because there are not enough job slots to satisfy its minimum processor requirements, the job slots that are currently available is reserved and accumulated.

A reserved job slot is unavailable to any other job. To avoid deadlock situations in which the system reserves job slots for multiple parallel jobs and none of them can acquire sufficient resources to start, a parallel job gives up all its reserved job slots if it has not accumulated enough to start within a specified time. The reservation time starts from the time the first slot is reserved. When the reservation time expires, the job cannot reserve any slots for one scheduling cycle, but then the reservation process can begin again.

If you specify first execution host candidates at the job or queue level, LSF tries to reserve a job slot on the first execution host. If LSF cannot reserve a first execution host job slot, it does not reserve slots on any other hosts.

- [Configuring processor reservation](#)

Configuring processor reservation

Procedure

To enable processor reservation, set `SLOT_RESERVE` in `lsb.queues` and specify the reservation time. A job cannot hold any reserved slots after its reservation time expires.

`SLOT_RESERVE=MAX_RESERVE_TIME [n]`.

where *n* is an integer by which to multiply `MBD_SLEEP_TIME`. `MBD_SLEEP_TIME` is defined in `lsb.params`; the default value is 60 seconds.

For example:

```
Begin Queue
.
PJOB_LIMIT=1
SLOT_RESERVE = MAX_RESERVE_TIME[5]
.
End Queue
```

In this example, if `MBD_SLEEP_TIME` is 60 seconds, a job can reserve job slots for 5 minutes. If `MBD_SLEEP_TIME` is 30 seconds, a job can reserve job slots for $5 * 30 = 150$ seconds, or 2.5 minutes.

Reserve memory for pending parallel jobs

By default, the `rusage` string reserves resources for running jobs. Because resources are not reserved for pending jobs, some memory-intensive jobs could be pending indefinitely because smaller jobs take the resources immediately before the larger jobs can start running. The more memory a job requires, the worse the problem is.

Memory reservation for pending jobs solves this problem by reserving memory as it becomes available, until the total required memory specified on the `rusage` string is accumulated and the job can start. Use memory reservation for pending jobs if memory-intensive jobs often compete for memory with smaller jobs in your cluster.

Unlike slot reservation, which only applies to parallel jobs, memory reservation applies to both sequential and parallel jobs.

- [Configuring memory reservation for pending parallel jobs](#)
- [Enabling per-task memory reservation](#)

Configuring memory reservation for pending parallel jobs

About this task

You can reserve host memory for pending jobs.

Procedure

Set the RESOURCE_RESERVE parameter in a queue defined in lsb.queues.

The RESOURCE_RESERVE parameter overrides the SLOT_RESERVE parameter. If both RESOURCE_RESERVE and SLOT_RESERVE are defined in the same queue, job slot reservation and memory reservation are enabled and an error is displayed when the cluster is reconfigured. SLOT_RESERVE is ignored. Backfill on memory may still take place.

The following queue enables both memory reservation and backfill in the same queue:

```
Begin Queue
QUEUE_NAME = reservation_backfill
DESCRIPTION = For resource reservation and backfill
PRIORITY = 40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
BACKFILL = Y
End Queue
```

Enabling per-task memory reservation

About this task

By default, memory is reserved for parallel jobs on a per-host basis. For example, by default, the command:

```
bsub -n 4 -R "rusage[mem=500]" -q reservation myjob
```

requires the job to reserve 500 MB on each host where the job runs.

Procedure

To enable per-task memory reservation, define RESOURCE_RESERVE_PER_TASK=y in lsb.params. In this example, if per-task reservation is enabled, the job must reserve 500 MB of memory for each task ($4 * 500 = 2$ GB) on the host in order to run.

Backfill scheduling

By default, a reserved job slot cannot be used by another job. To make better use of resources and improve performance of LSF, you can configure backfill scheduling.

About backfill scheduling

Backfill scheduling allows other jobs to use the reserved job slots, as long as the other jobs do not delay the start of another job. Back filling, together with processor reservation, allows large parallel jobs to run while not under utilizing resources.

In a busy cluster, processor reservation helps to schedule large parallel jobs sooner. However, by default, reserved processors remain idle until the large job starts. This degrades the performance of LSF because the reserved resources are idle while jobs are waiting in the queue.

Backfill scheduling allows the reserved job slots to be used by small jobs that can run and finish before the large job starts. This improves the performance of LSF because it increases the utilization of resources.

How back filling works

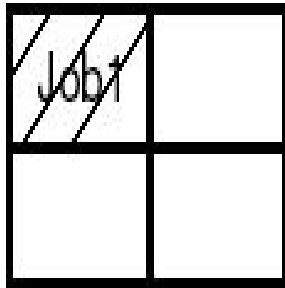
For backfill scheduling, LSF assumes that a job can run until its run limit expires. Backfill scheduling works most efficiently when all the jobs in the cluster have a run limit.

Since jobs with a shorter run limit have more chance of being scheduled as backfill jobs, users who specify appropriate run limits in a backfill queue is rewarded by improved turnaround time.

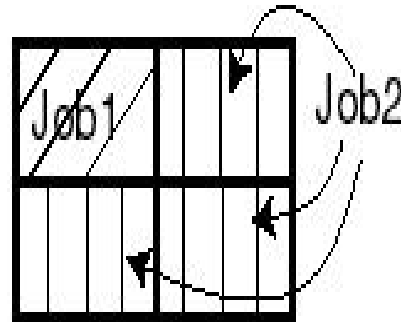
Once the big parallel job has reserved sufficient job slots, LSF calculates the start time of the big job, based on the run limits of the jobs currently running in the reserved slots. LSF cannot backfill if the big job is waiting for a job that has no run limit defined.

If LSF can backfill the idle job slots, only jobs with run limits that expire before the start time of the big job is allowed to use the reserved job slots. LSF cannot backfill with a job that has no run limit.

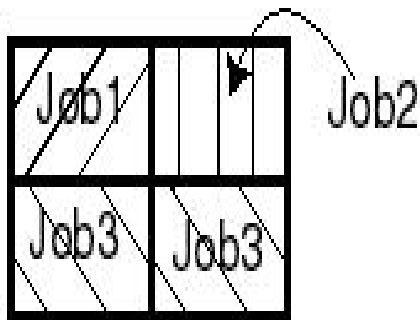
Example



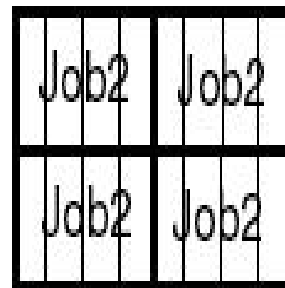
(a) Job1 started at 8:00 am.
Will finish at 10:00 am.



(b) Job2, submitted but can't start
since it needs 4 processors.
Remaining 3 reserved by Job2.



(c) At 8:30 am Job3 submitted.
Job3 backfills Job2.



(d) At 10:00 am, Job2 starts.

In this scenario, assume the cluster consists of a 4-CPU multiprocessor host.

1. A sequential job (`job1`) with a run limit of 2 hours is submitted and gets started at 8:00 am (figure a).
2. Shortly after, a parallel job (`job2`) requiring all 4 CPUs is submitted. It cannot start right away because `job1` is using one CPU, so it reserves the remaining 3 processors (figure b).
3. At 8:30 am, another parallel job (`job3`) is submitted requiring only two processors and with a run limit of 1 hour. Since `job2` cannot start until 10:00am (when `job1` finishes), its reserved processors can be backfilled by `job3` (figure c). Therefore `job3` can complete before `job2`'s start time, making use of the idle processors.
4. `Job3` finishes at 9:30am and `job1` at 10:00am, allowing `job2` to start shortly after 10:00am. In this example, if `job3`'s run limit was 2 hours, it would not be able to backfill `job2`'s reserved slots, and would have to run after `job2` finishes.

Limitations

- A job does not have an estimated start time immediately after mbatchd is reconfigured.

Back filling and job slot limits

A backfill job borrows a job slot that is already taken by another job. The backfill job does not run at the same time as the job that reserved the job slot first. Back filling can take place even if the job slot limits for a host or processor have been reached. Back filling cannot take place if the job slot limits for users or queues have been reached.

Job resize allocation requests

Pending job resize allocation requests are supported by backfill policies. However, the run time of pending resize request is equal to the remaining run time of the running re-sizable job. For example, if RUN LIMIT of a re-sizable job is 20 hours and 4 hours have already passed, the run time of pending resize request is 16 hours.

Configure backfill scheduling

Backfill scheduling is enabled at the queue level. Only jobs in a backfill queue can backfill reserved job slots. If the backfill queue also allows processor reservation, then back filling can occur among jobs within the same queue.

- [Configuring a backfill queue](#)
- [Enforce run limits](#)
Backfill scheduling requires all jobs to specify a duration. If you specify a run time limit using the command line **bsub -W** option or by defining the RUNLIMIT parameter in lsb.queues or lsb.applications, LSF uses that value as a hard limit and terminates jobs that exceed the specified duration. Alternatively, you can specify an estimated duration by defining the RUNTIME parameter in lsb.applications. LSF uses the RUNTIME estimate for scheduling purposes only, and does not terminate jobs that exceed the RUNTIME duration.
- [Use backfill on memory](#)
If BACKFILL is configured in a queue, and a run limit is specified with -W on **bsub** or with RUNLIMIT in the queue, backfill jobs can use the accumulated memory reserved by the other jobs, as long as the backfill job can finish before the predicted start time of the jobs with the reservation.
- [Use interruptible backfill](#)
Interruptible backfill scheduling can improve cluster utilization by allowing reserved job slots to be used by low priority small jobs that are terminated when the higher priority large jobs are about to start.
- [Submitting backfill jobs according to available slots](#)

Configuring a backfill queue

Procedure

1. To configure a backfill queue, define BACKFILL in lsb.queues.
2. Specify Y to enable back filling. To disable back filling, specify N or blank space.
BACKFILL=Y

Enforce run limits

Backfill scheduling requires all jobs to specify a duration. If you specify a run time limit using the command line **bsub -W** option or by defining the RUNLIMIT parameter in lsb.queues or lsb.applications, LSF uses that value as a hard limit and terminates jobs that exceed the specified duration. Alternatively, you can specify an estimated duration by defining the RUNTIME parameter in lsb.applications. LSF uses the RUNTIME estimate for scheduling purposes only, and does not terminate jobs that exceed the RUNTIME duration.

Use backfill on memory

If BACKFILL is configured in a queue, and a run limit is specified with -W on **bsub** or with RUNLIMIT in the queue, backfill jobs can use the accumulated memory reserved by the other jobs, as long as the backfill job can finish before the predicted start time of the jobs with the reservation.

Unlike slot reservation, which only applies to parallel jobs, backfill on memory applies to sequential and parallel jobs.

The following queue enables both memory reservation and backfill on memory in the same queue:

```
Begin Queue
QUEUE_NAME = reservation_backfill
DESCRIPTION = For resource reservation and backfill
PRIORITY = 40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
BACKFILL = Y
End Queue
```

Examples of memory reservation and backfill on memory

The following queues are defined in lsb.queues:

```
Begin Queue
QUEUE_NAME = reservation
DESCRIPTION = For resource reservation
PRIORITY=40
RESOURCE_RESERVE = MAX_RESERVE_TIME[20]
End Queue
```

```
Begin Queue
QUEUE_NAME = backfill
DESCRIPTION = For backfill scheduling
PRIORITY = 30
BACKFILL = y
End Queue
```

lsb.params

Per-slot memory reservation is enabled by RESOURCE_RESERVE_PER_TASK=y in lsb.params.

Assumptions

Assume one host in the cluster with 10 CPUs and 1 GB of free memory currently available.

Sequential jobs

Each of the following sequential jobs requires 400 MB of memory. The first three jobs run for 300 minutes.

Job 1:

```
bsub -W 300 -R "rusage[mem=400]" -q reservation myjob1
```

The job starts running, using 400M of memory and one job slot.

Job 2:

Submitting a second job with same requirements get the same result.

Job 3:

Submitting a third job with same requirements reserves one job slot, and reserve all free memory, if the amount of free memory is between 20 MB and 200 MB (some free memory may be used by the operating system or other software.)

Job 4:

```
bsub -W 400 -q backfill -R "rusage[mem=50]" myjob4
```

The job keeps pending, since memory is reserved by job 3 and it runs longer than job 1 and job 2.

Job 5:

```
bsub -W 100 -q backfill -R "rusage[mem=50]" myjob5
```

The job starts running. It uses one free slot and memory reserved by job 3. If the job does not finish in 100 minutes, it is killed by LSF automatically.

Job 6:

```
bsub -W 100 -q backfill -R "rusage[mem=300]" myjob6
```

The job keeps pending with no resource reservation because it cannot get enough memory from the memory reserved by job 3.

Job 7:

```
bsub -W 100 -q backfill myjob7
```

The job starts running. LSF assumes it does not require any memory and enough job slots are free.

Parallel jobs

Each process of a parallel job requires 100 MB memory, and each parallel job needs 4 cpus. The first two of the following parallel jobs run for 300 minutes.

Job 1:

```
bsub -W 300 -n 4 -R "rusage[mem=100]" -q reservation myJob1
```

The job starts running and use 4 slots and get 400MB memory.

Job 2:

Submitting a second job with same requirements gets the same result.

Job 3:

Submitting a third job with same requirements reserves 2 slots, and reserves all 200 MB of available memory, assuming no other applications are running outside of LSF.

Job 4:

```
bsub -W 400 -q backfill -R "rusage[mem=50]" myJob4
```

The job keeps pending since all available memory is already reserved by job 3. It runs longer than job 1 and job 2, so no backfill happens.

Job 5:

```
bsub -W 100 -q backfill -R "rusage[mem=50]" myJob5
```

This job starts running. It can backfill the slot and memory reserved by job 3. If the job does not finish in 100 minutes, it is killed by LSF automatically.

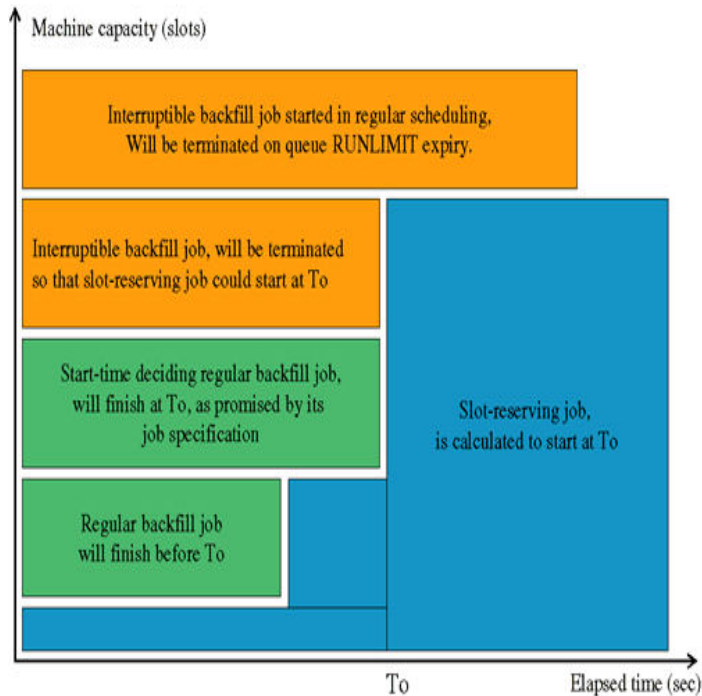
Use interruptible backfill

Interruptible backfill scheduling can improve cluster utilization by allowing reserved job slots to be used by low priority small jobs that are terminated when the higher priority large jobs are about to start.

An interruptible backfill job:

- Starts as a regular job and is killed when it exceeds the queue runtime limit, or
- Is started for backfill whenever there is a backfill time slice longer than the specified minimal time, and killed before the slot-reservation job is about to start. This applies to compute-intensive serial or single-node parallel jobs that can run a long time, yet be able to checkpoint or resume from an arbitrary computation point.

Resource allocation diagram



Job life cycle

1. Jobs are submitted to a queue configured for interruptible backfill. The job runtime requirement is ignored.
2. Job is scheduled as either regular job or backfill job.
3. The queue runtime limit is applied to the regularly scheduled job.
4. In backfill phase, the job is considered for run on any reserved resource, which duration is longer than the minimal time slice configured for the queue. The job runtime limit is set in such way, that the job releases the resource before it is needed by the slot reserving job.
5. The job runs in a regular manner. It is killed upon reaching its runtime limit, and re-queued for the next run. Re-queueing must be explicitly configured in the queue.

Assumptions and limitations

- The interruptible backfill job holds the slot-reserving job start until its calculated start time, in the same way as a regular backfill job. The interruptible backfill job is killed when its run limit expires.
- Killing other running jobs prematurely does not affect the calculated run limit of an interruptible backfill job. Slot-reserving jobs do not start sooner.
- While the queue is checked for the consistency of interruptible backfill, backfill and runtime specifications, the re-queue exit value clause is not verified, nor executed automatically. Configure re-queue exit values according to your site policies.
- When using the LSF multicluster capability, **bhist** does not display interruptible backfill information for remote clusters.
- A migrated job belonging to an interruptible backfill queue is migrated as if LSB_MIG2PEND is set.
- Interruptible backfill is disabled for re-sizable jobs. A re-sizable job can be submitted into interruptible backfill queue, but the job cannot be re-sized.
- [Configuring an interruptible backfill queue](#)

Configuring an interruptible backfill queue

Procedure

Configure `INTERRUPTIBLE_BACKFILL=seconds` in the lowest priority queue in the cluster. There can only be one interruptible backfill queue in the cluster.

Specify the minimum number of seconds for the job to be considered for backfilling. This minimal time slice depends on the specific job properties; it must be longer than at least one useful iteration of the job. Multiple queues may be created if a site has jobs of distinctively different classes.

For example:

```
Begin Queue
QUEUE_NAME      = background
# REQUEUE_EXIT_VALUES (set to whatever needed)
DESCRIPTION     = Interruptible Backfill queue
BACKFILL        = Y
INTERRUPTIBLE_BACKFILL = 1
RUNLIMIT        = 10
PRIORITY        = 1
End Queue
```

Interruptible backfill is disabled if `BACKFILL` and `RUNLIMIT` are not configured in the queue.

The value of `INTERRUPTIBLE_BACKFILL` is the minimal time slice in seconds for a job to be considered for backfill. The value depends on the specific job properties; it must be longer than at least one useful iteration of the job. Multiple queues may be created for different classes of jobs.

`BACKFILL` and `RUNLIMIT` must be configured in the queue.

`RUNLIMIT` corresponds to a maximum time slice for backfill, and should be configured so that the wait period for the new jobs submitted to the queue is acceptable to users. 10 minutes of runtime is a common value.

You should configure `REQUEUE_EXIT_VALUES` for the queue so that resubmission is automatic. In order to terminate completely, jobs must have specific exit values:

- If jobs are checkpoint-able, use their checkpoint exit value.
- If jobs periodically save data on their own, use the `SIGTERM` exit value.

Submitting backfill jobs according to available slots

Procedure

1. Use **bslots** to display job slots available for backfill jobs.
2. Submit a job to a backfill queue. Specify a runtime limit and the number of processors required that are within the availability shown by **bslots**.

Results

Submitting a job according to the backfill slot availability shown by **bslots** does not guarantee that the job is back filled successfully. The slots may not be available by the time job is actually scheduled, or the job cannot be dispatched because other resource requirements are not satisfied.

How deadline constraint scheduling works for parallel jobs

Deadline constraint scheduling is enabled by default.

If deadline constraint scheduling is enabled and a parallel job has a CPU limit but no run limit, LSF considers the number of processors when calculating how long the job takes.

LSF assumes that the minimum number of processors are used, and that they are all the same speed as the candidate host. If the job cannot finish under these conditions, LSF does not place the job.

The formula is:

$(\text{deadline time} - \text{current time}) > (\text{CPU limit on candidate host} / \text{minimum number of processors})$

Optimized preemption of parallel jobs

You can configure preemption for parallel jobs to reduce the number of jobs suspended in order to run a large parallel job.

When a high-priority parallel job preempts multiple low-priority parallel jobs, sometimes LSF preempts more low-priority jobs than are necessary to release sufficient job slots to start the high-priority job.

The PREEMPT_FOR parameter in lsb.params with the MINI_JOB keyword enables the optimized preemption of parallel jobs, so LSF preempts fewer of the low-priority parallel jobs.

Enabling the feature only improves the efficiency in cases where both preemptive and preempted jobs are parallel jobs.

How optimized preemption works

When you run many parallel jobs in your cluster, and parallel jobs preempt other parallel jobs, you can enable a feature to optimize the preemption mechanism among parallel jobs.

By default, LSF can over-preempt parallel jobs. When a high-priority parallel job preempts multiple low-priority parallel jobs, sometimes LSF preempts more low-priority jobs than are necessary to release sufficient job slots to start the high-priority job. The optimized preemption mechanism reduces the number of jobs that are preempted.

Enabling the feature only improves the efficiency in cases where both preemptive and preempted jobs are parallel jobs. Enabling or disabling this feature has no effect on the scheduling of jobs that require only a single processor.

- [Configuring optimized preemption](#)

Configuring optimized preemption

Procedure

Use the PREEMPT_FOR parameter in lsb.params and specify the keyword MINI_JOB to configure optimized preemption at the cluster level.

If the parameter is already set, the MINI_JOB keyword can be used along with other keywords; the other keywords do not enable or disable the optimized preemption mechanism.

Controlling CPU and memory affinity

IBM® Spectrum LSF can schedule jobs that are affinity aware. This allows jobs to take advantage of different levels of processing units (NUMA nodes, sockets, cores, and threads). Affinity scheduling is supported only on Linux and Power 7 and Power 8 hosts. Affinity scheduling is supported in LSF Standard Edition and LSF Advanced Edition. Affinity scheduling is not supported on LSF Express Edition.

An *affinity resource requirement* string specifies CPU or memory binding requirements for the tasks of jobs requiring topology-aware scheduling. An affinity[] resource requirement section controls CPU and memory resource allocations and specifies the distribution of *processor units* within a host according to the hardware topology information that LSF collects. The syntax supports basic affinity requirements for sequential jobs, as well as very complex task affinity requirements for parallel jobs.

affinity[] sections are accepted by **bsub -R**, and by **bmod -R** for non-running jobs, and can be specified in the RES_REQ parameter in lsb.applications and lsb.queues. Job-level affinity resource requirements take precedence over application-level requirements, which in turn override queue-level requirements.

You can use **bmod** to modify affinity resource requirements. After using **bmod** to modify memory resource usage of a running job with affinity requirements, **bhosts -l -aff** may show some inconsistencies between host-level memory and available memory in NUMA nodes. The modified memory resource for affinity binding requirements take effect on the affinity level only after re-queuing the running job and the next time **mbschd** schedules it; it takes effect immediately for host level memory allocation.

Enabling affinity scheduling

Enable CPU and memory affinity scheduling with the AFFINITY keyword in lsb.hosts.

Make sure that the affinity scheduling plugin **schmod_affinity** is defined in lsb.modules.

```
Begin PluginModule
SCH_PLUGIN          RB_PLUGIN          SCH_DISABLE_PHASES
schmod_default      ()                  ()
...
schmod_affinity      ()                  ()
End PluginModule
```

Limitations and known issues

CPU and memory affinity scheduling has the following limitations:

- Affinity resources cannot be released during preemption, so you should configure `mem` as a preemptable resource in lsb.params.
- When a job with affinity resources allocated has been stopped with **bstop**, the allocated affinity resources (thread, core, socket, NUMA node, NUMA memory) will not be released.
- Affinity scheduling is disabled for hosts with **cpuset** scheduling enabled, and on Cray Linux hosts.
- Affinity scheduling does not support NUMA level memory decay.
- When reservation is enabled, affinity reservation allocations appear as part of the allocated resources in **bhosts -aff**. Jobs that are submitted with a `membind=localprefer` binding policy may over-commit the memory of the NUMA node they are allocated to .

bhosts -aff output may occasionally show the total allocated memory on the NUMA nodes of a host as exceeding the maximum memory of the host, this is because the *reservations* that show in **bhosts -aff** over-commit the NUMA node. However, LSF will never allow the allocation of *running* jobs on a host to exceed the maximum memory of a host.

- When reservation is enabled, and an affinity job requests enough resources to consume an entire node in the host topology. (for example, enough cores to consume an entire socket), LSF will not reserve the socket for the job if there are any jobs running on its cores. In a situation when there are always smaller jobs running consuming cores, then larger jobs that require entire sockets will not be able to reserve resources. The workaround is to require that all jobs have estimated run times, and to use time-based reservation.
- [Submit jobs with affinity resource requirements](#)
Submit jobs for CPU and memory affinity scheduling by specifying an affinity[] section either in the **bsub -R** command, to a queue defined in the lsb.queues file or to an application profile with a RES_REQ parameter containing an affinity[] section.
- [Managing jobs with affinity resource requirements](#)
You can view resources allocated for jobs and tasks with CPU and memory affinity resource requirements with the `-l -aff` option of **bjobs**, **bhist**, and **bacct**. Use **bhosts -aff** to view host resources allocated for affinity jobs.
- [Affinity preemption](#)
To enable affinity preemption, set the `PREEMPT_JOBTYPE = AFFINITY` parameter in the Parameters section of the lsb.params file. By default, affinity resources are not preemptable.

- [Affinity binding based on Linux cgroup cpuset subsystem](#)
LSF can enforce CPU binding on systems that support the Linux cgroup cpuset subsystem. When CPU affinity binding through Linux cgroups is enabled, LSF will create a cpuset to contain job processes if the job has affinity resource requirements, so that the job processes cannot escape from the allocated CPUs. Each affinity job cpuset includes only the CPU and memory nodes that LSF distributes. Linux cgroup cpusets are only created for affinity jobs.
- [Portable hardware locality](#)
Portable Hardware Locality (**hwloc**) is an open source software package that is distributed under BSD license. It provides a portable abstraction (across OS, versions, architectures, and so on) of the hierarchical topology of modern architectures, including NUMA memory nodes, socket, shared caches, cores, and simultaneous multithreading (SMT). **hwloc** is integrated into LSF to detect hardware information, and can support most of the platforms that LSF supports.

Submit jobs with affinity resource requirements

Submit jobs for CPU and memory affinity scheduling by specifying an affinity[] section either in the **bsub -R** command, to a queue defined in the lsb.queues file or to an application profile with a RES_REQ parameter containing an affinity[] section.

Tip: Starting in Fix Pack 14, you can configure the [LSF_CGROUP_CORE_AUTO_CREATE](#) parameter set to `y` to enable LSF to automatically create Linux cgroups for a job, without the need to specify affinity[] requirements. With this setting, LSF automatically adds the "**affinity[core(1)]**" resource requirement string to the **bsub -R** command whenever jobs are submitted.

The affinity[] resource requirement string controls job slot and processor unit allocation and distribution within a host.

See [Affinity string](#) for detailed syntax of the affinity[] resource requirement string.

If the JOB_INCLUDE_POSTPROC=Y parameter is set in the lsb.params file, or the LSB_JOB_INCLUDE_POSTPROC=Y environment variable is set in the job environment, LSF does not release affinity resources until post-execution processing has finished, since slots are still occupied by the job during post-execution processing. For interactive jobs, the interactive job will be finished before the post-execution completes.

Examples: processor unit allocation requests

The following examples illustrate affinity jobs that request specific processor unit allocations and task distributions.

The following job asks for six slots and runs within single host. Each slot maps to one core. LSF tries to pack six cores as close as possible on single NUMA or socket. If the task distribution cannot be satisfied, the job can not be started.

```
bsub -n 6 -R "span[hosts=1] affinity[core(1):distribute=pack]" myjob
```

The following job asks for six slots and runs within single host. Each slot maps to one core, but in this case it must be packed into a single socket, otherwise, the job remains pending:

```
bsub -n 6 -R "span[hosts=1] affinity[core(1):distribute=pack(socket=1)]" myjob
```

The following Job asks for two slots on a single host. Each slot maps to two cores. Two cores for a single slot (task) must come from the same socket; however, the other two cores for second slot (task) must be on different socket:

```
bsub -n 2 -R "span[hosts=1] affinity[core(2, same=socket, exclusive=(socket, injob))]" myjob
```

The following job specifies that each task in the job requires two cores from the same socket. The allocated socket will be marked exclusive for all other jobs. The task will be CPU bound to socket level. LSF attempts to distribute the tasks of the job so that they are balanced across all cores:

```
bsub -n 4 -R "affinity[core(2, same=socket, exclusive=(socket, alljobs))]:  
cpubind=socket:distribute=balance]" myjob
```

Examples: CPU and memory binding requests

You can submit affinity jobs with CPU various binding and memory binding options. The following examples illustrate this.

In the following job, both tasks require five cores in the same NUMA node and binds the tasks on the NUMA node with memory mandatory binding:

```
bsub -n 2 -R "affinity[core(5,same=numa):cpubind=numa:membind=localonly]" myjob
```

The following job binds a multithread job on a single NUMA node:

```
bsub -n 2 -R "affinity[core(3,same=numa):cpubind=numa:membind=localprefer]" myjob
```

The following job distributes tasks across sockets. Each task needs two cores from the same socket and binds each task at the socket level. The allocated socket is exclusive, so no other tasks can use it:

```
bsub -n 2 -R "affinity[core(2,same=socket,exclusive=(socket,injob|alljobs)):cpubind=socket]" myjob
```

The following job packs job tasks in one NUMA node:

```
bsub -n 2 -R "affinity[core(1,exclusive=(socket,injob)):distribute=pack(numa= 1)]" myjob
```

Each task needs one core and no other tasks from the same job will allocate CPUs from the same socket. LSF attempts to pack all tasks in the same job to one NUMA node.

- [Submit jobs with affinity resource requirements on IBM POWER8 systems](#)
Use the **esub.p8aff** external submission (**esub**) script to automatically generate the optimal job-level affinity requirements for job submissions on IBM POWER8 (ppc64le) systems.

Job execution environment for affinity jobs

LSF sets several environment variables in the execution environment of each job and task. These are designed to integrate and work with IBM Parallel Environment, and IBM Spectrum LSF MPI. However, these environment variables are available to all affinity jobs and could potentially be used by other applications. Because LSF provides the variables expected by both IBM Parallel Environment and LSF MPI, there is some redundancy: environment variables prefixed by **RM_** are implemented for compatibility with IBM Parallel Environment, although LSF MPI uses them as well, while those prefixed with **LSB_** are only used by LSF MPI. The two types of variable provide similar information, but in different formats.

The following variables are set in the job execution environment:

- **LSB_BIND_CPU_LIST**
- **LSB_BIND_MEM_LIST**
- **LSB_BIND_MEM_POLICY**
- **RM_CPUTASK_n**
- **RM_MEM_AFFINITY**
- **OMP_NUM_THREADS**

Application integration

For *single-host applications* the application itself does not need to do anything, and only the **OMP_NUM_THREADS** variable is relevant.

For the first execution host of a *multi-host parallel application* LSF MPI running under LSF will select CPU resources for each task, start up the LSF MPI agent (**mpid**) to bind **mpid** to all allocated CPUs and memory policies. Corresponding environment variables are set including **RM_CPUTASK_n**. LSF MPI reads **RM_CPUTASK_n** on each host, and does the task-level binding. LSF MPI follows the **RM_CPUTASK_n** setting and binds each task to the selected CPU list per task. This is the default behavior when LSF MPI runs under LSF.

To support *IBM Parallel Operating Environment* jobs, LSF starts the PMD program, binds the PMD process to the allocated CPUs and memory nodes on the host, and sets **RM_CPUTASK_n**, **RM_MEM_AFFINITY**, and **OMP_NUM_THREADS**. The IBM Parallel Operating Environment will then do the binding for individual tasks.

OpenMPI provides a rank file as the interface for users to define CPU binding information per task. The rank file includes MPI rank, host, and CPU binding allocations per rank. LSF provides a simple script to generate an OpenMPI rank file based on **LSB_AFFINITY_HOSTFILE**. The following is an example of an OpenMPI rankfile corresponding to the affinity hostfile in the description of **LSB_AFFINITY_HOSTFILE**:

```

Rank 0=Host1 slot=0,1,2,3
Rank 1=Host1 slot=4,5,6,7
Rank 2=Host2 slot=0,1,2,3
Rank 3=Host2 slot=4,5,6,7
Rank 4=Host3 slot=0,1,2,3
Rank 5=Host4 slot=0,1,2,3

```

The script (openmpi_rankfile.sh) is located in \$LSF_BINDIR. Use the DJOB_ENV_SCRIPT parameter in an application profile in lsb.applications to configure the path to the script.

For *distributed applications* that use **blaunch** directly to launch tasks or agent per slot (not per host) by default, LSF binds the task to all allocated CPUs and memory nodes on the host. That is, the CPU and memory node lists are generated at the host level. Certain distributed application may need to generate the binding lists on a task-by-task basis. This behaviour is configurable in either job submission environment or an application profile as an environment variable named LSB_DJOB_TASK_BIND=Y | N. N is the default. When this environment variable is set, the binding list will be generated on a task per task basis.

Examples

The following examples assume that the cluster comprises only hosts with the following topology:

```

Host[64.0G] HostN
  NUMA[0: 0M / 32.0G]      NUMA[1: 0M / 32.0G]
    Socket0                Socket0
      core0(0 22)           core0(1 23)
      core1(2 20)           core1(3 21)
      core2(4 18)           core2(5 19)
      core3(6 16)           core3(7 17)
      core4(8 14)           core4(9 15)
      core5(10 12)          core5(11 13)
    Socket1                Socket1
      core0(24 46)          core0(25 47)
      core1(26 44)          core1(27 45)
      core2(28 42)          core2(29 43)
      core3(30 40)          core3(31 41)
      core4(32 38)          core4(33 39)
      core5(34 36)          core5(35 37)

```

Each host has 64 GB of memory split over two NUMA nodes, each node containing two processor sockets with 6 cores each, and each core having two threads. Each of the following examples consists of the following:

- A **bsub** command line with an affinity requirement
- An allocation for the resulting job displayed as in **bjobs**
- The same allocation displayed as in **bhosts**
- The values of the job environment variables above once the job is dispatched

The examples cover some of the more common examples: serial and parallel jobs with simple CPU and memory requirements, as well as the effect of the exclusive clause of the affinity resource requirement string.

1. `bsub -R "affinity[core(1)]"` is a serial job asking for a single core.

The allocation shown in **bjobs**:

```

...
          CPU BINDING
          -----
HOST      TYPE  LEVEL  EXCL  IDS
Host1     core   -      -    /0/0/0
...
          MEMORY BINDING
          -----
POL  NUMA  SIZE
-    -    -

```

In **bhosts** (assuming no other jobs are on the host):

```

...
Host[64.0G] Host1
  NUMA[0: 0M / 32.0G]      NUMA[1: 0M / 32.0G]
    Socket0                Socket0
      core0(*0 *22)         core0(1 23)
      core1(2 20)           core1(3 21)
      core2(4 18)           core2(5 19)
      core3(6 16)           core3(7 17)

```

```

        core4 (8 14)
        core5 (10 12)
Socket1
    core0 (24 46)
    core1 (26 44)
    core2 (28 42)
    core3 (30 40)
    core4 (32 38)
    core5 (34 36)
...
        core4 (9 15)
        core5 (11 13)
Socket1
    core0 (25 47)
    core1 (27 45)
    core2 (29 43)
    core3 (31 41)
    core4 (33 39)
    core5 (35 37)

```

Contents of affinity host file:

```
Host1 0,22
```

Job environment variables:

```
LSB_BIND_CPU_LIST=0,22
RM_CPUTASK1=0,22
```

2. `bsub -R "affinity[socket(1)]"` is a serial job asking for an entire socket.

The allocation shown in **bjobs**:

```

...
          CPU BINDING
          -----
HOST      TYPE  LEVEL  EXCL  IDS
Host1     socket -      -    /0/0
          MEMORY BINDING
          -----
          POL  NUMA SIZE
          ---  --- ---
Host1     -    -    -
...

```

In **bhosts** (assuming no other jobs are on the host):

```

...
Host[64.0G] Host1
  NUMA[0: 0M / 32.0G]  NUMA[1: 0M / 32.0G]
    Socket0
      core0 (*0 *22)
      core1 (*2 *20)
      core2 (*4 *18)
      core3 (*6 *16)
      core4 (*8 *14)
      core5 (*10 *12)
    Socket1
      core0 (24 46)
      core1 (26 44)
      core2 (28 42)
      core3 (30 40)
      core4 (32 38)
      core5 (34 36)
    Socket0
      core0 (1 23)
      core1 (3 21)
      core2 (5 19)
      core3 (7 17)
      core4 (9 15)
      core5 (11 13)
    Socket1
      core0 (25 47)
      core1 (27 45)
      core2 (29 43)
      core3 (31 41)
      core4 (33 39)
      core5 (35 37)
...

```

Contents of affinity host file:

```
Host1 0,2,4,6,8,10,12,14,16,18,20,22
```

Job environment variables:

```
LSB_BIND_CPU_LIST=0,2,4,6,8,10,12,14,16,18,20,22
RM_CPUTASK1=0,2,4,6,8,10,12,14,16,18,20,22
```

3. `bsub -R "affinity[core(4):membind=localonly] rusage[mem=2048]"` is a multi-threaded single-task job requiring 4 cores and 2 GB of memory.

The allocation shown in **bjobs**:

```

...
          CPU BINDING
          -----
HOST      TYPE  LEVEL  EXCL  IDS
Host1     core  -      -    /0/0/0
          /0/0/1
          MEMORY BINDING
          -----
          POL  NUMA SIZE
          ---  --- ---
Host1     local 0    2.0GB

```



```

...
/0/0/2
/0/0/3

```

In **bhosts** (assuming no other jobs are on the host):

```

...
Host[64.0G] Host1
  NUMA[0: 2.0G / 32.0G]      NUMA[1: 0M / 32.0G]
    Socket0
      core0(*0 *22)
      core1(*2 *20)
      core2(*4 *18)
      core3(*6 *16)
      core4(8 14)
      core5(10 12)
    Socket1
      core0(24 46)
      core1(26 44)
      core2(28 42)
      core3(30 40)
      core4(32 38)
      core5(34 36)
    Socket0
      core0(1 23)
      core1(3 21)
      core2(5 19)
      core3(7 17)
      core4(9 15)
      core5(11 13)
    Socket1
      core0(25 47)
      core1(27 45)
      core2(29 43)
      core3(31 41)
      core4(33 39)
      core5(35 37)

```

Contents of affinity host file:

```
Host1 0,2,4,6,16,18,20,22 0 1
```

Job environment variables:

```

LSB_BIND_CPU_LIST=0,2,4,6,16,18,20,22
LSB_BIND_MEM_LIST=0
LSB_BIND_MEM_POLICY=localonly
RM_MEM_AFFINITY=yes
RM_CPUTASK1=0,2,4,6,16,18,20,22
OMP_NUM_THREADS=4

```

Note: OMP_NUM_THREADS is now present because the only task in the job asked for 4 cores.

4. `bsub -n 2 -R "affinity[core(2)] span[hosts=1]"` is a multi-threaded parallel job asking for 2 tasks with 2 cores each running on the same host.

The allocation shown in **bjobs**:

```

...
          CPU BINDING
          -----
HOST      TYPE  LEVEL  EXCL  IDS
Host1     core  -      -    /0/0/0
          /0/0/1
Host1     core  -      -    /0/0/2
          /0/0/3
          MEMORY BINDING
          -----
          POL  NUMA  SIZE
          ---  ---  ---

```

In **bhosts** (assuming no other jobs are on the host):

```

...
Host[64.0G] Host1
  NUMA[0: 0M / 32.0G]      NUMA[1: 0M / 32.0G]
    Socket0
      core0(*0 *22)
      core1(*2 *20)
      core2(*4 *18)
      core3(*6 *16)
      core4(8 14)
      core5(10 12)
    Socket1
      core0(24 46)
      core1(26 44)
      core2(28 42)
      core3(30 40)
      core4(32 38)
    Socket0
      core0(1 23)
      core1(3 21)
      core2(5 19)
      core3(7 17)
      core4(9 15)
      core5(11 13)
    Socket1
      core0(25 47)
      core1(27 45)
      core2(29 43)
      core3(31 41)
      core4(33 39)

```

```
... core5 (34 36) core5 (35 37)
...
```

Contents of affinity host file:

```
Host1 0,2,4,6
Host1 16,18,20,22
```

Job environment variables set for *each of the two tasks*:

```
LSB_BIND_CPU_LIST=0,2,4,6,16,18,20,22
RM_CPUS_TASK1=0,2,4,6
RM_CPUS_TASK2=16,18,20,22
OMP_NUM_THREADS=2
```

Note: Each task sees RM_CPU_TASK1 and RM_CPU_TASK2 and that LSB_BIND_CPU_LIST is the combined list of all the CPUs allocated to the job on this host.

If you run the job through the **blaunch** command and set the LSB_DJOB_TASK_BIND parameter, then everything is the same except that the job environment variables of the two tasks are different for each task:

- Task 1:

```
LSB_BIND_CPU_LIST=0,2,20,22
RM_CPUS_TASK1=0,2,20,22
OMP_NUM_THREADS=2
```

- Task 2:

```
LSB_BIND_CPU_LIST=4,6,16,18
RM_CPUS_TASK1=4,6,16,18
OMP_NUM_THREADS=2
```

5. `bsub -n 2 -R "affinity[core(2)] span[ptile=1]"` is a multi-threaded parallel job asking for a 2 tasks with 2 cores each running on a different host. This is almost identical to the previous example except that the allocation is across two hosts.

The allocation shown in **bjobs**:

```
...

```

HOST	CPU BINDING				MEMORY BINDING		
	TYPE	LEVEL	EXCL	IDS	POL	NUMA	SIZE
Host1	core	-	-	/0/0/0	-	-	-
				/0/0/1			
Host2	core	-	-	/0/0/0	-	-	-
				/0/0/1			

```
...
```

In **bhosts** (assuming no other jobs are on the host), each of Host1 and Host2 would be allocated as:

```
...
Host[64.0G] Host{1,2}
  NUMA[0: 0M / 32.0G]      NUMA[1: 0M / 32.0G]
    Socket0                Socket0
      core0 (*0 *22)        core0 (1 23)
      core1 (*2 *20)        core1 (3 21)
      core2 (4 18)          core2 (5 19)
      core3 (6 16)          core3 (7 17)
      core4 (8 14)          core4 (9 15)
      core5 (10 12)         core5 (11 13)
    Socket1                Socket1
      core0 (24 46)          core0 (25 47)
      core1 (26 44)          core1 (27 45)
      core2 (28 42)          core2 (29 43)
      core3 (30 40)          core3 (31 41)
      core4 (32 38)          core4 (33 39)
      core5 (34 36)          core5 (35 37)
...
```

Contents of affinity host file:

```
Host1 0,2,20,22
Host2 0,2,20,22
```

Job environment variables set for each of the two tasks:

```
LSB_BIND_CPU_LIST=0,2,20,22
RM_CPUTASK1=0,2,20,22
OMP_NUM_THREADS=2
```

Note: Each task only sees RM_CPU_TASK1. This is the same as LSB_BIND_CPU_LIST because only one task is running on each host. Setting DJOB_TASK_BIND=Y would have no effect in this case.

6. `bsub -R "affinity[core(1,exclusive=(socket,alljobs))]"` is an example of a single threaded serial job asking for a core that it would like to have exclusive use of a socket across all jobs. Compare this with examples (1) and (2) above of a jobs simply asking for a core or socket.

The allocation shown in **bjobs** is the same as the job asking for a core except for the EXCL column:

```
...
```

HOST	CPU BINDING				MEMORY BINDING		
	TYPE	LEVEL	EXCL	IDS	POL	NUMA	SIZE
Host1	core	-	socket	/0/0/0	-	-	-

```
...
```

In **bhosts**, however, the allocation is the same as the job asking for a socket because it needs to reserve it all:

```
...
Host[64.0G] Host1
  NUMA[0: 0M / 32.0G]      NUMA[1: 0M / 32.0G]
    Socket0                Socket0
      core0 (*0 *22)        core0 (1 23)
      core1 (*2 *20)        core1 (3 21)
      core2 (*4 *18)        core2 (5 19)
      core3 (*6 *16)        core3 (7 17)
      core4 (*8 *14)        core4 (9 15)
      core5 (*10 *12)       core5 (11 13)
    Socket1                Socket1
      core0 (24 46)         core0 (25 47)
      core1 (26 44)         core1 (27 45)
      core2 (28 42)         core2 (29 43)
      core3 (30 40)         core3 (31 41)
      core4 (32 38)         core4 (33 39)
      core5 (34 36)         core5 (35 37)
...
```

The affinity hosts file, however, shows that the job is only bound to the allocated core when it runs

```
Host1 0,22
```

This is also reflected in the job environment:

```
LSB_BIND_CPU_LIST=0,22
RM_CPUTASK1=0,22
```

From the point of view of what is available to other jobs (that is, the allocation counted against the host), the job has used an entire socket. However in all other aspects the job is only binding to a single core.

7. `bsub -R "affinity[core(1):cpubind=socket]"` asks for a core but asks for the binding to be done at the socket level. Contrast this with the previous case where the core wanted exclusive use of the socket.

Again, the **bjobs** allocation is the same as example (1), but this time the LEVEL column is different:

```
...
```

HOST	CPU BINDING				MEMORY BINDING		
	TYPE	LEVEL	EXCL	IDS	POL	NUMA	SIZE
Host1	core	socket	-	/0/0/0	-	-	-

```
...
```

In **bhosts**, the job just takes up a single core, rather than the whole socket like the exclusive job:

```

...
Host[64.0G] Host1
  NUMA[0: 0M / 32.0G]      NUMA[1: 0M / 32.0G]
    Socket0                Socket0
      core0(*0 *22)         core0(1 23)
      core1(2 20)          core1(3 21)
      core2(4 18)          core2(5 19)
      core3(6 16)          core3(7 17)
      core4(8 14)          core4(9 15)
      core5(10 12)         core5(11 13)
    Socket1                Socket1
      core0(24 46)          core0(25 47)
      core1(26 44)          core1(27 45)
      core2(28 42)          core2(29 43)
      core3(30 40)          core3(31 41)
      core4(32 38)          core4(33 39)
      core5(34 36)          core5(35 37)
...

```

The view from the execution side though is quite different: from here the list of CPUs that populate the job's binding list on the host is the entire socket.

Contents of the affinity host file:

```
Host1 0,2,4,6,8,10,12,14,16,18,20,22
```

Job environment:

```
LSB_BIND_CPU_LIST=0,2,4,6,8,10,12,14,16,18,20,22
RM_CPULIST=0,2,4,6,8,10,12,14,16,18,20,22
```

Compared to the previous example, from the point of view of what is available to other jobs (that is, the allocation counted against the host), the job has used a single core. However in terms of the binding list, the job process will be free to use any CPU in the socket while it is running.

Submit jobs with affinity resource requirements on IBM POWER8 systems

Use the **esub.p8aff** external submission (**esub**) script to automatically generate the optimal job-level affinity requirements for job submissions on IBM POWER8 (ppc64le) systems.

To simplify job submission with affinity requirements for IBM POWER8 (ppc64le) systems, LSF includes the **esub.p8aff** external submission (**esub**) script to automatically generate affinity requirements based on the input requirements for the jobs. For the generated affinity requirements, LSF attempts to reduce the risks of CPU bottlenecks for the CPI allocation in MPI task and OpenMP thread levels.

Requirements

To submit affinity jobs on IBM POWER8 (ppc64le) systems, these hosts must be able to retrieve and change SMT configurations. Ensure that the following features are enabled in the LSF cluster:

- Configure the initial SMT mode on all ppc64le execution hosts to the maximum available SMT number.
- Install the **ppc64_cpu** command on all ppc64le execution hosts.
- Enable at least one of the Linux cgroup features are enabled in LSF to support customized SMT. That is, ensure that at least one of the following parameters are enabled in the lsf.conf file:
 - LSB_RESOURCE_ENFORCE
 - LSF_LINUX_CGROUP_ACCT
 - LSF_PROCESS_TRACKING

How to use the esub.p8aff script

Use the **bsub -a** command to run the **esub.p8aff** script:

```
bsub -a "p8aff (num_threads_per_task, SMT, cpus_per_core, distribution_policy)"
```

num_threads_per_task

The number of OpenMP threads per MPI task.

LSF uses this number to calculate the list of logical CPUs that are bound to each OpenMP thread. LSF sets the `OMP_NUM_THREADS` environment variable to this value for SMT jobs.

SMT

Optional. The required per-job SMT mode on the execution hosts.

Use this argument to specify the expected SMT mode that is enabled on the execution hosts. LSF sets the `LSB_JOB_SMT_MODE` environment variable to this value for SMT jobs.

LSF automatically adds the exclusive job option (-x) to the job submission to ensure that the execution hosts are not allocated for other jobs.

If you do not specify this argument, SMT mode is not used on the execution hosts.

cpus_per_core

Optional. The number of logical CPUs used per core for each MPI task.

LSF uses this argument to determine how many cores are spanned for each MPI task.

For example, if this argument is specified as 2 (two logical CPUs per core), LSF allocates the MPI task on three cores if the task requires a total of six threads.

distribution_policy

Optional. The required task distribution policy for the job.

This argument specifies the expected task distribution policy of the MPI tasks. Valid values are `balance`, `pack`, and `any`, which are the same as the corresponding values for the `distribute` keyword in the LSF affinity[] resource requirement string.

If you are not specifying the optional arguments, leave those arguments blank:

- Use `"-a p8aff(10,8,,pack)"` if the `cpus_per_core` argument is not specified.
- Use `"-a p8aff(10,,,pack)"` if the `SMT` and `cpus_per_core` arguments are not specified.
- Use `"-a p8aff(10,8,2)"` or `"-a p8aff(10,8,2,)"` if the `distribution_policy` argument is not specified.
- Use `"-a p8aff(10,8)"` or `"-a p8aff(10,8,,)"` if the `cpus_per_core` and `distribution_policy` arguments are not specified.

Submitting jobs without job-level affinity requirements

If the user specifies at least the first argument (`num_threads_per_task`) when submitting jobs without job-level affinity requirements (that is, without specifying `-R "affinity[]"`), the **esub.p8aff** script automatically generates the job-level affinity requirements for the job.

For example, if you specify the following arguments, the **esub.p8aff** script informs LSF that each MPI task in the job that uses 10 OpenMP threads and the execution hosts must be configured with 4 SMTs:

```
bsub -n 2 -a "p8aff(10,4,1,balance)" myjob
```

The **esub.p8aff** script generates the following job-level affinity requirement based on these arguments:

```
-R "affinity[thread(1, exclusive=(core,intask))*10:cpubind=thread:distribute=balance]"
```

LSF generates the affinity requirements to ensure that it allocates one logical CPU on each physical core for each MPI task. LSF attempts to distribute the job tasks equally across all processor units on the allocated hosts.

Submitting jobs with job-level affinity requirements

If you specify the `num_threads_per_task` or `SMT` arguments when submitting jobs with job-level affinity requirements (that is, by specifying `-R "affinity[]"`), **esub.p8aff** sets the `OMP_NUM_THREADS` or `LSB_JOB_SMT_MODE` environment variables

for the job. LSF configures OpenMP thread affinity and the requested SMT mode on the execution hosts according to these environment variables.

Note: If you specify the `num_threads_per_task`, `SMT`, or both arguments while using job-level affinity requirements, **esub.p8aff** ignores the `cpus_per_core` and `distribution_policy` arguments.

For example, if you specify the following arguments while using the job-level affinity requirements, the **esub.p8aff** script informs LSF that each MPI task in the job that uses 10 OpenMP threads and the execution hosts must be configured with 4 SMTs:

```
bsub -n 2 -a "p8aff(10,4)" -R "affinity[thread(1, exclusive=(core,intask))*10:cpubind:distribute=balance]" myjob
```

This particular example shows that you specified your own affinity requirements without relying on **esub.p8aff** to automatically generate one for you.

Per-job SMT mode configurations

The LSF administrator must set the initial SMT mode to the maximum available SMT number on all execution hosts before LSF schedules jobs. LSF uses this number as the default SMT mode if there are no jobs running on the hosts. If the initial SMT mode is not the maximum allowed SMT number, LSF uses this mode as the default SMT mode on the hosts, which means that LSF assumes that this smaller value is the maximum number of SMT resources available for job scheduling.

LSF uses the **ppc64_cpu** command before starting tasks on each host to configure the SMT mode according to the `LSB_JOB_SMT_MODE` environment variable. After the job finishes, LSF sets the SMT mode on each host back to the default SMT number for that host.

Integration with OpenMP thread affinity

LSF supports OpenMP thread affinity in the **blaunch** distributed application framework. LSF MPI distributions must integrate with LSF to enable the OpenMP thread affinity.

When the `OMP_NUM_THREADS` environment variable is set for the job, LSF automatically sets the following environment variables that are related to the OpenMP thread affinity that is based on the allocated logical CPUs for the task:

- `OMP_PROC_BIND`: LSF sets this environment variable to `"TRUE"` if the `OMP_NUM_THREAD` environment variable is available.
- `OMP_PLACES`: LSF calculates the list of logical CPUs to bind each OpenMP thread to a logical CPU.

Each OpenMP thread is bound to an individual logical CPU to avoid the thread from switching overheads. LSF evenly binds each OpenMP thread to one logical CPU from the list of allocated logical CPUs for the current MPI task.

LSF applies the OpenMP thread affinity in the following manner:

- If the number of allocated logical CPUs is equal to the number of OpenMP threads per task (that is, the `OMP_NUM_THREADS` environment variable value), each OpenMP thread is bound to a separate logical CPU. For example, if the logical CPUs that are allocated to an MPI task with two OpenMP threads are logical CPUs 0 and 8, the `OMP_PLACES` environment variable is set to `"{0},{8}"`.
- If the number of allocated logical CPUs is larger than the value of the `OMP_NUM_THREADS` environment variable, the list of logical CPUs is placed into multiple groups based on the number specified by the `OMP_NUM_THREADS` environment variable. The first logical CPU in each group is bound to each OpenMP thread. All remaining CPUs that do not divide evenly into the number of OpenMP threads are added to the last group. For example, if the logical CPUs that are allocated to an MPI task are 0, 1, 2, 3, and 4, and there are two OpenMP threads that are started by this task, there would be two CPU groups: {0,1} and {2,3,4}. Then, `OMP_PLACES` environment variable is set to `"{0},{2}"`.
- If the number of allocated logical CPUs is smaller than the value of the `OMP_NUM_THREADS` environment variable, all the threads are bound to all the logical CPUs. For example, if a job's OpenMP thread number is 5 and the affinity requirement is `"thread(2)"`, LSF evenly binds the allocated CPUs to each OpenMP thread by using the round-robin manner. The `OMP_PLACES` environment variable is set to `"{0},{1},{0},{1},{0}"` if LSF allocates logical CPUs 0 and 1 to the task.

Managing jobs with affinity resource requirements

You can view resources allocated for jobs and tasks with CPU and memory affinity resource requirements with the `-l -aff` option of **bjobs**, **bhist**, and **bacct**. Use **bhosts -aff** to view host resources allocated for affinity jobs.

Viewing job resources for affinity jobs (-aff)

The `-aff` option displays information about jobs with CPU and memory affinity resource requirement for each task in the job. A table headed **AFFINITY** shows detailed memory and CPU binding information for each task in the job, one line for each allocated processor unit.

Use only with the `-l` option of **bjobs**, **bhist**, and **bacct**.

Use **bjobs -l -aff** to display information about CPU and memory affinity resource requirements for job tasks. A table with the heading **AFFINITY** is displayed containing the detailed affinity information for each task, one line for each allocated processor unit. CPU binding and memory binding information are shown in separate columns in the display.

For example the following job starts 6 tasks with the following affinity resource requirements:

```
bsub -n 6 -R"span[hosts=1] rusage[mem=100]affinity[core(1,same=socket,exclusive=(socket,injob)) :cpubind=socket:membind=localonly:distribute=pack]" myjob
Job <6> is submitted to default queue <normal>.
```

```
bjobs -l -aff 61
```

```
Job <61>, User <user1>, Project <default>, Status <RUN>, Queue <normal>, Command <myjob1>
Thu Feb 14 14:13:46: Submitted from host <hostA>, CWD <${HOME}>, 6 Processors Requested, Requested Resources <span[hosts=1] rusage[mem=100]affinity[core(1,same=socket,exclusive=(socket,injob)):cpubind=socket:membind=localonly:distribute=pack]>;
Thu Feb 14 14:15:07: Started on 6 Hosts/Processors <hostA> <hostA> <hostA> <hostA> <hostA> <hostA>, Execution Home </home/user1>, Execution CWD </home/user1>;
```

SCHEDULING PARAMETERS:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

RESOURCE REQUIREMENT DETAILS:

```
Combined: select[type == local] order[r15s:pg] rusage[mem=100.00] span[hosts=1] affinity[core(1,same=socket,exclusive=(socket,injob))*1:cpubind=socket:membind=localonly:distribute=pack]
Effective: select[type == local] order[r15s:pg] rusage[mem=100.00] span[hosts=1] affinity[core(1,same=socket,exclusive=(socket,injob))*1:cpubind=socket:membind=localonly:distribute=pack]
```

AFFINITY:

HOST	CPU BINDING					MEMORY BINDING		
	TYPE	LEVEL	EXCL	IDS		POL	NUMA	SIZE
hostA	core	socket	socket	/0/0/0		local	0	16.7MB
hostA	core	socket	socket	/0/1/0		local	0	16.7MB
hostA	core	socket	socket	/0/2/0		local	0	16.7MB
hostA	core	socket	socket	/0/3/0		local	0	16.7MB
hostA	core	socket	socket	/0/4/0		local	0	16.7MB
hostA	core	socket	socket	/0/5/0		local	0	16.7MB
...								

Use **bhist -l -aff** to display historical job information about CPU and memory affinity resource requirements for job tasks.

If the job is pending, the requested affinity resources are displayed. For running jobs, the effective and combined affinity resource allocation decision made by LSF is also displayed, along with a table headed **AFFINITY** that shows detailed memory

and CPU binding information for each task, one line for each allocated processor unit. For finished jobs (EXIT or DONE state), the affinity requirements for the job, and the effective and combined affinity resource requirement details are displayed.

The following example shows **bhist** output for job 61, submitted above.

```
bhist -l -aff 61
```

```
Job <61>, User <user1>, Project <default>, Command <myjob>
Thu Feb 14 14:13:46: Submitted from host <hostA>, to Queue <normal>, CWD <$HOME>, 6 Processors Requested, Requested Resources <span[hosts=1] rusage[mem=100]affinity[core(1,same=socket,exclusive=(socket,injob)):cpubind=socket:membind=localonly:distribute=pack]>;
Thu Feb 14 14:15:07: Dispatched to 6 Hosts/Processors <hostA> <hostA> <hostA> <hostA> <hostA> <hostA>, Effective RES_REQ <select[type == local] order[r15s:pg] rusage[mem=100.00] span[hosts=1] affinity[core(1,same=socket,exclusive=(socket,injob))*1:cpubind=socket:membind=localonly:distribute=pack] >
;
```

AFFINITY:

HOST	CPU BINDING				MEMORY BINDING		
	TYPE	LEVEL	EXCL	IDS	POL	NUMA	SIZE
hostA	core	socket	socket	/0/0/0	local	0	16.7MB
hostA	core	socket	socket	/0/1/0	local	0	16.7MB
hostA	core	socket	socket	/0/2/0	local	0	16.7MB
hostA	core	socket	socket	/0/3/0	local	0	16.7MB
hostA	core	socket	socket	/0/4/0	local	0	16.7MB
hostA	core	socket	socket	/0/5/0	local	0	16.7MB

```
Thu Feb 14 14:15:07: Starting (Pid 3630709);
Thu Feb 14 14:15:07: Running with execution home </home/jsmith>, Execution CWD </home/jsmith>, Execution Pid <3630709>;
Thu Feb 14 14:16:47: Done successfully. The CPU time used is 0.0 seconds;
Thu Feb 14 14:16:47: Post job process done successfully;
```

MEMORY USAGE:

```
MAX MEM: 2 Mbytes; AVG MEM: 2 Mbytes
```

```
Summary of time in seconds spent in various states by Thu Feb 14 14:16:47
  PEND    PSUSP    RUN    USUSP    SSUSP    UNKWN    TOTAL
   81       0    100       0       0       0    181
```

Use **bacct -l -aff** to display accounting job information about CPU and memory affinity resource allocations for job tasks. A table with the heading **AFFINITY** is displayed containing the detailed affinity information for each task, one line for each allocated processor unit. CPU binding and memory binding information are shown in separate columns in the display. The following example shows **bhist** output for job 61, submitted above.

```
bacct -l -aff 61
```

Accounting information about jobs that are:

- submitted by all users.
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on all service classes.

```
Job <61>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Command <myjob>
Thu Feb 14 14:13:46: Submitted from host <hostA>, CWD <$HOME>;
Thu Feb 14 14:15:07: Dispatched to 6 Hosts/Processors <hostA> <hostA> <hostA> <hostA> <hostA> <hostA>, Effective RES_REQ <select[type == local] order[r15s:pg] rusage[mem=100.00] span[hosts=1] affinity[core(1,same=socket,exclusive=(socket,injob))*1:cpubind=socket:membind=localonly:distribute=pack] >
;
Thu Feb 14 14:16:47: Completed <done>.
```


AFFINITY:							
HOST	CPU BINDING				MEMORY BINDING		
	TYPE	LEVEL	EXCL	IDS	POL	NUMA	SIZE
hostA	core	socket	socket	/0/0/0	local	0	16.7MB
hostA	core	socket	socket	/0/1/0	local	0	16.7MB
hostA	core	socket	socket	/0/2/0	local	0	16.7MB
hostA	core	socket	socket	/0/3/0	local	0	16.7MB
hostA	core	socket	socket	/0/4/0	local	0	16.7MB
hostA	core	socket	socket	/0/5/0	local	0	16.7MB

Accounting information about this job:						
CPU_T	WAIT	TURNAROUND	STATUS	HOG_FACTOR	MEM	SWAP
0.01	81	181	done	0.0001	2M	137M

SUMMARY: (time unit: second)			
Total number of done jobs:	1	Total number of exited jobs:	0
Total CPU time consumed:	0.0	Average CPU time consumed:	0.0
Maximum CPU time of a job:	0.0	Minimum CPU time of a job:	0.0
Total wait time in queues:	81.0		
Average wait time in queue:	81.0		
Maximum wait time in queue:	81.0	Minimum wait time in queue:	81.0
Average turnaround time:	181 (seconds/job)		
Maximum turnaround time:	181	Minimum turnaround time:	181
Average hog factor of a job:	0.00 (cpu time / turnaround time)		
Maximum hog factor of a job:	0.00	Minimum hog factor of a job:	0.00

Viewing host resources for affinity jobs (-aff)

Use `bhosts -aff` or `bhosts -l -aff` to display host topology information for CPU and memory affinity scheduling. `bhosts -l -aff` cannot show remote host topology information in clusters configured with the LSF XL feature of LSF Advanced Edition.

The following fields are displayed:

Host[memory] *host_name*

Available memory on the host. If memory availability cannot be determined, a dash (-) is displayed for the host. If the `-l` option is specified with the `-aff` option, the host name is not displayed.

For hosts that do not support affinity scheduling, a dash (-) is displayed for host memory and no host topology is displayed.

NUMA[*numa_node: requested_mem / max_mem*]

Requested and available NUMA node memory. It is possible for requested memory for the NUMA node to be greater than the maximum available memory displayed.

Socket, core, and thread IDs are displayed for each NUMA node.

A *socket* is a collection of cores with a direct pipe to memory. Each socket contains 1 or more cores. This does not necessarily refer to a physical socket, but rather to the memory architecture of the machine.

A *core* is a single entity capable of performing computations. On hosts with hyperthreading enabled, a core can contain one or more threads.

For example:

`bhosts -l -aff hostA`

HOST	hostA											
STATUS	CPUF	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV	DISPATCH_WINDOW			
ok	60.00	-	8	0	0	0	0	0	-			

CURRENT LOAD USED FOR SCHEDULING:												
	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem	slots
Total	0.0	0.0	0.0	30%	0.0	193	25	0	8605M	5.8G	13.2G	8
Reserved	0.0	0.0	0.0	0%	0.0	0	0	0	0M	0M	0M	-

LOAD THRESHOLD USED FOR SCHEDULING:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

CONFIGURED AFFINITY CPU LIST: all**AFFINITY: Enabled**

Host[15.7G]

NUMA[0: 0M / 15.7G]

Socket0

core0 (0)

Socket1

core0 (1)

Socket2

core0 (2)

Socket3

core0 (3)

Socket4

core0 (4)

Socket5

core0 (5)

Socket6

core0 (6)

Socket7

core0 (7)

When LSF detects missing elements in the topology, it attempts to correct the problem by adding the missing levels into the topology. For example, sockets and cores are missing on `hostB` below:

...

Host[1.4G] `hostB`

NUMA[0: 1.4G / 1.4G] (*0 *1)

...

A job requesting 2 cores, or 2 sockets, or 2 CPUs will run. Requesting 2 cores from the same NUMA node will also run. However, a job requesting 2 cores from the same socket will remain pending.

Use `lshosts -T` to display host topology information for each host.

Displays host topology information for each host or cluster:

The following fields are displayed:

Host[*memory*] *host_name*

Maximum memory available on the host followed by the host name. If memory availability cannot be determined, a dash (-) is displayed for the host.

For hosts that do not support affinity scheduling, a dash (-) is displayed for host memory and no host topology is displayed.

NUMA[*numa_node: max_mem*]

Maximum NUMA node memory. It is possible for requested memory for the NUMA node to be greater than the maximum available memory displayed.

If no NUMA nodes are present, then the NUMA layer in the output is not shown. Other relevant items such as host, socket, core and thread are still shown.

If the host is not available, only the host name is displayed. A dash (-) is shown where available host memory would normally be displayed.

A *socket* is a collection of cores with a direct pipe to memory. Each socket contains 1 or more cores. This does not necessarily refer to a physical socket, but rather to the memory architecture of the machine.

A *core* is a single entity capable of performing computations. On hosts with hyperthreading enabled, a core can contain one or more threads.

lshosts -T differs from the **bhosts -aff** output:

- Socket and core IDs are not displayed for each NUMA node.
- The requested memory of a NUMA node is not displayed
- **lshosts -T** displays all enabled CPUs on a host, not just those defined in the CPU list in `lsb.hosts`

A node contains sockets, a socket contains cores, and a core can contain threads if the core is enabled for multithreading.

In the following example, full topology (NUMA, socket, and core) information is shown for `hostA`. Hosts `hostB` and `hostC` are either not NUMA hosts or they are not available:

```
lshosts -T
Host[15.7G] hostA
  NUMA[0: 15.7G]
    Socket
      core (0)
    Socket
      core (1)
    Socket
      core (2)
    Socket
      core (3)
    Socket
      core (4)
    Socket
      core (5)
    Socket
      core (6)
    Socket
      core (7)
```

```
Host[-] hostB
```

```
Host[-] hostC
```

When LSF cannot detect processor unit topology, **lshosts -T** displays processor units to the closest level. For example:

```
lshosts -T
Host[1009M] hostA
  Socket (0 1)
```

On `hostA` there are two processor units: 0 and 1. LSF cannot detect core information, so the processor unit is attached to the socket level.

Hardware topology information is not shown for client hosts and hosts in a mixed cluster or MultiCluster environment running a version of LSF that is older than 10.1.0.

Affinity preemption

To enable affinity preemption, set the `PREEMPT_JOBTYPE = AFFINITY` parameter in the Parameters section of the `lsb.params` file. By default, affinity resources are not preemptable.

Affinity preemption supports the following:

- Preemption of affinity resources (cores, threads, sockets, NUMA nodes, and NUMA memory)
- Backfill of reserved affinity resources
- Pending License Scheduler jobs can use the affinity resources of a suspended License Scheduler job, as long as both jobs request at least one license in common

Affinity preemption interacts with the following LSF features:

Queue-based affinity resource preemption

A running job with affinity requirements may occupy cores in a low priority queue. When affinity preemption is enabled, a pending job in a high priority queue that also has an affinity requirement is potentially able to preempt the running job in the low priority queue to get its affinity resources (threads, cores, sockets, NUMA nodes). When

PREEMPTABLE_RESOURCES = mem is enabled in lsb.params a higher priority affinity job can preempt a running low priority job for host memory, NUMA memory as well as slots.

Affinity resources are treated similar to slots and memory: when a job is suspended, the job continues to occupy its slots and its affinity resources, preventing another job from using these resources, unless that other job is in a queue that has a preemption relationship with the suspended job.

Affinity resource backfill

A job in a reservation queue may reserve slots, memory and affinity resources (and potentially other reservable resources). If the reserving job has an affinity requirement, LSF can reserve affinity resources for the job. A job in a backfill queue that has an affinity requirement can use the reserved affinity resources of a pending job if the backfill job is expected to finish before the earliest expected start time of the reserving job. The rule of thumb is that if a job in a backfill queue is able to use the slots reserved by another job during backfill scheduling, then it should be also able to use the reserved affinity resources. Affinity backfill is enabled by default, and cannot be disabled.

License Scheduler affinity resource preemption

Slots, and optionally, memory are released by a suspended License Scheduler job only to other License Scheduler jobs that request at least one license in common with the suspended job.

This feature also applies to affinity resources. Once a License Scheduler job is suspended, the affinity resources occupied by the job are available to other License Scheduler jobs that request at least one license in common with the suspended job, in its usage. When LSF_LIC_SCHED_PREEMPT_SLOT_RELEASE=N in lsf.conf, affinity resources along with slots and memory are not released to pending License Scheduler jobs. LSF_LIC_SCHED_PREEMPT_SLOT_RELEASE is enabled by default.

Preemption queue preference

You can configure which queues should have preference to preempt from. Preemption queue preference is enabled by USE_SUSP_SLOTS=Y in lsb.params. USE_SUSP_SLOTS=Y supports affinity preemption. With this parameter enabled, pending jobs in preemptable queues are allowed to use the slots of suspended jobs in higher priority preemptive queues. The queues must have a preemption relationship with each other. When USE_SUSP_SLOTS=N, pending jobs in a low priority preemptable queue cannot use the slots of a suspended job in a high priority preemptive queue. When USE_SUSP_SLOTS=Y, then pending jobs in preemptable queues are allowed to use the affinity resources occupied by suspended jobs in higher priority preemptive queues, if the queues have a preemption relationship. Note that SSUSP jobs on a host are always allowed to try to resume and use the non-releasable resources, including slots, memory, and affinity resources, occupied by other suspended jobs on the same host.

Memory preemption

By default, LSF considers memory to be a non-releasable resource. When a running job is suspended, LSF continues to reserve memory for the suspended job. When memory preemption is enabled by setting PREEMPTABLE_RESOURCES = mem in lsb.params, jobs with memory requirements submitted to high priority preemptive queues can preempt jobs in low priority queues for memory. When LSF allows jobs in preemptive queues to use memory reserved for suspended jobs, LSF essentially allows host memory to be overcommitted. Host-based memory is a separate resource from the memory reservations made on the NUMA nodes. However, preemption can be triggered for NUMA-level memory as well when memory is configured as a preemptable resource.

Affinity binding based on Linux cgroup cpuset subsystem

LSF can enforce CPU binding on systems that support the Linux cgroup cpuset subsystem. When CPU affinity binding through Linux cgroups is enabled, LSF will create a cpuset to contain job processes if the job has affinity resource requirements, so that the job processes cannot escape from the allocated CPUs. Each affinity job cpuset includes only the CPU and memory nodes that LSF distributes. Linux cgroup cpusets are only created for affinity jobs.

With this feature, LSF collects processor topology from hosts, including NUMA nodes, sockets, cores, and hyperthreads. Users can submit jobs specifying how processes of a job should be bound to these computing elements. LSF uses the system call sched_setaffinity() to bind CPUs. It is possible for user applications to escape from the bound CPUs by calling sched_setaffinity() directly to bind to other CPUs.

For example, submit a job with core affinity requirement and localprefer memory binding:

```
bsub -R "affinity[core:membind=localprefer]" ./myapp
```

LSF will create a cpuset which contains one core and attach the process ID of the application ./myapp to this cpuset. The cpuset serves as a strict container for job processes, so that the application ./myapp cannot bind to other CPUs.

In this example, the memory binding policy is `localprefer`. When `membind=localprefer`, or it is not specified, LSF adds all memory nodes to the cpuset to make sure the job can access all memory nodes on the host, and will make sure job processes will access preferred memory nodes first. If the memory binding policy is `localonly`, LSF only adds the memory nodes that the LSF scheduler distributes to the cpuset, and `myapp` only uses those memory nodes, not all memory nodes.

To enable the cpuset enforcement feature, configure `LSB_RESOURCE_ENFORCE="cpu"` in `lsf.conf`.

Portable hardware locality

Portable Hardware Locality (**hwloc**) is an open source software package that is distributed under BSD license. It provides a portable abstraction (across OS, versions, architectures, and so on) of the hierarchical topology of modern architectures, including NUMA memory nodes, socket, shared caches, cores, and simultaneous multithreading (SMT). **hwloc** is integrated into LSF to detect hardware information, and can support most of the platforms that LSF supports.

Functionality

The **hwloc** package gathers various system attributes such as cache and memory information as well as the locality of I/O device such as network interfaces. It primarily aims at helping applications with gathering information about computing hardware.

It also detects each host hardware topology when the LIM starts and the host topology information is changed. The management host LIM detects the topology of the management host. The server host LIM detects the topology of the local host. It updates the topology information to the management host when it joins the cluster or sends topology information to the management host LIM for host configuration. Host topology information is updated once the hardware topology changes. Hardware topology changes if any NUMA memory node, caches, socket, core, PU and so on, changes. Sometimes topology information changes even though the core number did not change.

Use the **lim -T** and **lshosts -T** commands to display host topology information. The **lim -t** command displays the total number of NUMA nodes, total number of processors, total number of cores, and total number of threads.

Structure of topology

A NUMA node contains sockets. Each socket contains cores (processes) which contain threads. If there is no hwloc library, LSF uses the PCT logic. Some AMD CPUs have the opposite structure where socket nodes contain NUMA nodes. The hierarchies of the topology is similar to a tree. Therefore, the host topology information (NUMA memory nodes, caches, sockets, cores, PUs, and so on) from hwloc is organized as a tree. Each tree node has its type. The type includes host, NUMA, socket, cache, core, and pu. Each tree node also includes its attributes.

In the following example, `hostA` has 64 GB of memory and two NUMA nodes. Each socket node has one NUMA, eight cores, 16 PUs (two PUs per core), and 32 GB of memory. Both the NUMA nodes and the PUs are numbered in series that is provided by the system. LSF displays NUMA information based on the level it detects from the system. The output format displays as a tree, and the NUMA information displays as `NUMA[ID: memory]`. The PU displays as `parent_node(ID ID ...)`, where `parent_node` may be host, NUMA, socket, or core.

In the following example, `NUMA[0: 32G]` means that the NUMA ID is 0 and has 32 GB of memory. `core0(0 16)` means that there are two PUs under the parent core node, and the ID of the two PUs are 0 and 16.

```
Host[64G] hostA
Socket0
  NUMA[0: 32G]
    core0(0 16)
    core1(1 17)
    core2(2 18)
    core3(3 19)
    core4(4 20)
    core5(5 21)
    core6(6 22)
    core7(7 23)
Socket1
  NUMA[1: 32G]
```

```
core8 (8 24)
core9 (9 25)
core10 (10 26)
core11 (11 27)
core12 (12 28)
core13 (13 29)
core14 (14 30)
core15 (15 31)
```

Some CPUs, especially old ones, may have incomplete hardware topology in terms of missing information for NUMA, socket, or core. Therefore, their topology is incomplete.

For example,

- `hostB` (with one Intel Pentium 4 CPU) has 2G of memory, one socket, one core, and two PUs per core. Information on `hostB` is displayed as follows:

```
Host[2G] hostB
Socket
  core (0 1)
```

- `hostC` (with one Intel Itanium CPU) has 4 GB of memory, and two PUs. Information on `hostC` is displayed as follows:

```
Host[4G] (0 1) hostC
```

Some platforms or operating system versions will only report a subset of topology information.

For example, `hostD` has the same CPU as `hostB`, but `hostD` is running RedHat Linux 4, which does not supply core information. Therefore, information on `hostD` is displayed as follows:

```
Host[1009M] hostD
Socket (0 1)
```

Dynamically load the hwloc library

You can configure LSF to dynamically load the **hwloc** library from the system library paths to detect newer hardware. This allows you to use the latest supported version of the **hwloc** and LSF integration at any time if there are no compatibility issues between this version of the **hwloc** library and header file for **hwloc**. If LSF fails to load the library, LSF defaults to using the **hwloc** functions in the static library.

Enable the dynamic loading of the **hwloc** library by enabling the `LSF_HWLOC_DYNAMIC` parameter in the `lsf.conf` file.

Processor binding for LSF job processes

Processor binding for LSF job processes takes advantage of the power of multiple processors and multiple cores to provide hard processor binding functionality for sequential LSF jobs and parallel jobs that run on a single host.

Rapid progress of modern processor manufacture technologies has enabled the low-cost deployment of LSF on hosts with multicore and multithread processors. The default soft affinity policy enforced by the operating system scheduler may not give optimal job performance. For example, the operating system scheduler may place all job processes on the same processor or core leading to poor performance. Frequently switching processes as the operating system schedules and reschedules work between cores can cause cache invalidations and cache miss rates to grow large.

Restriction: Processor binding is supported on hosts running Linux with kernel version 2.6 or higher.

For multi-host parallel jobs, LSF sets two environment variables (`LSB_BIND_JOB` and `LSB_BIND_CPU_LIST`) but does not attempt to bind the job to any host.

When processor binding for LSF job processes is enabled on supported hosts, job processes of an LSF job are bound to a processor according to the binding policy of the host. When an LSF job is completed (exited or done successfully) or suspended, the corresponding processes are unbound from the processor.

When a suspended LSF job is resumed, the corresponding processes are bound again to a processor. The process is not guaranteed to be bound to the same processor it was bound to before the job was suspended.

The processor binding affects the whole job process group. All job processes forked from the root job process (the job RES) are bound to the same processor.

Processor binding for LSF job processes does not bind daemon processes.

If processor binding is enabled, but the execution hosts do not support processor affinity, the configuration has no effect on the running processes. Processor binding has no effect on a single-processor host.

Processor, core, and thread-based CPU binding

By default, the number of CPUs on a host represents the number of cores a machine has. For LSF hosts with multiple cores, threads, and processors, `ncpus` can be defined by the cluster administrator to consider one of the following:

- Processors
- Processors and cores
- Processors, cores, and threads

Globally, this definition is controlled by the parameter `EGO_DEFINE_NCPUS` in `lsf.conf` or `ego.conf`. The default behavior for `ncpus` is to consider the number of cores (`EGO_DEFINE_NCPUS=cores`).

Note: When `PARALLEL_SCHED_BY_SLOT=Y` in `lsb.params`, the resource requirement string keyword `ncpus` refers to the number of slots instead of the number of CPUs, however **lshosts** output will continue to show `ncpus` as defined by `EGO_DEFINE_NCPUS` in `lsf.conf`.

Binding job processes randomly to multiple processors, cores, or threads, may affect job performance. Processor binding configured with `LSF_BIND_JOB` in `lsf.conf` or `BIND_JOB` in `lsb.applications`, detects the `EGO_DEFINE_NCPUS` policy to bind the job processes by processor, core, or thread (PCT).

For example, if the PCT policy for the host is set to processor (`EGO_DEFINE_NCPUS=procs`) and the binding option is set to `BALANCE`, the first job process is bound to the first physical processor, the second job process is bound to the second physical processor and so on.

If the PCT policy for the host is set to core level (`EGO_DEFINE_NCPUS=cores`) and the binding option is set to `BALANCE`, the first job process is bound to the first core on the first physical processor, the second job process is bound to the first core on the second physical processor, the third job process is bound to the second core on the first physical processor, and so on.

If the PCT policy for the host is set to thread level (`EGO_DEFINE_NCPUS=threads`) and the binding option is set to `BALANCE`, the first job process is bound to the first thread on the first physical processor, the second job process is bound to the first thread on the second physical processor, the third job process is bound to the second thread on the first physical processor, and so on.

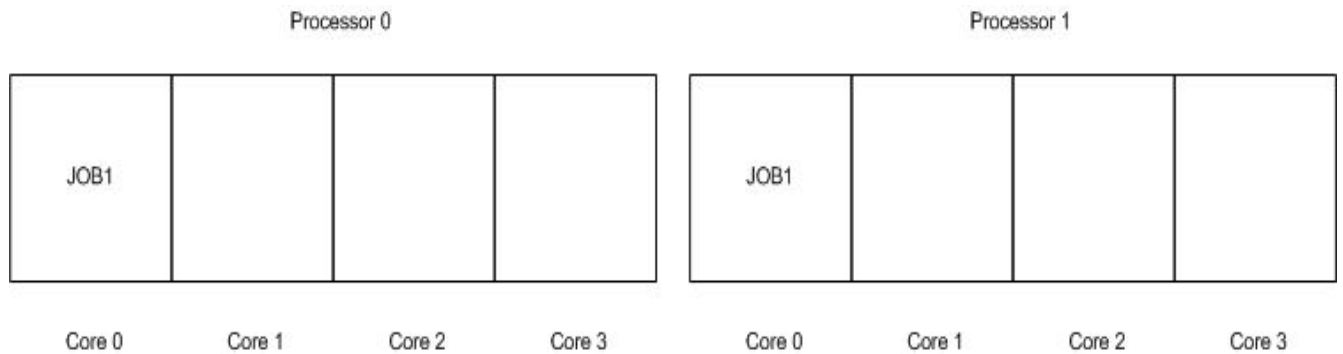
Note: `BIND_JOB` and `LSF_BIND_JOB` are deprecated in LSF Standard Edition and LSF Advanced Edition. You should enable LSF CPU and memory affinity scheduling in with the `AFFINITY` parameter in `lsb.hosts`. If both `BIND_JOB` and affinity scheduling are enabled, affinity scheduling takes effect, and `BIND_JOB` is disabled. If both `LSF_BIND_JOB` and affinity scheduling are enabled, affinity scheduling takes effect, and `LSF_BIND_JOB` is disabled. `BIND_JOB` and `LSF_BIND_JOB` are the only affinity options available in LSF Express Edition.

BIND_JOB=BALANCE

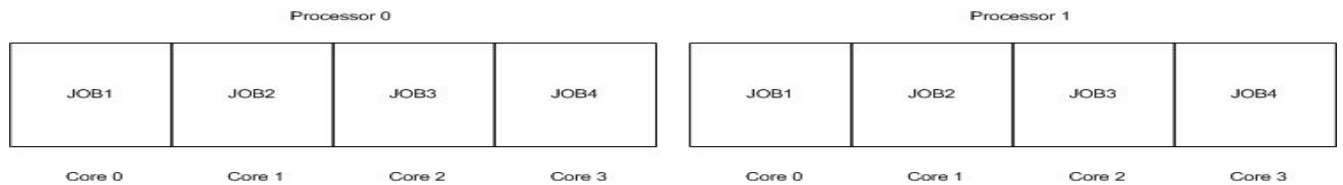
The `BIND_JOB=BALANCE` option instructs LSF to bind the job that is based on the load of the available processors/cores/threads. For each slot:

- If the PCT level is set to processor, the lowest loaded physical processor runs the job.
- If the PCT level is set to core, the lowest loaded core on the lowest loaded processor runs the job.
- If the PCT level is set to thread, the lowest loaded thread on the lowest loaded core on the lowest loaded processor runs the job.

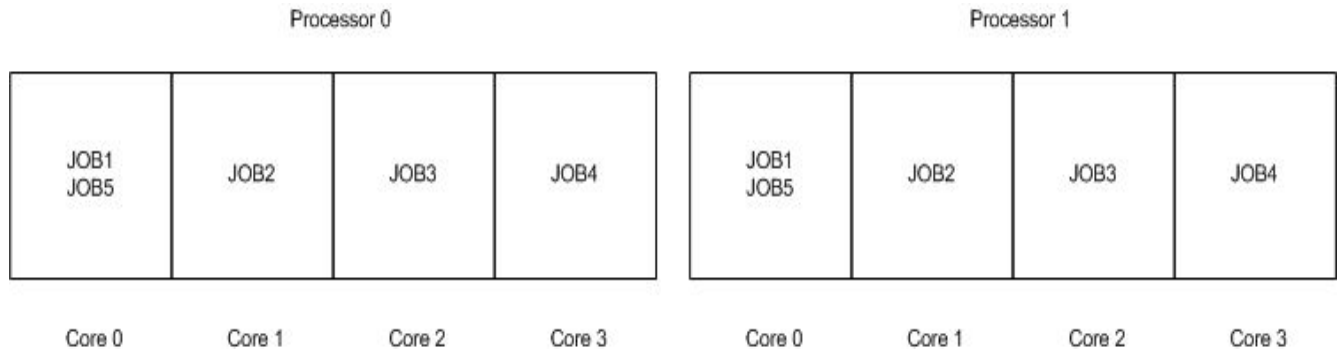
If there is a single 2 processor quad core host and you submit a parallel job with `-n 2 -R"span[hosts=1]"` when the PCT level is core, the job is bound to the first core on the first processor and the first core on the second processor:



After submitting another three jobs with `-n 2 -R"span[hosts=1]":`



If `PARALLEL_SCHED_BY_SLOT=Y` is set in `lsb.params`, the job specifies a maximum and minimum number of job slots instead of processors. If the `MXJ` value is set to 16 for this host (there are 16 job slots on this host), LSF can dispatch more jobs to this host. Another job submitted to this host is bound to the first core on the first processor and the first core on the second processor:



BIND_JOB=PACK

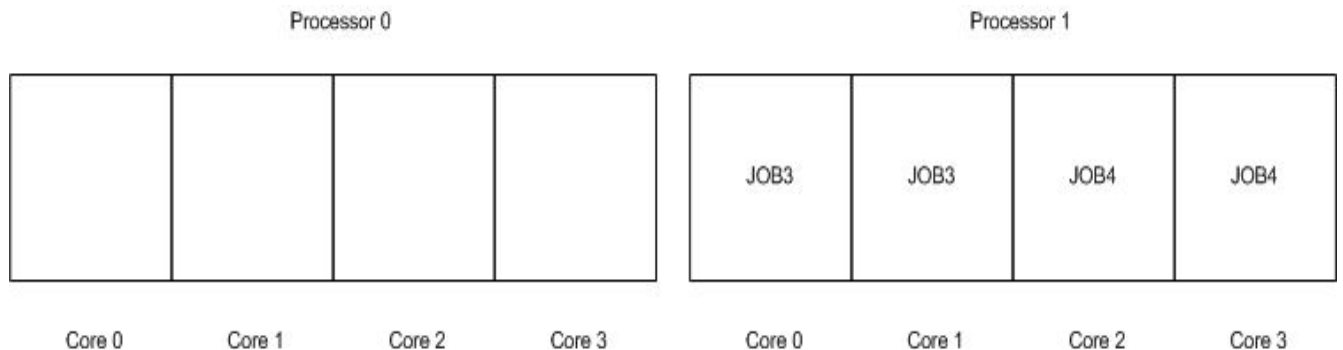
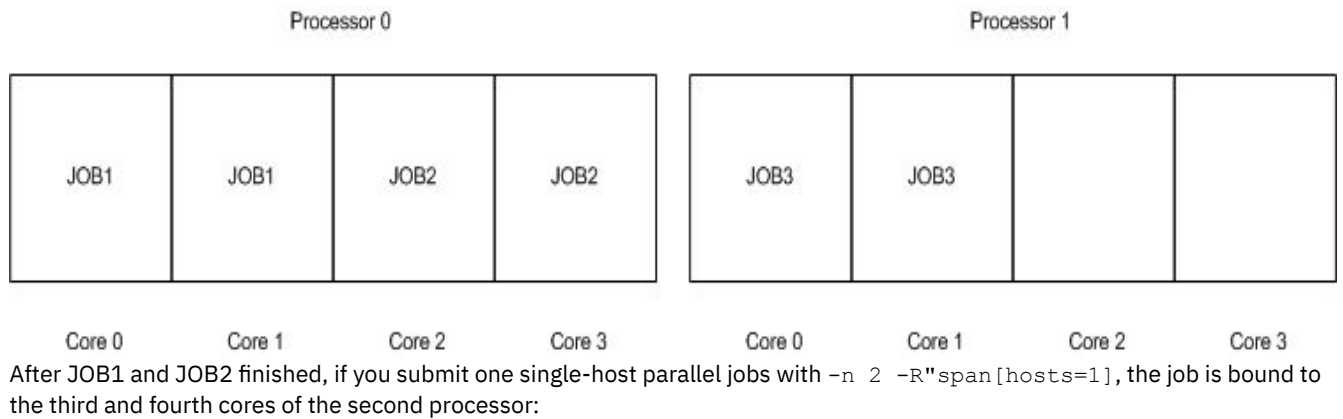
The `BIND_JOB=PACK` option instructs LSF to try to pack all the processes onto a single processor. If this cannot be done, LSF tries to use as few processors as possible. Email is sent to you after job dispatch and when job finishes. If no processors/cores/threads are free (when the PCT level is processor/core/thread level), LSF tries to use the `BALANCE` policy for the new job.

LSF depends on the order of processor IDs to pack jobs to a single processor.

If PCT level is processor, there is no difference between `BALANCE` and `PACK`.

This option binds jobs to a single processor where it makes sense, but does not oversubscribe the processors/cores/threads. The other processors are used when they are needed. For instance, when the PCT level is core level, if we have a single four processor quad core host and we had bound 4 sequential jobs onto the first processor, the 5th-8th sequential job is bound to the second processor.

If you submit three single-host parallel jobs with `-n 2 -R"span[hosts=1]"` when the PCT level is core level, the first job is bound to the first and second cores of the first processor, the second job is bound to the third and fourth cores of the first processor. Binding the third job to the first processor oversubscribes the cores in the first processor, so the third job is bound to the first and second cores of the second processor:



BIND_JOB=ANY

BIND_JOB=ANY binds the job to the first N available processors/cores/threads with no regard for locality. If the PCT level is core, LSF binds the first N available cores regardless of whether they are on the same processor or not. LSF arranges the order based on APIC ID.

If PCT level is processor (default value after installation), there is no difference between ANY and BALANCE.

For example, with a single 2-processor quad core host and the following table is the relationship of APIC ID and logic processor/core id:

APC ID	Processor ID	Core ID
0	0	0
1	0	1
2	0	2
3	0	3
4	1	0
5	1	1
6	1	2
7	1	3

If the PCT level is core level and you submit two jobs to this host with `-n 3 -R "span[hosts=1]"`, then the first job is bound to the first, second, and third core of the first physical processor, the second job is bound to the fourth core of the first physical processor and the first, second core in the second physical processor.

BIND_JOB=USER

The BIND_JOB=USER parameter binds the job to the value of `$LSB_USER_BIND_JOB` as specified in the user submission environment. This allows the Administrator to delegate binding decisions to the actual user. This value must be one of Y, N, NONE, BALANCE, PACK, or ANY. Any other value is treated as ANY.

BIND_JOB=USER_CPU_LIST

The BIND_JOB=USER_CPU_LIST parameter binds the job to the explicit logic CPUs specified in environment variable \$LSB_USER_BIND_CPU_LIST. LSF does not check that the value is valid for the execution host(s). It is the user's responsibility to correctly specify the CPU list for the hosts they select.

The correct format of \$LSB_USER_BIND_CPU_LIST is a list which may contain multiple items, separated by comma, and ranges. For example, 0,5,7,9-11.

If the value's format is not correct or there is no such environment variable, jobs are not bound to any processor.

If the format is correct and it cannot be mapped to any logic CPU, the binding fails. But if it can be mapped to some CPUs, the job is bound to the mapped CPUs. For example, with a two-processor quad core host and the logic CPU ID is 0-7:

1. If user1 specifies 9,10 into \$LSB_USER_BIND_CPU_LIST, his job is not bound to any CPUs.
2. If user2 specifies 1,2,9 into \$LSB_USER_BIND_CPU_LIST, his job is bound to CPU 1 and 2.

If the value's format is not correct or it does not apply for the execution host, the related information is added to the email sent to users after job dispatch and job finish.

If user specifies a minimum and a maximum number of processors for a single-host parallel job, LSF may allocate processors between these two numbers for the job. In this case, LSF binds the job according to the CPU list specified by the user.

BIND_JOB=NONE

The BIND_JOB=NONE parameter is functionally equivalent to the the former BIND_JOB=N parameter where the processor binding is disabled.

Feature interactions

- Existing CPU affinity featuresProcessor binding of LSF job processes will not take effect on a management host with the following parameters configured.
 - MBD_QUERY_CPUS
 - LSF_DAEMONS_CPUS
 - EGO_DAEMONS_CPUS
- Job requeue, rerun, and migration
When a job is requeued, rerun or migrated, a new job process is created. If processor binding is enabled when the job runs, the job processes will be bound to a processor.
- **bctrl restart sbd**
bctrl restart sbd restarts a new sbatchd. If a job process has already been bound to a processor, after sbatchd is restarted, processor binding for the job processes are restored.
- **badmin reconfig**
If the BIND_JOB parameter is modified in an application profile, **badmin reconfig** only affects pending jobs. The change does not affect running jobs.
- LSF multicluster capability job forwarding model
In the LSF multicluster capability environment, the behavior is similar to the current application profile behavior. If the application profile name specified in the submission cluster is not defined in the execution cluster, the job is rejected. If the execution cluster has the same application profile name, but does not enable processor binding, the job processes are not bound at the execution cluster.
- [Enabling processor binding for LSF job processes](#)
- [Processor binding for parallel jobs](#)
By default, there is no processor binding.

Enabling processor binding for LSF job processes

About this task

LSF supports the following binding options for sequential jobs and parallel jobs that run on a single host:

- BALANCE
- PACK
- ANY
- USER
- USER_CPU_LIST
- NONE

Procedure

Enable processor binding cluster-wide or in an application profile.

- Cluster-wide configuration (lsf.conf)

Define LSF_BIND_JOB in lsf.conf to enable processor binding for all execution hosts in the cluster. On the execution hosts that support this feature, job processes are hard bound to selected processors.

- Application profile configuration (lsb.applications)

Define BIND_JOB in an application profile configuration in lsb.applications to enable processor binding for all jobs that are submitted to the application profile. On the execution hosts that support this feature, job processes are hard bound to selected processors.

If BIND_JOB is not set in an application profile in lsb.applications, the value of LSF_BIND_JOB in lsf.conf takes effect. The BIND_JOB parameter that is configured in an application profile overrides the lsf.conf setting.

Note: BIND_JOB and LSF_BIND_JOB are deprecated in LSF Standard Edition and LSF Advanced Edition. You should enable LSF CPU and memory affinity scheduling in with the AFFINITY parameter in lsb.hosts. If both BIND_JOB and affinity scheduling are enabled, affinity scheduling takes effect, and BIND_JOB is disabled. If both LSF_BIND_JOB and affinity scheduling are enabled, affinity scheduling takes effect, and LSF_BIND_JOB is disabled. BIND_JOB and LSF_BIND_JOB are the only affinity options available in LSF Express Edition.

Processor binding for parallel jobs

By default, there is no processor binding.

For multi-host parallel jobs, LSF sets two environment variables (\$LSB_BIND_JOB and \$LSB_BIND_CPU_LIST) but does not attempt to bind the job to any host even if you enable the processor binding.

Re-sizable jobs

Adding slots to or removing slots from a re-sizable job triggers unbinding and rebinding of job processes. Rebinding does not guarantee that the processes can be bound to the same processors they were bound to previously.

If a multi-host parallel job becomes a single-host parallel job after resizing, LSF does not bind it.

If a single-host parallel job or sequential job becomes a multi-host parallel job after resizing, LSF does not bind it.

After unbinding and binding, the job CPU affinity is changed. LSF puts the new CPU list in the LSB_BIND_CPU_LIST environment variable and the binding method to LSB_BIND_JOB environment variable. And it is the responsibility of the notification command to tell the job that CPU binding has changed.

Running parallel jobs with blaunch

Learn how to configure and use the **blaunch** command for launching parallel and distributed applications within LSF. Task geometry allows for flexibility in how tasks are grouped for execution on system nodes. A typical LSF parallel job launches its tasks across multiple hosts. By default you can enforce limits on the total resources used by all the tasks in the job.

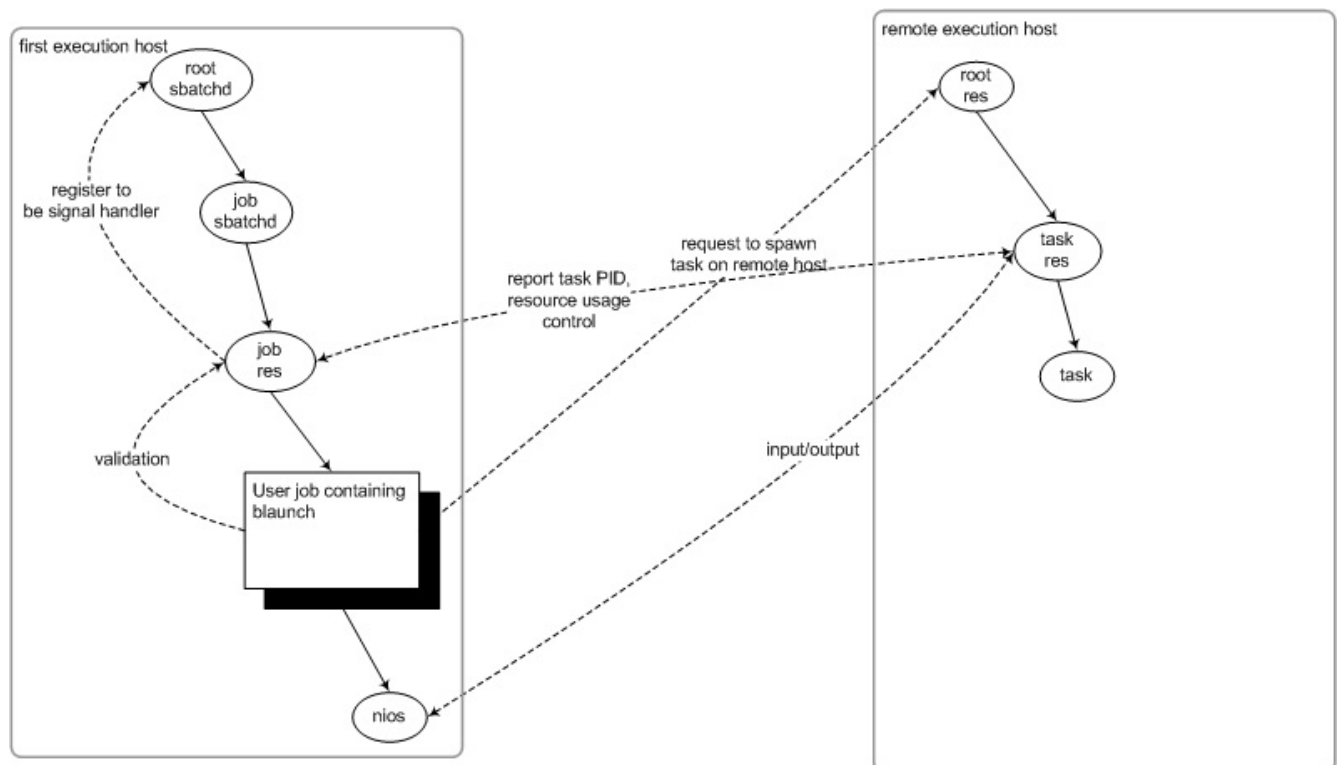
- [blaunch distributed application framework](#)
Most MPI implementations and many distributed applications use **rsh** and **ssh** as their task launching mechanism. The **blaunch** command provides a drop-in replacement for **rsh** and **ssh** as a transparent method for launching parallel and distributed applications within LSF.
- [SGI vendor MPI support](#)
Run your SGI MPI jobs through LSF.
- [Running jobs with task geometry](#)
Specifying task geometry allows you to group tasks of a parallel job step to run together on the same node. Task geometry allows for flexibility in how tasks are grouped for execution on system nodes. You cannot specify the particular nodes that these groups run on; the scheduler decides which nodes run the specified groupings.
- [Enforcing resource usage limits for parallel tasks](#)
A typical LSF parallel job launches its tasks across multiple hosts. By default you can enforce limits on the total resources used by all the tasks in the job.

blaunch distributed application framework

Most MPI implementations and many distributed applications use **rsh** and **ssh** as their task launching mechanism. The **blaunch** command provides a drop-in replacement for **rsh** and **ssh** as a transparent method for launching parallel and distributed applications within LSF.

About the blaunch command

The following figure illustrates **blaunch** processing:



Similar to the LSF **lsrun** command, **blaunch** transparently connects directly to the RES and **sbatchd** on the remote host, and subsequently creates and tracks the remote tasks, and provides the connection back to LSF. You do not need to insert **pam** or **taskstarter** into the **rsh** or **ssh** calling sequence, or configure any wrapper scripts.

blaunch supports the following core command line options as **rsh** and **ssh**:

- **rsh** *host_name* *command*
- **ssh** *host_name* *command*

The host name value for **rsh** and **ssh** can only be a single host name, so you can use the **-z** option to specify a space-delimited list of hosts where tasks are started in parallel. All other **rsh** and **ssh** options are silently ignored.

You cannot run the **blaunch** command directly from the command line as a standalone command. **blaunch** only works within an LSF job; it can only be used to launch tasks on remote hosts that are part of a job allocation. On success, **blaunch** exits with 0.

Restriction: You cannot run concurrent **blaunch** commands in background mode.

blaunch is supported on Windows 2000 or later with the following exceptions:

- Only the following signals are supported: **SIGKILL**, **SIGSTOP**, **SIGCONT**.
- The **-n** option is not supported.
- `CMD.EXE /C <user command line>` is used as intermediate command shell when **-no-shell** is not specified
- `CMD.EXE /C` is not used when **-no-shell** is specified.
- Windows User Account Control must be configured correctly to run jobs.

LSF APIs for the blaunch distributed application framework

LSF provides the following APIs for programming your own applications to use the **blaunch** distributed application framework:

- `lsb_launch()`: Synchronous API call to allow source level integration with vendor MPI implementations. This API launches the specified command (**argv**) on the remote nodes in parallel. LSF must be installed before integrating your MPI implementation with `lsb_launch()`. The `lsb_launch()` API requires the full set of `liblsf.so`, `libbat.so` (or `liblsf.a`, `libbat.a`).
- `lsb_getalloc()`: Allocates memory for a host list to be used for launching parallel tasks through **blaunch** and the `lsb_launch()` API. It is the responsibility of the caller to free the host list when it is no longer needed. On success, the host list is a list of strings. Before freeing the host list, the individual elements must be freed. An application using the `lsb_getalloc()` API is assumed to be part of an LSF job, and that `LSB_MCPU_HOSTS` is set in the environment.

The blaunch job environment

blaunch determines from the job environment what job it is running under, and what the allocation for the job is. These can be determined by examining the environment variables `LSB_JOBID`, `LSB_JOBINDEX`, and `LSB_MCPU_HOSTS`. If any of these variables do not exist, **blaunch** exits with a non-zero value. Similarly, if **blaunch** is used to start a task on a host not listed in `LSB_MCPU_HOSTS`, the command exits with a non-zero value.

The job submission script contains the **blaunch** command in place of **rsh** or **ssh**. The **blaunch** command does sanity checking of the environment to check for `LSB_JOBID` and `LSB_MCPU_HOSTS`. The **blaunch** command contacts the job RES to validate the information determined from the job environment. When the job RES receives the validation request from **blaunch**, it registers with the root **sbatchd** to handle signals for the job.

The job RES periodically requests resource usage for the remote tasks. This message also acts as a heartbeat for the job. If a resource usage request is not made within a certain period of time it is assumed the job is gone and that the remote tasks should be shut down. This timeout is configurable in an application profile in `lsb.applications`.

The **blaunch** command also honors the parameters `LSB_CMD_LOG_MASK`, `LSB_DEBUG_CMD`, and `LSB_CMD_LOGDIR` when defined in `lsf.conf` or as environment variables. The environment variables take precedence over the values in `lsf.conf`.

To ensure that no other users can run jobs on hosts allocated to tasks launched by **blaunch** set the `LSF_DISABLE_LSRUN=Y` parameter in the `lsf.conf` file. When the `LSF_DISABLE_LSRUN=Y` parameter is defined, RES refuses remote connections from **lsrun** and **lsgrun** unless the user is either an LSF administrator or root. The `LSF_ROOT_REX` parameter must be defined for remote execution by root. Other remote execution commands, such as **ch** and **lsmake** are not affected.

Job control actions defined in the `JOB_CONTROLS` parameter in the `lsb.queues` file only take effect on the first execution host. Job control actions defined in the queue do not affect tasks running on other hosts. If the `JOB_CONTROLS` parameter is defined, the default job control signals of LSF (`SUSPEND`, `RESUME`, `TERMINATE`) do not reach each task on each execution host.

Temporary directory for tasks launched by blaunch

By default, LSF creates a temporary directory for a job only on the first execution host. If the LSF_TMPDIR parameter is set in the lsf.conf file, the path of the job temporary directory on the first execution host is set to LSF_TMPDIR/job_ID.tmpdir.

If the LSB_SET_TMPDIR= Y parameter is set, the environment variable *TMPDIR* will be set equal to the path specified by LSF_TMPDIR. This value for *TMPDIR* overrides any value that might be set in the submission environment.

Tasks launched through the **blaunch** distributed application framework make use of the LSF temporary directory specified by the LSF_TMPDIR parameter:

- When the environment variable *TMPDIR* is set on the first execution host, the **blaunch** framework propagates this environment variable to all execution hosts when launching remote tasks.
- The job RES or the task RES creates the directory specified by *TMPDIR* if it does not already exist before starting the job.
- The directory created by the job RES or task RES has permission 0700 and is owned by the execution user.
- If the *TMPDIR* directory was created by the task RES, LSF deletes the temporary directory and its contents when the task is complete.
- If the *TMPDIR* directory was created by the job RES, LSF will delete the temporary directory and its contents when the job is done.
- If the *TMPDIR* directory is on a shared file system, it is assumed to be shared by all the hosts allocated to the **blaunch** job, so LSF does not remove *TMPDIR* directories created by the job RES or task RES.

Automatic generation of the job host file

LSF automatically places the allocated hosts for a job into the *\$LSB_HOSTS* and *\$LSB_MCPU_HOSTS* environment variables. Since most MPI implementations and parallel applications expect to read the allocated hosts from a file, LSF creates a host file in the default job output directory \$HOME/.lsbatch on the execution host before the job runs, and deletes it after the job has finished running. The name of the host file created has the format:

.lsb.<jobid>.hostfile

The host file contains one host per line. For example, if *LSB_MCPU_HOSTS*="hostA 2 hostB 2 hostC 1", the host file contains the following host names:

- hostA
- hostA
- hostB
- hostB
- hostC

LSF publishes the full path to the host file by setting the environment variable *LSB_DJOB_HOSTFILE*.

Handle remote task exit

You can configure an application profile in lsb.applications to control the behavior of a parallel or distributed application when a remote task exits. Specify a value for the RTASK_GONE_ACTION parameter in the application profile to define what the LSF does when a remote task exits. The default behavior is as follows:

- When task exits with zero value, LSF does nothing.
- When task exits with non-zero value, LSF does nothing.
- When task crashes, LSF shuts down the entire job.

The RTASK_GONE_ACTION parameter has the following syntax:

```
RTASK_GONE_ACTION="[KILLJOB_TASKDONE | KILLJOB_TASKEEXIT]
[IGNORE_TASKCRASH]"
```

Where:

- The IGNORE_TASKCRASH parameter: A remote task crashes. LSF does nothing. The job continues to launch the next task.

- The KILLJOB_TASKDONE parameter: A remote task exits with zero value. LSF terminates all tasks in the job.
- The KILLJOB_TASKEXIT parameter: A remote task exits with non-zero value. LSF terminates all tasks in the job.

For example:

```
RTASK_GONE_ACTION="IGNORE_TASKCRASH KILLJOB_TASKEXIT"
```

The RTASK_GONE_ACTION parameter only applies to the **blaunch** distributed application framework. When defined in an application profile, the LSB_DJOB_RTASK_GONE_ACTION variable is set when running **bsub -app** for the specified application. You can also use the environment variable LSB_DJOB_RTASK_GONE_ACTION to override the value set in the application profile.

The RTASK_GONE_ACTION=IGNORE_TASKCRASH parameter has no effect on PE jobs: When a user application is killed, POE triggers the job to quit.

Handling communication failure

By default, LSF shuts down the entire job if connection is lost with the task RES, validation timeout, or heartbeat timeout. You can configure an application profile in lsb.applications so only the current tasks are shut down, not the entire job.

Use the DJOB_COMMFAIL_ACTION="KILL_TASKS" parameter to define the behavior of LSF when it detects a communication failure between itself and one or more tasks. If not defined, LSF terminates all tasks, and shuts down the job. If set to KILL_TASKS, LSF tries to kill all the current tasks of a parallel or distributed job associated with the communication failure.

The DJOB_COMMFAIL_ACTION parameter only applies to the **blaunch** distributed application framework. When defined in an application profile, the LSB_DJOB_COMMFAIL_ACTION environment variable is set when running **bsub -app** for the specified application.

Set up job launching environment

LSF can run an appropriate script that is responsible for setup and cleanup of the job launching environment. You can specify the name of the appropriate script in an application profile in lsb.applications.

Use the DJOB_ENV_SCRIPT parameter to define the path to a script that sets the environment for the parallel or distributed job launcher. The script runs as the user, and is part of the job. The DJOB_ENV_SCRIPT parameter only applies to the **blaunch** distributed application framework. If a full path is specified, LSF uses the path name for the execution. If a full path is not specified, LSF looks for it in LSF_BINDIR.

The specified script must support a setup argument and a cleanup argument. LSF invokes the script with the setup argument before launching the actual job to set up the environment, and with cleanup argument after the job is finished.

LSF assumes that if setup cannot be performed, the environment to run the job does not exist. If the script returns a non-zero value at setup, an error is printed to stderr of the job, and the job exits. Regardless of the return value of the script at cleanup, the real job exit value is used. If the return value of the script is non-zero, an error message is printed to stderr of the job.

When defined in an application profile, the LSB_DJOB_ENV_SCRIPT variable is set when running **bsub -app** for the specified application. For example, if DJOB_ENV_SCRIPT=mpich.script, LSF runs the \$LSF_BINDIR/mpich.script setup script to set up the environment to run an MPICH job. After the job completes, LSF runs the \$LSF_BINDIR/mpich.script script for cleanup.

On cleanup, the mpich.script file could, for example, remove any temporary files and release resources used by the job. Changes to the LSB_DJOB_ENV_SCRIPT environment variable made by the script are visible to the job.

Update job heartbeat and resource usage

Use the DJOB_HB_INTERVAL parameter in an application profile in lsb.applications to configure an interval in seconds used to update the heartbeat between LSF and the tasks of a parallel or distributed job. The DJOB_HB_INTERVAL parameter only applies to the **blaunch** distributed application framework. When the DJOB_HB_INTERVAL parameter is specified, the interval is scaled according to the number of tasks in the job:

```
max(DJOB_HB_INTERVAL, 10) + host_factor
```

where *host_factor* = 0.01 * number of hosts allocated for the job.

When defined in an application profile, the `LSB_DJOB_HB_INTERVAL` variable is set in the parallel or distributed job environment. You should not manually change the value of `LSB_DJOB_HB_INTERVAL`.

By default, the interval is equal to the `SBD_SLEEP_TIME` parameter in the `lsb.params` file, where the default value of `SBD_SLEEP_TIME` is 30 seconds.

How blaunch supports task geometry and process group files

The current support for task geometry in LSF requires the user submitting a job to specify the wanted task geometry by setting the environment variable `LSB_TASK_GEOMETRY` in their submission environment before job submission. LSF checks for `LSB_TASK_GEOMETRY` and modifies `LSB_MCPU_HOSTS` appropriately.

The environment variable `LSB_TASK_GEOMETRY` is checked for all parallel jobs. If `LSB_TASK_GEOMETRY` is set users submit a parallel job (a job that requests more than 1 slot), LSF attempts to shape `LSB_MCPU_HOSTS` accordingly.

The `LSB_TASK_GEOMETRY` variable was introduced to replace the `LSB_PJL_TASK_GEOMETRY` variable, which is kept for compatibility with earlier versions. However, task geometry does not work using **blaunch** alone; it works with the PE/**blaunch** integration.

Resource collection for all commands in a job script

Parallel and distributed jobs are typically launched with a job script. If your job script runs multiple commands, you can ensure that resource usage is collected correctly for all commands in a job script by configuring the `LSF_HPC_EXTENSIONS=CUMULATIVE_RUSAGE` parameter in the `lsf.conf` file. Resource usage is collected for jobs in the job script, rather than being overwritten when each command is executed.

Resizable jobs and blaunch

Because a resizable job can be resized any time, the **blaunch** framework is aware of the newly added resources (hosts) or released resources. When a validation request comes with those additional resources, the **blaunch** framework accepts the request and launches the remote tasks accordingly. When part of an allocation is released, the **blaunch** framework makes sure no remote tasks are running on those released resources, by terminating remote tasks on the released hosts if any. Any further validation requests with those released resources are rejected.

The **blaunch** framework provides the following functionality for resizable jobs:

- The **blaunch** command and `lsb_getalloc()` API call accesses up to date resource allocation through the `LSB_DJOB_HOSTFILE` environment variable
- Validation request (to launch remote tasks) with the additional resources succeeds.
- Validation request (to launch remote tasks) with the released resources fails.
- Remote tasks on the released resources are terminated and the **blaunch** framework terminates tasks on a host when the host has been completely removed from the allocation.
- When releasing resources, LSF allows a configurable grace period (the `DJOB_RESIZE_GRACE_PERIOD` parameter in the `lsb.applications` file) for tasks to clean up and exit themselves. By default, there is no grace period.
- When remote tasks are launched on new additional hosts but the notification command fails, those remote tasks are terminated.

Note: Automatic job resizing is a signaling mechanism only. It does not expand the extent of the original job launched with **blaunch**. The resize notification script is required along with a signal listening script. The signal listening script runs additional **blaunch** commands on notification to allocate the resized resources to make them available to the job tasks. For help creating signal listening and notification scripts, contact IBM Support.

Submitting jobs with blaunch

Use **bsub** to call **blaunch**, or to invoke an execution script that calls **blaunch**. The **blaunch** command assumes that **bsub -n** implies one task per job slot.

- Submit a job:
`bsub -n 4 blaunch myjob`

- Submit a job to launch tasks on a specific host:
bsub -n 4 blaunch hostA myjob
- Submit a job with a host list:
bsub -n 4 blaunch -z "hostA hostB" myjob
- Submit a job with a host file:
bsub -n 4 blaunch -u ./hostfile myjob
- Submit a job to an application profile
bsub -n 4 -app djob blaunch myjob

Launching ANSYS jobs

To launch an ANSYS job through LSF using the **blaunch** framework, substitute the path to rsh or ssh with the path to **blaunch**. For example:

```
#BSUB -o stdout.txt
#BSUB -e stderr.txt
# Note: This case statement should be used to set up any
# environment variables needed to run the different versions
# of Ansys. All versions in this case statement that have the
# string "version list entry" on the same line will appear as
# choices in the Ansys service submission page.

case $VERSION in
  10.0) #version list entry
    export ANSYS_DIR=/usr/share/app/ansys_inc/v100/Ansys
    export ANSYS_LMD_LICENSE_FILE=1051@licserver.company.com
    export MPI_REMSH=/opt/lsf/bin/blaunch
    program=${ANSYS_DIR}/bin/ansys100
    ;;
  *)
    echo "Invalid version ($VERSION) specified"
    exit 1
    ;;
esac

if [ -z "$JOBNAME" ]; then
  export JOBNAME=ANSYS-$$
fi

if [ $CPUS -eq 1 ]; then
  ${program} -p ansys -j $JOBNAME -s read -l en-us -b -i $INPUT $OPTS
else
  if [ $MEMORY_ARCH = "Distributed" ] Then
    HOSTLIST=`echo $LSB_HOSTS | sed s/" /":1:"/g` ${program} -j $JOBNAME - p
  ansys -pp -dis -machines \
    ${HOSTLIST}:1 -i $INPUT $OPTS
  else
    ${program} -j $JOBNAME -p ansys -pp -dis -np $CPUS \
    -i $INPUT $OPTS
  fi
fi
```

blaunch parameters

The **blaunch** application framework uses the following parameters:

- LSF_RES_ALIVE_TIMEOUT
 - LSF_DJOB_TASK_REG_WAIT_TIME
 - LSB_FANOUT_TIMEOUT_PER_LAYER
 - LSB_TASK_GEOMETRY
- This parameter replaces the LSB_PJL_TASK_GEOMETRY parameter.

For details on these parameters, see the *IBM® Spectrum LSF Configuration Reference*.

SGI vendor MPI support

Run your SGI MPI jobs through LSF.

Compiling and linking your MPI program

You must use the SGI C compiler (cc by default). You cannot use **mpicc** to build your programs.

Configuring LSF to work with SGI MPI

To use 32-bit or 64-bit SGI MPI with LSF, set the following parameters in `lsf.conf`:

- Set `LSF_VPLUGIN` to the full path to the MPI library `libxmpi.so`. You can specify multiple paths for `LSF_VPLUGIN`, separated by colons (:). For example, the following configures both `/usr/lib32/libxmpi.so` and `/usr/lib/libxmpi.so`:

```
LSF_VPLUGIN="/usr/lib32/libxmpi.so:/usr/lib/libxmpi.so"
```

For PAM to access the `libxmpi.so` library, the file permission mode must be 755 (-rwxr-xr-x).

To run a multihost MPI applications, you must also enable **rsh** without password prompt between hosts:

- The remote host must be defined in the arrayd configuration.
- Configure `.rhosts` so that `rsh` does not require a password.

Running jobs

To run a job and have LSF select the host, the command **mpirun -np 4 a.out** is entered as:

```
bsub -n 4 pam -mpi -auto_place a.out
```

To run a single-host job and have LSF select the host, the command **mpirun -np 4 a.out** is entered as:

```
bsub -n 4 -R "span[hosts=1]" pam -mpi -auto_place a.out
```

To run a multihost job (5 processors per host) and have LSF select the hosts, the following command:

```
mpirun hosta -np 5 a.out: hostb -np 5 a.out
```

is entered as:

```
bsub -n 10 -R "span[ptile=5]" pam -mpi -auto_place a.out
```

Limitations

- The **mbatchd** and **sbatchd** daemons take a few seconds to get the process IDs and process group IDs of the PAM jobs from the SGI MPI components. If you use **bstop**, **bresume**, or **bkill** before this happens, uncontrolled MPI child processes may be left running.
- A single MPI job cannot run on a heterogeneous architecture. The entire job must run on systems of a single architecture.

Running jobs with task geometry

Specifying task geometry allows you to group tasks of a parallel job step to run together on the same node. Task geometry allows for flexibility in how tasks are grouped for execution on system nodes. You cannot specify the particular nodes that these groups run on; the scheduler decides which nodes run the specified groupings.

Using the task geometry environment variable

Use the `LSB_TASK_GEOMETRY` environment variable to specify task geometry for your jobs. `LSB_TASK_GEOMETRY` replaces `LSB_PJL_TASK_GEOMETRY`, which is kept for compatibility with earlier versions. `LSB_TASK_GEOMETRY` overrides any process group or command file placement options.

The environment variable `LSB_TASK_GEOMETRY` is checked for all parallel jobs. If `LSB_TASK_GEOMETRY` is set users submit a parallel job (a job that requests more than 1 slot), LSF attempts to shape `LSB_MCPU_HOSTS` accordingly.

The `mpirun.lsf` script sets the `LSB_MCPU_HOSTS` environment variable in the job according to the task geometry specification.

The syntax is:

```
setenv LSB_TASK_GEOMETRY "{(task_ID,...) ...}"
```

For example, to submit a job to spawn 8 tasks and span 4 nodes, specify:

```
setenv LSB_TASK_GEOMETRY "{(2,5,7)(0,6)(1,3)(4)}"
```

The results are:

- Tasks 2, 5, and 7 run on one node
- Tasks 0 and 6 run on another node
- Tasks 1 and 3 run on a third node
- Task 4 runs on one node alone

Each `task_ID` number corresponds to a task ID in a job and each set of parenthesis contains the task IDs assigned to one node. Tasks can appear in any order, but the entire range of tasks specified must begin with 0, and must include all task ID numbers; you cannot skip a task ID number. Use braces to enclose the entire task geometry specification, and use parentheses to enclose groups of nodes. Use commas to separate task IDs.

For example:

```
setenv LSB_TASK_GEOMETRY "{(1)(2)}"
```

is incorrect because it does not start from task 0.

```
setenv LSB_TASK_GEOMETRY "{(0)(3)}"
```

is incorrect because it does not specify task 1 and 2.

`LSB_TASK_GEOMETRY` cannot request more hosts than specified by the **`bsub -n`** option. For example:

```
setenv LSB_TASK_GEOMETRY "{(0)(1)(2)}"
```

specifies three nodes, one task per node. A correct job submission must request at least 3 hosts:

```
bsub -n 3 -R "span[ptile=1]" -I -a pe mpirun.lsf my_job
```

```
Job <564> is submitted to queue <hpc_linux>
```

```
<<Waiting for dispatch ...>>
```

```
<<Starting on hostA>>
```

```
...
```

Planning your task geometry specification

You should plan task geometry in advance and specify the job resource requirements for LSF to select hosts appropriately.

Use **`bsub -n`** and **`-R "span[ptile=]"`** to make sure LSF selects appropriate hosts to run the job, so that:

- The correct number of nodes is specified
- All execution hosts have the same number of available slots
- The *ptile* value is the maximum number of CPUs required on one node by task geometry specifications.

LSB_TASK_GEOMETRY only guarantees the geometry but does not guarantee the host order. You must make sure each host selected by LSF can run any group of tasks specified in LSB_TASK_GEOMETRY.

You can also use **bsub -x** to run jobs exclusively on a host. No other jobs share the node once this job is scheduled.

Enforcing resource usage limits for parallel tasks

A typical LSF parallel job launches its tasks across multiple hosts. By default you can enforce limits on the total resources used by all the tasks in the job.

Resource usage limits

Since PAM only reports the sum of parallel task resource usage, LSF does not enforce resource usage limits on individual tasks in a parallel job. For example, resource usage limits cannot control allocated memory of a single task of a parallel job to prevent it from allocating memory and bringing down the entire system. For some jobs, the total resource usage may be exceed a configured resource usage limit even if no single task does, and the job is terminated when it does not need to be.

Attempting to limit individual tasks by setting a system-level swap hard limit (RLIMIT_AS) in the system limit configuration file (/etc/security/limits.conf) is not satisfactory, because it only prevents tasks running on that host from allocating more memory than they should; other tasks in the job can continue to run, with unpredictable results.

By default, custom job controls (the JOB_CONTROL parameter in the lsb.queues file) apply only to the entire job, not individual parallel tasks.

Enabling resource usage limit enforcement for parallel tasks

Use the LSF_HPC_EXTENSIONS options TASK_SWAPLIMIT and TASK_MEMLIMIT in lsf.conf to enable resource usage limit enforcement and job control for parallel tasks. When the TASK_SWAPLIMIT parameter or the TASK_MEMLIMIT parameter is set in the LSF_HPC_EXTENSIONS parameter, LSF terminates the entire parallel job if any single task exceeds the limit setting for memory and swap limits.

Other resource usage limits (CPU limit, process limit, run limit, and so on) continue to be enforced for the entire job, not for individual tasks.

Assumptions and behavior

- To enforce resource usage limits by parallel task, you must use the LSF generic Parallel Job Launcher (PJP) framework (PAM/TS) to launch your parallel jobs.
- This feature only affects parallel jobs monitored by PAM. It has no effect on other LSF jobs.
- The LSF_HPC_EXTENSIONS=TASK_SWAPLIMIT parameter overrides the default behavior of swap limits (**bsub -v**, **bmod -v**, or SWAPLIMIT in lsb.queues).
- The LSF_HPC_EXTENSIONS=TASK_MEMLIMIT parameter overrides the default behavior of memory limits (**bsub -M**, **bmod -M**, or MEMLIMIT in lsb.queues).
- The LSF_HPC_EXTENSIONS=TASK_MEMLIMIT parameter overrides LSB_MEMLIMIT_ENFORCE=Y or LSB_JOB_MEMLIMIT=Y in lsf.conf
- When a parallel job is terminated because of task limit enforcement, LSF sets a value in the LSB_JOBEXIT_INFO environment variable for any post-execution programs:
 - LSB_JOBEXIT_INFO=SIGTERM -29 SIG_TERM_SWAPLIMIT
 - LSB_JOBEXIT_INFO=SIGTERM -25 SIG_TERM_MEMLIMIT
- When a parallel job is terminated because of task limit enforcement, LSF logs the job termination reason in lsb.acct file:
 - TERM_SWAP for swap limit
 - TERM_MEMLIMIT for memory limit**bacct** displays the termination reason.

Running MPI workload through IBM Parallel Environment Runtime Edition

IBM® Spectrum LSF integrates with the IBM Parallel Environment Runtime Edition (IBM PE Runtime Edition) program product - Version 1.3 or later to run PE jobs through the IBM Parallel Operating Environment (POE). The integration enables network-aware scheduling, allowing an LSF job to specify network resource requirements, collect network information, and schedule the job according to the requested network resources.

Note: HPC integration for the IBM Parallel Environment (PE) is now deprecated and might be removed in a future version of LSF.

IBM PE Runtime Edition jobs can be submitted through **bsub**, and monitored and controlled through LSF commands. Network requirements can be specified at job submission with the **bsub -network** option, and configured at the queue (lsb.queues) and application level (lsb.applications) with the NETWORK_REQ parameter.

Important: This integration is based on the LSF **blaunch** framework, which improves performance and reduces the MPI job overhead.

Note: To make this information easier to read, the name *IBM Parallel Environment Runtime Edition* is abbreviated to *IBM PE Runtime Edition*, *Parallel Environment*, or more generally, *PE* throughout the LSF documentation.

Related information

For more information about IBM Parallel Environment Runtime Edition, see the *IBM Parallel Environment: Operation and Use* guide (SC23-6667).

To access the most recent Parallel Environment documentation in PDF and HTML format, refer to the IBM Clusters Information Center:

<http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp>

Both the current Parallel Environment documentation and earlier versions of the library are also available in PDF format on the IBM Publications Center:

www-05.ibm.com/e-business/linkweb/publications/servlet/pbi.wss

- [Enabling IBM PE Runtime Edition for LSF](#)
Complete the following steps to enable the LSF integration with the IBM Parallel Environment Runtime Edition (IBM PE Runtime Edition).
- [Network-aware scheduling](#)
LSF can schedule and launch IBM Parallel Environment (PE) jobs according to the job requirements, IBM Parallel Environment requirements, network availability, and LSF scheduling policies.
- [Submitting IBM Parallel Environment jobs through LSF](#)
- [Managing IBM Parallel Environment jobs through LSF](#)

Enabling IBM PE Runtime Edition for LSF

Complete the following steps to enable the LSF integration with the IBM Parallel Environment Runtime Edition (IBM PE Runtime Edition).

Procedure

1. In `lsf.conf`, set `LSF_PE_NETWORK_NUM`.
Specify a value between 0 and 8 to set the number of InfiniBand networks on the host. If the number is changed, run **lsadmin reconfig** and **badmadmin mbdrestart** to make the change take effect.

LSF_PE_NETWORK_NUM must be defined with a non-zero value in `lsf.conf` for LSF to collect network information to run IBM PE Runtime Edition jobs.

2. Run **hostsetup** or manually set a symbolic link from /usr/lib64/libpermapi.so to \$LSF_LIBDIR/permapi.so.

Network-aware scheduling

LSF can schedule and launch IBM Parallel Environment (PE) jobs according to the job requirements, IBM Parallel Environment requirements, network availability, and LSF scheduling policies.

Network resource collection

To schedule a PE job, LSF must know what network resources are available.

LSF_PE_NETWORK_NUM must be defined with a non-zero value in lsf.conf, LSF collects network information for PE jobs. If LSF_PE_NETWORK_NUM is set to a value greater than zero, two string resources are created:

pe_network

A host-based string resource that contains the network ID and the number of network windows available on the network.

pnsd

Set to Y if the PE network resource daemon **pnsd** responds successfully, or N if there is no response. PE jobs can only run on hosts with **pnsd** installed and running.

Use **lsload -l** to view network information for PE jobs. For example, the following **lsload** command displays network information for `hostA` and `hostB`, both of which have 2 networks available. Each network has 256 windows, and **pnsd** is responsive on both hosts. In this case, LSF_PE_NETWORK_NUM=2 should be set in lsf.conf:

```
lsload -l
HOST_NAME  status  r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem  pnsd
pe_network
hostA      ok     1.0  0.1  0.2  10% 0.0  4  12  1  33G 4041M 2208M Y
ID= 1111111,win=256;ID= 2222222,win=256
hostB      ok     1.0  0.1  0.2  10% 0.0  4  12  1  33G 4041M 2208M Y
ID= 1111111,win=256;ID= 2222222,win=256
```

Specifying network resource requirements

The network resource requirements for PE jobs are specified in the parameter NETWORK_REQ, which can be specified at queue-level in lsb.queues or in an application profile in lsb.applications, and on the **bsub** command with the -network option.

The NETWORK_REQ parameter and the -network option specifies network communication protocols, the adapter device type to use for message passing, network communication system mode, network usage characteristics, and number of network windows (instances) required by the PE job.

`network_res_req` has the following syntax:

```
[type=sn_all | sn_single] [:protocol=protocol_name[(protocol_number)][,protocol_name[(protocol_number)]] [:mode=US | IP]
[:usage=shared | dedicated] [:instance=positive_integer]
```

LSF_PE_NETWORK_NUM must be defined to a non-zero value in lsf.conf for the LSF to recognize the -network option. If LSF_PE_NETWORK_NUM is not defined or is set to 0, the job submission is rejected with a warning message.

The -network option overrides the value of NETWORK_REQ defined in lsb.applications, which overrides the value defined in lsb.queues.

The following IBM LoadLeveler job command file options are not supported in LSF:

- collective_groups
- imm_send_buffers
- rcxtblocks

For detailed information on the supported network resource requirement options, see [Command reference](#) and [configuration reference](#).

Network window reservation

On hosts with IBM PE installed, LSF reserves a specified number of network windows for job tasks. For a job with type=sn_single, LSF reserves windows from one network for each task. LSF ensures that the reserved windows on different hosts are from same network, such that:

$$\text{reserved_window_per_task} = \text{num_protocols} * \text{num_instance}$$

For jobs with type=sn_all, LSF reserve windows from all networks for each task, such that:

$$\text{reserved_window_per_task_per_network} = \text{num_protocols} * \text{num_instance} \text{ where:}$$

- *num_protocols* is the number of communication protocols specified by the protocols of **bsub -network** or NETWORK_REQ (lsb.queues and lsb.applications)
- *num_instance* is the number of instances specified by the instances of **bsub -network** or NETWORK_REQ (lsb.queues and lsb.applications)

Network load balancing

LSF balances network window load. LSF does not to balance network load for jobs with type=sn_all because these jobs request network windows from all networks. Jobs with type=sn_single job request network windows from only one network, so LSF chooses a network with the lowest load, which is typically the network with most total available windows.

Network data striping

When multiple networks are configured in a cluster, a PE job can request striping over the networks by setting type=sn_all in the bsub -network option or the NETWORK_REQ parameter in lsb.queues or lsb.applications. LSF supports the IBM LoadLeveller striping with minimum networks feature, which specifies whether or not nodes which have more than half of their networks in READY state are considered for sn_all jobs. This makes certain that at least one network is UP and in READY state between any two nodes assigned for the job.

Network data striping is enabled in LSF for PE jobs with the STRIPING_WITH_MINIMUM_NETWORK parameter in lsb.params, which tells LSF how to select nodes for sn_all jobs when one or more networks are unavailable. For example, if there are 8 networks connected to a node and STRIPING_WITH_MINIMUM_NETWORK=n, all 8 networks would have to be up and in the READY state to consider that node for sn_all jobs. If STRIPING_WITH_MINIMUM_NETWORK=y, nodes with at least 5 networks up and in the READY state would be considered for sn_all jobs.

In a cluster with 8 networks, due to hardware failure, only 3 networks are ok on hostA, and 5 networks are ok on hostB. If STRIPING_WITH_MINIMUM_NETWORK=n, an sn_all job cannot run on either hostA or hostB. If STRIPING_WITH_MINIMUM_NETWORK=y, an sn_all job can run on hostB, but it cannot run on hostA.

Note: LSF_PE_NETWORK_NUM must be defined with a value greater than 0 for STRIPING_WITH_MINIMUM_NETWORK to take effect.

See the *IBM Parallel Environment: Operation and Use* guide (SC23-6781-05) and the *LoadLeveller Using and Administering* guide (SC23-6792-04) for more information about data striping for PE jobs.

LSF network options, PE environment variables, POE options

The following table shows the LSF network resource requirement options, and their equivalent PE environment variable POE job command file option:

LSF network option	PE Environment variable	POE option
bsub -n	MP_PROCS	-procs
bsub -network "protocol=..."	MP_MSG_API	-msg_api
bsub -network "type=..."	MP_EUIDEVICE	-euidevice
bsub -network "mode=..."	MP_EUILIB	-euilib

LSF network option	PE Environment variable	POE option
bsub -network "instance=..."	MP_INSTANCE	-instances
bsub -network "usage=..."	MP_ADAPTER_USE	-adapter_use

Submitting IBM Parallel Environment jobs through LSF

Use the **bsub -network** option to specify the network resource requirements for an IBM Parallel Environment (PE) job. If any network resource requirement is specified in the job, queue, or application profile, the job is treated as a PE job. PE jobs can only run on hosts where IBM PE **pnsd** daemon is running.

Examples

The following examples assume two hosts in cluster, `hostA` and `hostB`, each with 4 cores and 2 networks. Each network has one IB adapter with 64 windows.

- `bsub -n2 -R "span[ptile=1]" -network "type=sn_single: usage=dedicated" poe /home/user1/mpi_prog`

For this job running on `hostA` and `hostB`, each task will reserve 1 window. The window can be on network 1 of `hostA` and network 1 of `hostB`, or on network 2 of `hostA` and network 2 of `hostB`.

- `bsub -n 2 -network "type=sn_all: usage=dedicated" poe /home/user1/mpi_prog`

For this job running on `hostA`, each task will reserve 2 windows (one window per network). The job totally reserves 4 windows on `hostA`. No other network job can run on `hostA` because all networks are dedicated for use by this job.

- `bsub -n 2 -R "span[ptile=1]" -network "type=sn_all: usage=dedicated]" poe /home/user1/mpi_prog`

For this job running on `hostA` and `hostB`, each task will reserve 2 windows (one window per network). The job reserves 2 windows on `hostA` and two windows on `hostB`. No other network jobs can run on `hostA` and `hostB` because all networks on all hosts are dedicated for use by this job.

- `bsub -n2 -R "span[ptile=1]" -network "protocol=mpi,lapi: type=sn_all: instances=2: usage=shared" poe /home/user1/mpi_prog`

For this job running on `hostA` and `hostB`, each task will reserve 8 windows ($2 \times 2 \times 2$), for 2 protocols, 2 instances and 2 networks. If enough network windows are available, other network jobs with `usage=shared` can run on `hostA` and `hostB` because networks used by this job are shared.

Managing IBM Parallel Environment jobs through LSF

Modifying network scheduling options for Parallel Environment jobs

Use the `bmod -network` option to modify the network scheduling options for submitted IBM Parallel Environment (PE) jobs. The `bmod -networkn` option removes any network scheduling options for the PE job.

You cannot modify the network scheduling options for running jobs, even if `LSB_MOD_ALL_JOBS=y` is defined.

Network resource information (lsload -l)

If `LSF_PE_NETWORK_NUM` is set to a value greater than zero in `lsf.conf`, LSF collects network information for scheduling IBM Parallel Environment (PE) jobs. Two string resources are created for PE jobs:

`pe_network`

A host-based string resource that contains the network ID and the number of network windows available on the network.

pnsd

Set to Y if the PE network resource daemon **pnsd** responds successfully, or N if there is no response. PE jobs can only run on hosts with **pnsd** installed and running.

lsload -l displays the value of these two resources and shows network information for PE jobs. For example, the following **lsload** command displays network information for `hostA` and `hostB`, both of which have 2 networks available. Each network has 256 windows, and **pnsd** is responsive on both hosts. In this case, `LSF_PE_NETWORK_NUM=2` should be set in `lsf.conf`:

```
lsload -l
HOST_NAME  status  r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem  pnsd
pe_network
hostA      ok    1.0   0.1   0.2  10%  0.0  4  12  1  33G 4041M 2208M Y
ID= 1111111,win=256;ID= 2222222,win=256
hostB      ok    1.0   0.1   0.2  10%  0.0  4  12  1  33G 4041M 2208M Y
ID= 1111111,win=256;ID= 2222222,win=256
```

Use **bjobs -l** to displays network resource information for submitted PE jobs. For example:

```
bjobs -l
Job <2106>, User <user1>;, Project <default>;, Status <RUN>;, Queue <normal>, Co
mmand <my_pe_job>
Fri Jun 1 20:44:42: Submitted from host <hostA>, CWD <${HOME}>, Requested Network
<protocol=mpi: mode=US: type=sn_all: instance=1: usage=dedicated>
```

If mode=IP is specified for the PE job, `instance` is not displayed.

Use **bacct -l** to display network resource allocations. For example:

```
bacct -l 210
Job <210>, User <user1>;, Project <default>, Status <DONE>. Queue <normal>,
Command <my_pe_job>
Tue Jul 17 06:10:28: Submitted from host <hostA>, CWD </home/pe_jobs>;
Tue Jul 17 06:10:31: Dispatched to <hostA>, Effective RES_REQ <select[type
== local] order[r15s:pg] rusage[mem=1.00] >, PE Network
ID <1111111> <2222222> used <1> window(s)
per network per task;
Tue Jul 17 06:11:31: Completed <done>.
```

Use **bhist -l** to display historical information about network resource requirements and information about network allocations for PE jobs. For example:

```
bhist -l 749
Job <749>, User <user1>;, Project <default>, Command <my_pe_job>

Mon Jun 4 04:36:12: Submitted from host <hostB>, to Queue <
priority>, CWD <${HOME}>, 2 Processors Requested, Network
<protocols=mpi:mode=US: type=sn_all: instance=1:usage= dedicated>;
Mon Jun 4 04:36:15: Dispatched to 2 Hosts/Processors <hostB>,
Effective RES_REQ <select[ty
pe == local] rusage[nt1=1.00] >, PE Network
ID <1111111> <2222222> used <1> window(s)
per network per task;
Mon Jun 4 04:36:17: Starting (Pid 21006);
```

Use **bhosts -l** to display host-based network resource information for PE jobs. For example:

```
bhosts -l
...
PE NETWORK INFORMATION
NetworkID      Status      rsv_windows/total_windows
1111111        ok          4/64
2222222        closed_Dedicated 4/64
```

`NetworkID` is the physical network ID returned by PE.

`Network Status` is one of the following:

- `ok` - normal status
- `closed_Full` - all network windows are reserved
- `closed_Dedicated` - a dedicated PE job is running on the network (usage=dedicated specified in the network resource requirement string)
- `unavail` - network information is not available

Advance reservation

Advance reservations ensure access to specific hosts or slots during specified times. During the time that an advance reservation is active only users or groups associated with the reservation have access to start new jobs on the reserved hosts or slots.

- [Types of advance reservations](#)
Advance reservations ensure access to specific hosts or slots during specified times. During the time that an advance reservation is active only users or groups associated with the reservation have access to start new jobs on the reserved hosts or slots.
- [Enable advance reservation](#)
To enable advance reservation in your cluster, make sure the advance reservation scheduling plug-in `schmod_advrsv` is configured in the `lsb.modules` file.
- [Allowing users to create advance reservations](#)
By default, only LSF administrators or root can add or delete advance reservations. To allow other users to create and delete advance reservations, configure advance reservation user policies. Use the `ResourceReservation` section of the `lsb.resources` file.
- [Use advance reservation](#)
Use advance reservation commands to add, delete, modify, and view reservations in your system.

Types of advance reservations

Advance reservations ensure access to specific hosts or slots during specified times. During the time that an advance reservation is active only users or groups associated with the reservation have access to start new jobs on the reserved hosts or slots.

Slot-based advance reservations reserve a number of slots among a group of hosts. Host-based advance reservations exclusively reserve a number of hosts, as specified by the user. Each reserved host is reserved in its entirety.

Only LSF administrators or root can create or delete advance reservations. Any LSF user can view existing advance reservations.

Each reservation consists of the number of job slots or hosts to reserve, a list of candidate hosts for the reservation, a start time, an end time, and an owner. You can also specify a resource requirement string instead of or in addition to a list of hosts or slots.

Active reservations

When a reservation becomes active, LSF attempts to run all jobs associated with the reservation. By default, jobs running before the reservation became active continue to run when the reservation becomes active. When a job associated with the reservation is pending, LSF suspends *all* jobs not associated with the reservation that are running on the required hosts.

During the time the reservation is active, only users or groups associated with the reservation have access to start new jobs on the reserved hosts. The reservation is active only within the time frame that is specified, and any given host may have several reservations in place, some of which may be active at the same time.

Jobs are suspended only if advance reservation jobs require the slots or hosts. Jobs using a reservation are subject to all job resource usage limits, but any resources freed by suspending non-advance reservation jobs are available for advance

reservation jobs to use.

Closed and open reservations

Reservations are typically *closed*. When a closed reservation expires, LSF kills jobs running in the reservation and allows any suspended jobs to run when the reservation becomes active.

Open advance reservations allow jobs to run even after the associated reservation expires. A job in the open advance reservation is only treated as an advance reservation job during the reservation window, after which it becomes a normal job. This prevents the job from being killed and makes sure that LSF does not prevent any previously suspended jobs from running or interfering with any existing scheduling policies.

Jobs running in a one-time, open reservation are detached from the reservation and suspended when the reservation expires, allowing them to be scheduled as regular jobs. Jobs submitted before the reservation became active are still suspended when the reservation becomes active. These are only resumed after the open reservation jobs finish.

Jobs running in a closed recurring reservation are killed when the reservation expires.

Jobs running in an open recurring reservation are suspended when the reservation expires, and remain pending until the reservation becomes active again to resume.

If a non-advance reservation job is submitted while the open reservation is active, it remains pending until the reservation expires. Any advance reservation jobs that were suspended and became normal jobs when the reservation expired are resumed first before dispatching the non-advance reservation job submitted while the reservation was active.

System reservations

Reservations can also be created for system maintenance. If a system reservation is active, no other jobs can use the reserved slots or hosts, and LSF does not dispatch jobs to the specified hosts while the reservation is active.

Dynamically scheduled reservations

A *dynamically scheduled* reservation accepts jobs based on currently available resources. Use the **brsvsub** command to create a dynamically scheduled reservation and submit a job to fill the advance reservation when the resources required by the job are available.

Jobs that are scheduled for the reservation run when the reservation is active. Because they are scheduled like jobs, dynamically scheduled reservations do not interfere with running workload (unlike normal advance reservations, which kill any running jobs when the reservation window opens.)

Jobs in a dynamically scheduled reservation remain pending until resources are available and the advance reservation becomes active. The reservation can request whole nodes if necessary.

Instead of starting at a predefined time, jobs start whenever the resources for reservation are available. You can also query whether a scheduled reservation can actually be fulfilled.

Enable advance reservation

To enable advance reservation in your cluster, make sure the advance reservation scheduling plug-in `schmod_advrsv` is configured in the `lsb.modules` file.

```
Begin PluginModule
SCH_PLUGIN          RB_PLUGIN          SCH_DISABLE_PHASES
schmod_default      ()                  ()
schmod_advrsv       ()                  ()
End PluginModule
```

Allowing users to create advance reservations

By default, only LSF administrators or root can add or delete advance reservations. To allow other users to create and delete advance reservations, configure advance reservation user policies. Use the `ResourceReservation` section of the `lsb.resources` file.

About this task

Note: The `USER_ADVANCE_RESERVATION` in the `lsb.params` file is obsolete from LSF Version 9 and later.

Procedure

Use the `ResourceReservation` section of the `lsb.resources` file to configure advance reservation policies for users in your cluster.

A `ResourceReservation` section contains the following information:

- Users or user groups that can create reservations
- Hosts that can be used for the reservation
- Time window when reservations can be created.

Each advance reservation policy is defined in a separate `ResourceReservation` section, so it is normal to have multiple `ResourceReservation` sections in a `lsb.resources` file.

In the following policy, only `user1` and `user2` can make advance reservations on `hostA` and `hostB`. The reservation time window is between 8:00 a.m. and 6:00 p.m. every day:

```
Begin ResourceReservation
NAME          = dayPolicy
USERS         = user1 user2    # optional
HOSTS         = hostA hostB    # optional
TIME_WINDOW   = 8:00-18:00     # weekly recurring reservation
End ResourceReservation
```

`user1` can add the following reservation for `user2` to use on `hostA` every Friday between 9:00 a.m. and 11:00 a.m.:

```
brsvadd -m "hostA" -n 1 -u "user2" -t "5:9:0-5:11:0"
Reservation "user1#2" is created
```

Users can only delete reservations that they created themselves. In the example, only user `user1` can delete the reservation; `user2` cannot. Administrators can delete any reservations that are created by users.

In the following policy, all users in user group `ugroup1` except `user1` can make advance reservations on any host in `hgroup1`, except `hostB`, between 10:00 p.m. and 6:00 a.m. every day.

```
Begin ResourceReservation
NAME          = nightPolicy
USERS         = ugroup1 ~user1
HOSTS         = hgroup1 ~hostB
TIME_WINDOW   = 20:00-8:00
End ResourceReservation
```

Important:

The not operator (`~`) does not exclude LSF administrators from the policy.

Example

1. Define a policy for `user1`:

```
Policy Name: dayPolicy
Users: user1
Hosts: hostA
Time Window: 8:00-18:00
```

2. `user1` creates a reservation matching the policy (the creator is `user1`, the user is `user2`):

```
brsvadd -n 1 -m hostA -u user2 -b 10:00 -e 12:00
user1#0 is created.
```

3. User `user1` modifies the policy to remove `user1` from the users list:

```
Policy Name: dayPolicy
Users: user3
Hosts: hostA
Time Window: 8:00-18:00
```

4. As the creator, `user1` can modify the reservation with the **brsvmod** command options `rmhost`, `-u`, `-o`, `-on`, `-d`, and the subcommands **adduser** and **rmuser**. However, `user1` cannot add hosts or change the time window of the reservation.

Use advance reservation

Use advance reservation commands to add, delete, modify, and view reservations in your system.

Use the following commands with advance reservations:

```
brsvadd
    Add a reservation.
brsvdel
    Delete a reservation.
brsvmod
    Modify a reservation.
brsvs
    View reservations.
```

Job scheduling in advance reservations

LSF treats advance reservation like other deadlines, such as dispatch windows or run windows. LSF does not schedule jobs that are likely to be suspended when a reservation becomes active. Jobs that are running in the reservation are killed when the reservation expires.

When the total number of slots on the reserved host is changed for whatever reason, LSF immediately updates the host reservation to reserve the new total number of slots or CPUs. The total number of slots change under the following conditions:

- Host status becomes **UNAVAIL**. LSF sets the number of slots to 1 because LSF cannot detect the correct information.
- The **MXJ** configuration in the `lsb.hosts` file changes
- A host is updated with the **bconf** command.
- The **SLOTS_PER_PROCESSOR** parameter in the `lsb.resources` file changes
- The **SLOTS** parameter in the `lsb.resources` file changes

Note:

If the **IGNORE_DEADLINE=Y** parameter is specified, advance reservations are not affected. Jobs are always prevented from starting if they might encounter an advance reservation.

Reservation policy checking

The following table summarizes how advance reservation commands interpret reservation policy configurations in the `lsb.resources` file:

Command		Policy checked		
		Creator	Host	Time window
brsvadd		Yes	Yes	Yes
brsvdel		No	No	No
brsvmod	<code>-u</code> (changing user and user groups)	No	No	No

Command		Policy checked		
		Creator	Host	Time window
	adduser	No	No	No
	rmuser	No	No	No
	addhost	Yes	Yes	Yes
	rmhost	No	No	No
	-b, -e, -t (change time window)	Yes	Yes	Yes
	-d (description)	No	No	No
	-o or -on	No	No	No

Reservation policies are checked at the following times:

- The reservation time window is changed.
- Hosts are added to the reservation.

Reservation policies are *not* checked under the following conditions:

- Running **brsvmod** to remove hosts
- Changing the reservation type (open or closed)
- Changing users or user groups for the reservation
- Modifying the reservation description
- Adding or removing users and user groups to or from the reservation
- [Adding an advance reservation](#)
Use the **brsvadd** command to create new advance reservations.
- [Changing an advance reservation](#)
Use the **brsvmod** command to change an existing advance reservation.
- [Removing an advance reservation](#)
Use the **brsvdel** command to delete reservations.
- [Viewing advance reservation information](#)
Use the **brsvs** command to view information about advance reservations. You can see all current reservations, show a weekly planner for your reservations, or see reservation types and their associated jobs. Use the **bjobs** command to see the reservation ID for an advance reservation job. Use the **bacct** command to view historical accounting information for advance reservations.
- [Submitting and modifying jobs that use advance reservations](#)
The -U option of the **bsub** command submits jobs with a reservation ID.
- [Viewing jobs that are associated with an advance reservation](#)
- [Advance reservation behavior and operations](#)
A job that uses a reservation is subject to all job resource usage limits. Advance reservation preemption allows advance reservation jobs to use the slots that are reserved by the reservation. You can create and use advance reservations for the LSF multicluster capability job forwarding model. Resizable jobs and jobs with compute unit resource requirements can be dispatched only after the advance reservation becomes active.

Adding an advance reservation

Use the **brsvadd** command to create new advance reservations.

About this task

Note: By default, only LSF administrators or root can add or delete advance reservations.

Procedure

On the **brsvadd** command, specify the following properties of the reservation.

- Number of job slots or hosts to reserve. This number must be less than or equal to the actual number of slots or hosts, which are selected by the -m or -R option.
- The unit (slots or hosts) to use for the reservation. By default (without the -unit option), the **brsvadd** command creates a slot-based reservation. Create a host-based reservation by specifying the -unit host option, or a slot-based reservation with -unit slot.
- Hosts for the reservation
- Owners of the reservation
- Time period for the reservation. Specify one of the following time periods:
 - Begin time and end time for a one-time reservation.
 - Time window for a recurring reservation.

Note: Advance reservations must be 10 minutes or more in length.

If the reservations overlap other advance reservations that begin or end within a 10-minute time period, they might be rejected.

A day is divided into 144 periods, and each period lasts for 10 minutes (0:0-0:10, 0:10-0:20, up to 23:50-24:00). If the start time or end time of a reservation is in the middle of a time period, LSF reserves the entire period. For example, if one reservation begins at 1:22 and ends at 4:24, a reservation request that starts at 4:25 is rejected because it lies within the already reserved 4:20-4:30 time period.

- For placeholder reservations, the user or user group name that uses the reservation is required. Optionally, you can specify a name and description for the reservation.

The **brsvadd** command returns a reservation ID that you use when you submit a job that needs the reserved hosts. Any single user or user group can have a maximum of 100 reservation IDs.

Specifying hosts for the reservation

Procedure

Use one or both of the following **brsvadd** command options to specify hosts that job slots are reserved for:

- The -m option lists the hosts that are needed for the reservation. The hosts that are listed by the -m option can be local to the cluster or hosts that are leased from remote clusters. At job submission, LSF considers the hosts in the specified order. If you also specify a resource requirement string with the -R option, the -m flag is optional.
- The -R option selects hosts for the reservation according to a resource requirements string. Only hosts that satisfy the resource requirement expression are reserved. The -R option accepts any valid resource requirement string, but only the `select` and `same` strings take effect. If you also specify a host list with the -m option, the -R flag is optional. Resource requirements are checked before jobs are forwarded to the remote cluster. If the selection string is valid, the job is forwarded. The strict resource requirement syntax check applies only to the `select` section. It does not apply to the other resource requirement sections (`order`, `rusage`, `same`, `span`, or `cu`).

For more information about strict syntax for resource requirement selection strings, see [Selection string](#).

Adding a one-time reservation

Procedure

Use the -b and -e options of the **brsvadd** command to specify the begin time and end time of a one-time advance reservation. One-time reservations are useful for dedicating hosts to a specific user or group for critical projects. The day and time are in the following form:

```
[ [ [year:] month:] day:] hour:minute
```

The begin and end times have the following ranges:

year

Any year after 1900 (YYYY).

month

1-12 (MM).

day of the month

1-31 (dd).

hour

0-23 (hh).
minute
0-59 (mm).

You must specify at least *hour:minute*. Year, month, and day are optional. Three fields are assumed to be *day:hour:minute*. Four fields are assumed to be *month:day:hour:minute*. Five fields are *year:month:day:hour:minute*.

If you do not specify a day, LSF assumes the current day. If you do not specify a month, LSF assumes the current month. If you specify a year, you must specify a month.

You must specify a begin and an end time. The time value for *-b* must use the same syntax as the time value for *-e*. The begin time must be earlier than the time value for *-e*. The begin time cannot be earlier than the current time.

The following command creates a one-time advance reservation for 1024 job slots on host *hostA* for user *user1* between 6:00 AM and 8:00 AM today:

```
brsvadd -n 1024 -m hostA -u user1 -b 6:0 -e 8:0  
Reservation "user1#0" is created
```

The hosts that you specify with the *-m* option can be local to the cluster or hosts that are leased from remote clusters. The following command creates a one-time advance reservation for 1024 job slots on a host of any type for user *user1* between 6:00 AM and 8:00 AM today:

```
brsvadd -n 1024 -R "type==any" -u user1 -b 6:0 -e 8:0  
Reservation "user1#1" is created
```

The following command creates a one-time advance reservation that reserves 12 slots on *hostA* between 6:00 PM on 01 December 2013 and 6:00 AM on 31 January 2014:

```
brsvadd -n 12 -m hostA -u user1 -b 2013:12:01:18:00 -e 2014:01:31:06:00  
Reservation user1#2 is created
```

Adding a recurring reservation

Procedure

Use the *-t* option of the **brsvadd** command to specify a recurring advance reservation.

The *-t* option specifies a time window for the reservation. Recurring reservations are useful for scheduling regular system maintenance jobs. The day and time are in the following form:

[*day*:]*hour*[:*minute*]

The day and time has the following ranges:

day of the week

0-23.

hour

0-6.

minute

0-59.

Specify a time window one of the following ways:

- *hour-hour*
- *hour:minute-hour:minute*
- *day:hour:minute-day:hour:minute*

You must specify at least the hour. Days of the week and minute are optional. Both the start time and end time values must use the same syntax. If you do not specify a minute, LSF assumes the first minute of the hour (:00). If you do not specify a day, LSF assumes every day of the week. If you do specify the day, you must also specify the minute.

If the current time when the reservation was created is within the time window of the reservation, the reservation becomes active immediately.

When the job starts running, the termination time of the advance reservation job is determined by the minimum of the job run limit (if specified), the queue run limit (if specified), or the duration of the reservation time window.

The following command creates an advance reservation for 1024 job slots on two hosts `hostA` and `hostB` for user group `groupA` every Wednesday from 12:00 midnight to 3:00 AM:

```
brsvadd -n 1024 -m "hostA hostB" -g groupA -t "3:0:0-3:3:0"  
Reservation "groupA#0" is created
```

The following command creates an advance reservation for 1024 job slots on `hostA` for user `user2` every weekday from 12:00 noon to 2:00 PM:

```
brsvadd -n 1024 -m "hostA" -u user2 -t "12:0-14:0"  
Reservation "user2#0" is created
```

The following command creates a system reservation on `hostA` every Friday from 6:00 PM to 8:00 PM:

```
brsvadd -n 1024 -m hostA -s -t "5:18:0-5:20:0"  
Reservation "system#0" is created
```

While the system reservation is active, no other jobs can use the reserved hosts, and LSF does not dispatch jobs to the specified hosts.

The following command creates an advance reservation for 1024 job slots on hosts `hostA` and `hostB` with more than 50 MB of swap space for user `user2` every weekday from 12:00 noon to 2:00 PM:

```
brsvadd -n 1024 -R "swp > 50" -m "hostA hostB" -u user2 -t "12:0-14:0"  
Reservation "user2#1" is created
```

Adding an open reservation

Procedure

Use the `-o` option of the **brsvadd** command to create an open advance reservation. Specify the same information as for normal advance reservations.

The following command creates a one-time open advance reservation for 1024 job slots on a host of any type for user `user1` between 6:00 AM and 8:00 AM today:

```
brsvadd -o -n 1024 -R "type==any" -u user1 -b 6:0 -e 8:0  
Reservation "user1#1" is created
```

The following command creates an open advance reservation for 1024 job slots on `hostB` for user `user3` every weekday from 12:00 noon to 2:00 PM:

```
brsvadd -o -n 1024 -m "hostB" -u user3 -t "12:0-14:0"  
Reservation "user2#0" is created
```

Specifying a reservation name

Procedure

Use the `-N` option of the **brsvadd** command to specify a user-defined advance reservation name unique in an LSF cluster.

The reservation name is a string of letters, numbers, underscores, and dashes. The name must begin with a letter. The maximum length of the name is 39 characters.

If no user-defined advance reservation name is specified, LSF creates the reservation with a system assigned name with the form

`creator_name#sequence`

```
brsvadd -n 3 -M "hostA hostB" -u user2 -b 16:0 -e 17:0 -d "Production AR test"  
Reservation user2#0 (Production AR test) is created
```

```
brsvadd -n 2 -N Production_AR -M hostA -u user2 -b 16:0 -e 17:0 -d "Production AR test"  
Reservation Production_AR (Production AR test) is created
```

If a job exists that references a reservation with the specified name, an error message is returned: The specified reservation name is referenced by a job.

Adding a reservation placeholder

Use a reservation *placeholder* to dynamically create and schedule an advance reservation in the same way as a job. Jobs that are scheduled for that reservation run within when the reservation is active.

Procedure

1. Use the **brsvadd -p** command to create a reservation placeholder.

Note: You must use the **-u** to define a user name or user group that uses the reservation.

```
brsvadd -p -u user1
Reservation user1#0 is created
```

2. Use the **brsvsub** command to create a schedulable advance reservation, and submit a job to the reservation. The **brsvsub** command specifies the properties of the reservation that is to be scheduled.

The following command fills placeholder reservation **user1#19**. The reservation has a duration of 10 minutes and allocates 2 hosts for user **user1**.

```
brsvsub -D 10 -n 2 -unit host -u user1
Placeholder reservation user1#19 is being scheduled by job <28> in the default queue
<normal>.
```

3. Submit a job to the scheduled advance reservation.

```
bsub -U user1#19 sleep 100
```

Results

Use the **brsvs** command to query the scheduled reservation.

```
brsvs -l user1#19
RSVID      TYPE      USER      NCPUS      RSV_HOSTS      TIME_WINDOW
user1#19    user      haoqing    0/16        lsfrhel02:0/8   4/20/19/3-4/20/19/13
lsfrhel04:0/8
Reservation Status: Active
Description: job <28>
Creator: user1
Reservation Type: CLOSED
Resource Unit: Host
```

Use the **brsvjob** command to see information about jobs submitted with the **brsvsub** command.

```
brsvjob user1#19
```

```
Job <28>, User <user1>, Project <default>, Status <DONE>, Queue <normal>, Comm
and <lsfrsv -N user1#19 -D 10 -n 2>, Share group charged
</user1>, Job Description <user1#19>
Tue Jun  6 21:47:58: Submitted from host <hostA>, CWD
</scratch/dev/user1/lsf>, 2 Task(s);

RUNLIMIT
11.0 min of hostA
Tue Jun  6 21:47:58: Started 2 Task(s) on Host(s) <hostA>
<hostB>, Allocated 2 Slot(s) on Host(s)
<hostA> <hostB>, Execution Home </home/user1>, Ex
ecution CWD </scratch/dev/user1/lsf>;
Tue Jun  6 21:47:58: Done successfully. The CPU time used is 0.1 seconds.
```

SCHEDULING PARAMETERS:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

RESOURCE REQUIREMENT DETAILS:

```
Combined: select[type == local] order[r15s:pg]
Effective: select[type == local] order[r15s:pg]
```

Allowing non-reservation jobs to run on reservation hosts

By default, LSF does not allow a job to start and run on hosts that belong to any advance reservations if the job might still run when the advance reservation becomes active. You can override this behavior by using the **brsvadd -q** option to allow jobs from the specified queue to be allowed to run on these advance reservation hosts. LSF suspends these jobs when the first advance reservation job starts unless you also use the **brsvadd -nosusp** option.

Procedure

1. Use the **-q** option of the **brsvadd** command to specify queues whose jobs can start on advance reservation hosts before the advance reservation is active.

```
brsvadd -q queue_name
```

```
brsvadd -q "queue_name ..."
```

When you specify multiple queues, use a space to separate multiple queues and quotation marks to enclose the list.

```
brsvadd -N AR1 -n 1 -unit host -m hostA -u user1 -q normal -b 14:00 -e 16:00
Reservation AR1 is created
```

Jobs that are submitted to the `normal` queue are able to start on the `hostA` host until the advance reservation becomes active at 2:00 PM.

If you specify a pre-time for the advance reservation (**-Et** option), LSF stops dispatching jobs from the specified queue when the pre-time is reached, otherwise, LSF stops dispatching jobs from the specified queue when the advance reservation becomes active.

2. Use the **-nosusp** option of the **brsvadd** command to enable non-advance reservation jobs to continue running on advance reservation hosts when the first advance reservation job starts. These jobs are not suspended when the advance reservation is active, and advance reservation jobs do not start until the required resources are available.

The **-nosusp** option is only available for user advance reservations. System advance reservations are not supported.

```
brsvadd -N AR2 -n 1 -unit host -m hostB -u user1 -q nosuspend -nosusp -b 14:00 -e 16:00
Reservation AR2 is created
```

Jobs that are submitted to the `nosuspend` queue are able to start on the `hostB` host until the advance reservation becomes active at 2:00 PM. In addition, these jobs are not suspended if they are still running when the first advance reservation job starts after 2:00 PM.

What to do next

Use the **brsvs -l** command to see which queues can use the advance reservation hosts before the advance reservation is active.

```
brsvs -l
RSVID      TYPE      USER      NCPUS      RSV_HOSTS      TIME_WINDOW
AR1         user      user1      0/8         hostA:0/8       14-16
Reservation Status: Active
Creator: user2
Queues that can use AR hosts before AR starts: normal
Reservation Type: CLOSED
Resource Unit: Host
RSVID      TYPE      USER      NCPUS      RSV_HOSTS      TIME_WINDOW
AR2         user      user1      0/8         hostB:0/8       14-16
Reservation Status: Active
Creator: user1
Queues that can use AR hosts before AR starts: nosuspend
Reservation Type: CLOSED
Resource Unit: Host
Start action: Allow non-AR jobs to continue running
```

Specifying scripts to run at the start or end of the reservation

Specify scripts to run at the start of an advance reservation (pre-script), or when the advance reservation expires (post-script).

Before you begin

To specify scripts to run with an advance reservation, you must first specify `LSB_START_EBROKERD=Y` in the `lsf.conf` file.

Procedure

1. Use the `-E` option of the **brsvadd** command to specify the absolute file path to a script that is run to create the advance reservation (the pre-script).

If the creator is not root or an LSF administrator, the creator's user group must be an LSF or queue administrator so that this pre-script can act on other users' jobs. The file path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.

The following environment variables are available for use in the script:

`AR_NAME`

Name of the advance reservation.

`AR_QUEUE_LIST`

List of queues whose jobs can be run in this advance reservation.

`AR_HOST_LIST`

List of hosts in this advance reservation. The host is reported even if the advance reservation does not use all slots on the host.

`AR_START_TIME`

Start time of this advance reservation in epoch seconds.

`AR_END_TIME`

End time of this advance reservation in epoch seconds.

`AR_JOBIDS`

The job IDs of jobs that are currently running on this advance reservation's hosts.

`AR_CREATOR`

Name of the user that created this advance reservation.

`AR_OWNERS`

Name of the owners of this advance reservation.

The script is run at the start time of the advance reservation unless a pre-time is set with the `-Et` option, then the script is run at the start time minus the specified pre-time. If the script is modified before the script is to be run, the latest version of the script is run at the start time of the script.

LSF does not take any specific action based on the success or failure of the script, and there is no timeout period or action that is associated with this script.

2. Use the `-Et` option of the **brsvadd** command to specify the amount of time, in minutes, before the start of the advance reservation for LSF to run this script and to stop dispatching new jobs to the advance reservation hosts.

If this option is specified without the `-E` option, LSF stops dispatching jobs to this advance reservation's hosts at the specified time without running a pre-script.

3. Use the `-Ep` option of the **brsvadd** command to specify the absolute file path to a script that is run as the creator of the advance reservation when the advance reservation expires (the post-script).

If the creator is not root or an LSF administrator, the creator's user group must be an LSF or queue administrator so that this post-script can act on other users' jobs. The file path can contain up to 4094 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.

The environment variables that are available for use in the post-script are the same as the ones that are available for the pre-script (`-E` option).

The script is run at the expiry time of the advance reservation unless a post-script time is set with the `-Ept` option, then the script is run at the expiry time minus the specified post-script time. If the script is modified before the script is to be run, the latest version of the script is run at the start time of the script.

LSF does not take any specific action based on the success or failure of the script, and there is no timeout period or action that is associated with this script.

4. Use the `-Ept` option of the **brsvadd** command to specify the amount of time, in minutes, before the expiry of the advance reservation for LSF to run this script.

This option is ignored if it is specified without the `-Ep` option.

Example

For example, to create an advance reservation that uses a pre-script that starts 5 minutes before the advance reservation starts, and a post-script that starts 10 minutes before the advance reservation ends:

```
brsvadd -N AR1 -n 1 -unit host -m hostA -u user1 -E /home/user1/pre.sh -Et 5 -Ep /home/user1/post.sh -Ept 10 -q normal -nosusp -b 9:00 -e 17:00
Reservation AR1 is created
```

What to do next

Use the **brsvs -l** command to show the pre-script, pre-time, post-script, and post-script time settings.

```
brsvs -l
RSVID      TYPE      USER      NCPUS      RSV_HOSTS      TIME_WINDOW
AR1        user      user1      0/8        hostA:0/8      9-17
Reservation Status: Active
Creator: user1
Queues that can use AR hosts before AR starts: normal
Pre-AR Script: /home/user1/pre.sh
Pre-AR Time: 5 minutes before AR becomes active
Post-AR Script: /home/user1/post.sh
Post-AR Time: 10 minutes before AR end time
Reservation Type: CLOSED
Resource Unit: Host
Start action: Allow non-AR jobs to continue running
```

Changing an advance reservation

Use the **brsvmod** command to change an existing advance reservation.

Specify the reservation ID for the reservation you want to modify. For example, run the following command to extend the duration from 6:00 AM. to 9:00 AM.

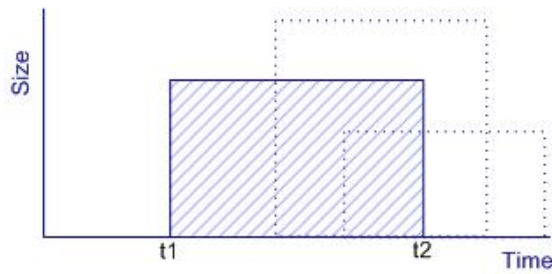
```
brsvmod -e "+60" user1#0
Reservation "user1#0" is modified
```

Note: Administrators and root can modify any reservations. Users listed in the `ResourceReservation` section of the `lsb.resources` file, can modify only reservations they created themselves.

Take the following actions on a reservation:

- Modify start time (postpone or move closer)
- Modify the duration of the reservation window (and thus the end time)
- Modify the slot numbers required by the reservation (add or remove slots with hosts)
- Modify the host or host group list (add or remove hosts or host groups)
- Replace the user or user group list or add or remove users or user groups
- Add hosts by resource requirement (-R)
- Modify the reservation type (open or closed)
- Disable the specified occurrences of a recurring reservation
- Modify the queue whose jobs are allowed to run on the advance reservation hosts before the advance reservation becomes active
- Modify whether LSF suspends non-advance reservation jobs that are running when the advance reservation becomes active
- Modify pre-scripts, post-scripts, and script start times.

For example, assume an advance reservation is the box between the time t_1 and t_2 , as shown in the following figure:



- The shadowed box shows the original reservation
- Time means the time window of the reservation
- t_1 is the begin time of the reservation
- t_2 is the end time of the reservation
- The reservation size means the resources that are reserved, such as hosts (slots) or host groups

Use the **brsvmod** command to shift, extend, or reduce the time window horizontally, and to grow or shrink the size vertically.

Extend the duration

The following command creates a one-time advance reservation for 1024 job slots on host `hostA` for user `user1` between 6:00 AM and 8:00 AM today:

```
brsvadd -n 1024 -m hostA -u user1 -b "6:0" -e "8:0"
Reservation "user1#0" is created
```

Run the following command to extend the duration from 6:00 AM to 9:00 AM:

```
brsvmod -e "+60" user1#0
Reservation "user1#0" is modified
```

Add hosts to a reservation allocation

Use the **brsvmod** command to add hosts and slots on hosts into the original advance reservation allocation. The hosts can be local to the cluster or hosts leased from remote clusters.

Slots cannot be added (-n option) to a system reservation. Only hosts can be added (-m option) to a system reservation.

Adding a host without the -n option reserves all available hosts or slots on the host that are not already reserved by other reservations. You can specify the number of slots to be added from the host list specified with the -n option, but the -n option cannot be used alone. The -m option can be used alone if no host group is specified in the list. You must specify the -R option together with the -n option.

The specified number of units (slots or hosts) must be less than or equal to the available number of slots for the hosts or hosts themselves.

The following examples reserve slots from a reservation with the **brsvmod addhost** command:

- Reserve 2 more slots from `hostA`:

```
brsvmod addhost -n2 -m "hostA"
```
- Reserve 4 slots in total from `hostA` and `hostB`:

```
brsvmod addhost -n4 -m "hostA hostB"
```
- Reserve 4 more slots from any Linux hosts:

```
brsvmod addhost -n4 -R "type==linux"
```
- Reserve 4 more slots from any Linux hosts in the host group `hostgroup1`:

```
brsvmod addhost -n4 -m "hostgroup1" -R "type==linux"
```
- Dynamically add any Linux hosts in the `hostgroup1` host group:

```
brsvmod addhost -f -m "hostgroup1" -R "type==linux"
```

- Reserve all available slots from `hostA` and `hostB`:

```
brsvmod addhost -m "hostA hostB"
```

The following command creates an advance reservation for 1024 slots on two hosts `hostA` and `hostB` for user group `groupA` every Wednesday from 12:00 midnight to 3:00 AM:

```
brsvadd -n 1024 -m "hostA hostB" -g groupA -t "3:0:0-3:3:0"
Reservation "groupA#0" is created
```

brsvs

RSVID	TYPE	USER	NCPUS	RSV_HOSTS	TIME WINDOW
groupA#0	user	groupA	0/1024	hostA:0/256 hostB:0/768	3:3:0-3:3:0 *

The following commands reserve 256 slots from each host for the reservation:

```
brsvmod addhost -n 256 -m "hostA" groupA#0
Reservation "groupA#0" is modified
brsvmod rmhost -n 256 -m "hostB" groupA#0
Reservation "groupA#0" is modified
```

Remove hosts from a reservation allocation

Use the **brsvmod rmhost** command to remove hosts or slots on hosts from the original reservation allocation. You must specify either the `-n` or `-m` option. Use the `-n` option to specify the number of slots to be released from the host. Removing a host without the `-n` option releases all reserved slots on the host. The slot specification must be less than or equal to the actual reserved slot number of the host.

The following examples remove slots from reservations with the **brsvmod rmhost** command:

- Remove 4 reserved slots from `hostA`

```
brsvmod rmhost -n 4 -m "hostA"
```

- Remove 4 slots in total from `hostA` and `hostB`.

```
brsvmod rmhost -n 4 -m "hostA hostB"
```

- Release reserved `hostA` and `hostB`

```
brsvmod rmhost -m "hostA hostB"
```

- Remove 4 slots from current reservation allocation.

```
brsvmod rmhost -n 4
```

You cannot remove slots from a system reservation. The following modification to the system reservation `System#1` is rejected:

```
brsvmod rmhost -n 2 -m "hostA" system#1
```

The number of slots or hosts that can be removed also depends on the number of slots that are free while the reservation is active. The **brsvmod rmhost** command cannot remove more slots than are free on a host. For example:

brsvs

RSVID	TYPE	USER	NCPUS	RSV_HOSTS	TIME WINDOW
user1_1	user	user1	3/4	hostA:2/2 hostB:1/2	1/24/12/2-1/24/13/0

The following modifications are accepted because `hostB` has free slots:

```
brsvmod rmhost -m hostB user1_1
brsvmod rmhost -n 1 -m hostB user1_1
```

The following modifications are rejected because no free slots are available to be removed from `hostA`:

```
brsvmod rmhost -n 2 user1_1 # <-- only 1 slot is free
brsvmod rmhost -m hostA user1_1 # <-- hostA has no free slots
brsvmod rmhost -n 1 -m hostA user1_1 # <-- hostA has no free slots
brsvmod rmhost -n 2 -m hostB user1_1 # <-- hostB only has 1 free slot
```

Modify closed reservations

The following command creates an open advance reservation for 1024 job slots on host `hostA` for user `user1` between 6:00 AM and 8:00 AM today.

```
brsvadd -o -n 1024 -m hostA -u user1 -b 6:0 -e 8:0
Reservation "user1#0" is created
```

Run the following command to close the reservation when it expires.

```
brsvmod -on user1#0
Reservation "user1#0" is modified
```

Modify a reservation placeholder

To add a time window to a reservation placeholder, use the `brsvmod -b begin_time -e end_time reservation_ID` command.

```
brsvmod -b 23:30 -e 23:40 user1#0
Reservation user1#0 is modified
```

- By default, a placeholder reservation is a one-time reservation. You can't change a placeholder to a recurring reservation.
- A placeholder reservation with a time window is cleaned when the reservation expires.

To add resources to a placeholder, use the **`brsvmod addhost`** command.

```
brsvmod addhost -m lsfrhel04 -n 2 user1#0
Reservation user1#0 is modified
```

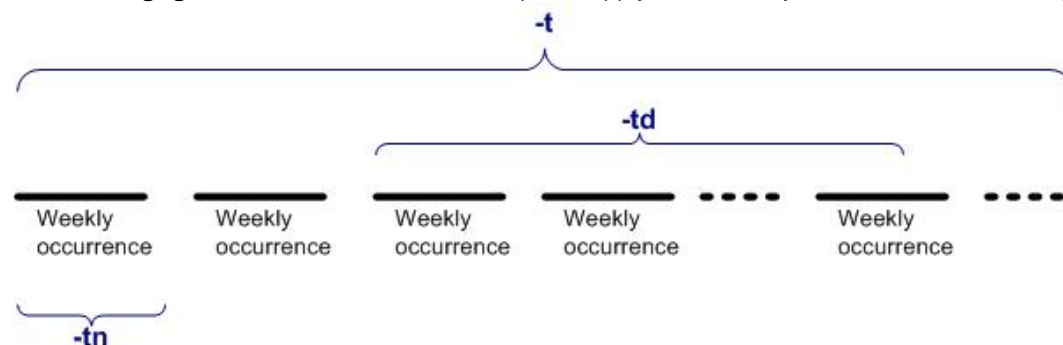
Disable specified occurrences for recurring reservations

Use the **`brsvmod disable`** command to disable specified periods, or *instances*, of a recurring advance reservation.

Recurring reservations may repeat either on a daily cycle or a weekly cycle. For daily reservations, the instances of the reservation that occur on disabled days will be inactive. Jobs using the reservation are not dispatched on those disabled days. Other reservations are permitted to use slots of the reservation on those days. For overnight reservations (active from 11:00 PM to 9:00 AM daily), if the reservation is disabled on the starting day of an instance, the reservation is disabled for the whole of that instance.

For a weekly reservation, if the reservation is disabled on the start date of an instance of the reservation then the reservation is disabled for the entire instance. For example, for a weekly reservation with time window from 9 AM Wednesday to 10 PM Friday, in one particular week, the reservation is disabled on Thursday, then the instance of the reservation remains active for that week. However, if the same reservation is disabled for the Wednesday of the week, then the reservation is disabled for the week.

The following figure illustrates how the disable options apply to the weekly occurrences of a recurring advance reservation.



Once a reservation is disabled for a period, it cannot be enabled again; that is, the disabled periods remain fixed. Before a reservation is disabled, you are prompted to confirm whether to continue disabling the reservation. Use the **-f** option to silently force the command to run without prompting for confirmation, for example, to allow for automating disabling reservations from a script.

For example, the following command creates a recurring advance reservation for 4 slots on host `hostA` for user `user1` between 6:00 AM and 8:00 AM every day.

```
Reservation "user1#0" is created
brsvadd -n 4 -m hostA -u user1 -t "6:0-8:0"
```

Run the following command to disable the reservation instance that is active between Dec 1 to Dec 10, 2007.

```
brsvmod -disable -td "2007:12:1-2007:12:10" user1#0
Reservation "user1#0" is modified
```

Then the administrator can use host `hostA` for other reservations during the duration

```
brsvadd -n 4 -m hostA -u user1 -b "2007:12:1:6:0" -e "2007:12:1:8:0"
Reservation "user1#2" is created
```

Change users and user groups

Use the **brsvmod -u**, **brsvmod adduser**, or **brsvmod rmuser** command to change the users or user groups that are able to submit jobs with the advance reservation.

Jobs that are submitted by the original user or user group to the reservation still belong to the reservation and are scheduled as advance reservation jobs, but new submitted jobs from the removed user or user group cannot use the reservation any longer.

brun command

An advance reservation job dispatched with the **brun** command is still subject to run windows and suspending conditions of the advance reservation for the job. The job must finish running before the time window of a closed reservation expires. Extending or shrinking a closed advance reservation duration prolongs or shortens lifetime of a **brun** job.

bslots command

The **bslots** command displays a snapshot of the slots currently not in use by parallel jobs or advance reservations. If the hosts or duration of an advance reservation is modified, the **bslots** command recalculates and displays the available slots and available run time accordingly.

How advance reservation modifications interact

The following table summarizes how advance reservation modification applies to various advance reservation instances.

Modification			Disable	Begin time	End time	Add hosts	Remove hosts	User/usergroup	open/closed	Pre command	Post command
One-time	Active		No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Inactive		No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Recurring	Occurrences	All	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
		Specified	Yes	No	No	No	No	No	No	No	No
	Active instance		No	No	No	No	No	No	No	No	No

In this table, Yes means that the modification is supported. No means that the change is not allowed. For example, all modifications are acceptable in the case that the advance reservation is inactive and not disabled.

Removing an advance reservation

Use the **brsvdel** command to delete reservations.

Procedure

Specify the reservation ID for the reservation you want to delete.

For example:

```
brsvdel user1#0Reservation user1#0 is being deleted
```

You can delete more than one reservation at a time. Administrators can delete any reservation, but users may only delete their own reservations.

If the recurring reservation is deleted with the **brsvdel** command, jobs running in the reservation are detached from the reservation and scheduled as normal jobs.

If an active reservation is removed with the **brsvdel** command, any specified post-scripts (-Ep option) are not run.

Viewing advance reservation information

Use the **brsvs** command to view information about advance reservations. You can see all current reservations, show a weekly planner for your reservations, or see reservation types and their associated jobs. Use the **bjobs** command to see the reservation ID for an advance reservation job. Use the **bacct** command to view historical accounting information for advance reservations.

Procedure

Use the **brsvs** command with no options to show current reservations.

```
brsvs
RSVID      TYPE    USER    NCPUS    RSV_HOSTS    TIME_WINDOW
user1#0     user    user1    0/1024    hostA:0/1024  11/12/6/0-11/12/8/0
user2#0     user    user2    0/1024    hostA:0/1024  12:0-14:0 *
groupA#0    group   groupA   -/2048    hostA:-/1024  3:0:0-3:3:0 *
            hostB:0/1024
system#0    sys     system   1024     hostA:0/1024  5:18:0-5:20:0 *
```

The `TIME_WINDOW` column shows the following information:

- A one-time reservation displays fields that are separated by slashes (*month/day/hour/minute*):
`11/12/14/0-11/12/18/0`
- A recurring reservation displays fields that are separated by colons (*day:hour:minute*). An asterisk (*) indicates a recurring reservation:

```
5:18:0-5:20:0 *
```

The `NCPUS` and `RSV_HOSTS` columns show remote reservations but do not display details:

```
-/2048    hostA:-/1024
```

Showing a weekly planner for advance reservations

Use the **brsvs -p** and **brsvs -z** commands to show a weekly planner for specified hosts using advance reservation.

Procedure

1. Use the all keyword to show the planner for all hosts with reservations.

The output of the **brsvs -p** command is displayed in weeks. The week starts on Sunday. The timeframe of a recurring reservation is not displayed, since it is unlimited. The timeframe of one-time reservation is displayed in terms of a week. If the reservation spans multiple weeks, these weeks are displayed separately. If a week contains a one-time reservation and a recurring reservation, the timeframe is displayed, since that is relevant for one-time reservation.

Tip: The **MAX** field indicates the configured maximum number of job slots for the host (the MXJ parameter that is defined in the lsb.hosts file).

```
brsvs -p all
RSVID      TYPE      USER      NCPUS      RSV_HOSTS      TIME_WINDOW
user1#0     user      user1     0/1024     hostA:0/1024   11/12/6/0-11/12/8/0
user2#0     user      user2     0/1024     hostA:0/1024   12:0-14:0 *
groupA#0    group    groupA    0/2048     hostA:0/1024   3:0:0-3:3:0 *
            hostB:0/1024
system#0    sys      system    1024      hostA:0/1024   5:18:0-5:20:0 *
HOST: hostA (MAX = 1024)
Week: 11/11/2009 - 11/17/2009
Hour:Min    Sun      Mon      Tue      Wed      Thu      Fri      Sat
0:0         0        0        0        1024     0        0        0
0:10        0        0        0        1024     0        0        0
0:20        0        0        0        1024     0        0        0
...
2:30        0        0        0        1024     0        0        0
2:40        0        0        0        1024     0        0        0
2:50        0        0        0        1024     0        0        0
3:0         0        0        0        0        0        0        0
3:10        0        0        0        0        0        0        0
3:20        0        0        0        0        0        0        0
...
5:30        0        0        0        0        0        0        0
5:40        0        0        0        0        0        0        0
5:50        0        0        0        0        0        0        0
6:0         0        1024     0        0        0        0        0
6:10        0        1024     0        0        0        0        0
6:20        0        1024     0        0        0        0        0
...
7:30        0        1024     0        0        0        0        0
7:40        0        1024     0        0        0        0        0
7:50        0        1024     0        0        0        0        0
8:0         0        0        0        0        0        0        0
8:10        0        0        0        0        0        0        0
8:20        0        0        0        0        0        0        0
...
11:30       0        0        0        0        0        0        0
11:40       0        0        0        0        0        0        0
11:50       0        0        0        0        0        0        0
12:0        1024    1024    1024    1024    1024    1024    1024
12:10       1024    1024    1024    1024    1024    1024    1024
12:20       1024    1024    1024    1024    1024    1024    1024
...
13:30       1024    1024    1024    1024    1024    1024    1024
13:40       1024    1024    1024    1024    1024    1024    1024
13:50       1024    1024    1024    1024    1024    1024    1024
14:0        0        0        0        0        0        0        0
14:10       0        0        0        0        0        0        0
14:20       0        0        0        0        0        0        0
...
17:30       0        0        0        0        0        0        0
17:40       0        0        0        0        0        0        0
17:50       0        0        0        0        0        0        0
18:0        0        0        0        0        0        1024    0
18:10       0        0        0        0        0        1024    0
18:20       0        0        0        0        0        1024    0
...
19:30       0        0        0        0        0        1024    0
19:40       0        0        0        0        0        1024    0
19:50       0        0        0        0        0        1024    0
20:0        0        0        0        0        0        0        0
20:10       0        0        0        0        0        0        0
20:20       0        0        0        0        0        0        0
```

```
...
```

23:30	0	0	0	0	0	0	0
23:40	0	0	0	0	0	0	0
23:50	0	0	0	0	0	0	0

HOST: hostB (MAX = 1024)
Week: 11/11/2009 - 11/17/2009

Hour:Min	Sun	Mon	Tue	Wed	Thu	Fri	Sat

0:0	0	0	0	1024	0	0	0
0:10	0	0	0	1024	0	0	0
0:20	0	0	0	1024	0	0	0
...							
2:30	0	0	0	1024	0	0	0
2:40	0	0	0	1024	0	0	0
2:50	0	0	0	1024	0	0	0
3:0	0	0	0	0	0	0	0
3:10	0	0	0	0	0	0	0
3:20	0	0	0	0	0	0	0
...							
23:30	0	0	0	0	0	0	0
23:40	0	0	0	0	0	0	0
23:50	0	0	0	0	0	0	0

2. Use the **brsvs -z** command instead of the **brsvs -p** command to show only the weekly items that have reservation configurations. Lines that show all zero are omitted.

```
brsvs -z all
```

RSVID	TYPE	USER	NCPUS	RSV_HOSTS	TIME_WINDOW
user1_1	user	user1	0/3	hostA:0/2 hostB:0/1	12/28/14/30-12/28/15/30

HOST: hostA (MAX = 2)
Week: 12/23/2007 - 12/29/2007

Hour:Min	Sun	Mon	Tue	Wed	Thu	Fri	Sat

14:30	0	0	0	0	0	1	0
14:40	0	0	0	0	0	1	0
14:50	0	0	0	0	0	1	0
15:0	0	0	0	0	0	1	0
15:10	0	0	0	0	0	1	0
15:20	0	0	0	0	0	1	0

HOST: hostB (MAX = 2)
Week: 12/23/2007 - 12/29/2007

Hour:Min	Sun	Mon	Tue	Wed	Thu	Fri	Sat

14:30	0	0	0	0	0	2	0
14:40	0	0	0	0	0	2	0
14:50	0	0	0	0	0	2	0
15:0	0	0	0	0	0	2	0
15:10	0	0	0	0	0	2	0
15:20	0	0	0	0	0	2	0

Showing reservation types and associated jobs

The **brsvs -l** command shows each advance reservation in long format.

Procedure

Use the **-l** option of the **brsvs** command to show each advance reservation in long format.

The rows that follow the reservation information show the

- The status of the reservation
- Time when the next instance of recurring reservation is active
- Type of reservation (open or closed)
- The status by job ID of any job associated with the specified reservation (FINISHED, PEND, RUN, or SUSP)

```
brsvs -l
```

RSVID	TYPE	USER	NCPUS	RSV_HOSTS	TIME_WINDOW
user1_1#0	user	user1_1	10/10	host1:4/4 host2:4/4	8:00-22:00 *

```

host3:2/2
Reservation Status: Active
Next Active Period:
    Sat Aug 22 08:00:00 2009 - Sat Aug 22 22:00:00 2009
Creator: user1 1
Reservation Type: CLOSED
FINISHED Jobs: 203 204 205 206 207 208 209 210 211 212
PEND Jobs: 323 324
RUN Jobs: 313 314 316 318 319 320 321 322
SUSP Jobs: 315 317
Resource Unit: Host

```

Specify a reservation name to see information about a single reservation:

```

brsvs -l user1_1#0
RSVID      TYPE  USER  NCPUS  RSV_HOSTS  TIME_WINDOW
user1_1#0  user  user1  0/2    lsfrhel04:0/2  4/13/23/30-4/13/23/40
Reservation Status: Inactive
Creator: user1
Reservation Type: CLOSED
Resource Unit: Slot

```

Showing the reservation ID for an advance reservation job

The **bjobs -l** command shows the reservation ID used by a job.

Procedure

Use the **bjobs -l** command.

```

bjobs -l
Job <1152>, User <user1>, Project <default>, Status <PEND>, Queue <normal>,
Reservation <user1#0>, Command <covfefe>
Mon Nov 12 5:13:21 2009: Submitted from host <hostB>, CWD </home/user1/jobs>;
...

```

Viewing historical accounting information for advance reservations

The **bacct -U** command shows historical accounting information about advance reservations.

Procedure

Use the **-U** option of the **bacct** command.

The **bacct -U** command summarizes all historical modification of the reservation and displays information similar to the **brsvs** command:

- The reservation ID specified on the **-U** option.
- The type of reservation (**user** or **system**)
- The user names of users who used the **brsvadd** command to create the advance reservations
- The user names of the users who can use the advance reservations for jobs that are submitted with the **bsub -U** option.
- Number of slots reserved
- List of hosts for which job slots are reserved
- Time window for the reservation.
 - A one-time reservation displays fields that are separated by slashes (*month/day/hour/minute*).

```
11/12/14/0-11/12/18/0
```

- A recurring reservation displays fields that are separated by colons (*day:hour:minute*).

```
5:18:0 5:20:0
```

For example, the following advance reservation has four time modifications during its life time. The original reservation has the scope of one user (**user1**) and one host (**hostA**) with one slot. The various modifications change the user to **user2**, then back to **user1**, adds, then removes one slot from the reservation.

```

bacct -U user1#1
Accounting about advance reservations that are:

```

```

- accounted on advance reservation IDs user1#1,
- accounted on advance reservations created by user1,
----- SUMMARY -----
RSVID:                user1#1
TYPE:                 user
CREATOR:              user1
Total number of jobs: 0
Total CPU time consumed: 0.0 second
Maximum memory of a job: 0.0 MB
Maximum swap of a job: 0.0 MB
Total active time:    0 hour 6 minute 42 second
Resource Unit:        Host
----- Configuration 0 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user1    1        hostA:1
Active time with this configuration: 0 hour 0 minute 16 second
----- Configuration 1 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user2    1        hostA:1
Active time with this configuration: 0 hour 0 minute 24 second
----- Configuration 2 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user2    1        hostA:1
Active time with this configuration: 0 hour 1 minute 58 second
----- Configuration 3 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user1    2        hostA:2
Active time with this configuration: 0 hour 1 minute 34 second
----- Configuration 4 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#1   user    user1    user1    1        hostA:2
Active time with this configuration: 0 hour 2 minute 30 second

```

The following reservation (user2#0) has one time modification during its life time. The original one has the scope of one user (user2) and one host (hostA) with one slot; the modification changes the user to user3.

```

bacct -U user2#0
Accounting about advance reservations that are:
- accounted on all advance reservation IDs:
- accounted on advance reservations created by all users:
----- SUMMARY -----
RSVID:                user2#0
TYPE:                 user
CREATOR:              user2
Total number of jobs: 1
Total CPU time consumed: 5.0 second
Maximum memory of a job: 1.7 MB
Maximum swap of a job: 7.5 MB
Total active time:    2 hour 0 minute 0 second
----- Configuration 0 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#0   user    user2    user2    1        hostA:1
Active time with this configuration: 1 hour 0 minute 0 second
----- Configuration 1 -----
RSVID    TYPE    CREATOR  USER    NCPUS    RSV_HOSTS
user1#0   user    user2    user3    1        hostA:1
Active time with this configuration: 1 hour 0 minute 0 second

```

Submitting and modifying jobs that use advance reservations

The -U option of the **bsub** command submits jobs with a reservation ID.

Procedure

Use the -U option of the **bsub** command to submit jobs with a reservation ID.

```
bsub -U user1#0 myjob
```

The job can use only hosts that are reserved by the reservation `user1#0`. By default, LSF selects only the hosts in the reservation. Use the `-m` option to specify particular hosts within the list of hosts that are reserved by the reservation. You can select only from hosts that were included in the original reservation.

If you do not specify hosts (the **bsub -m** command) or resource requirements (the **bsub -R** command), the default resource requirement selects the hosts that are of any host type (assumes "type==any" instead of "type==local" as the default `select` string).

If you later delete the advance reservation while it is still active, any pending jobs still keep the "type==any" attribute.

A job can use only one reservation. The number of jobs that can be submitted to a reservation is not limited, but the reservation might run out of slots available on the hosts in the reservation. For example, reservation `user2#0` reserves 1024 slots on `hostA`. When all 1024 slots on `hostA` are used by jobs the reference `user2#0`, `hostA` is no longer available to other jobs that use reservation `user2#0`. Any single user or user group can have a maximum of 100 reservation IDs.

Jobs referencing the reservation are killed when the reservation expires.

Modifying a job reservation ID

The `-U` option of the **bmod** command changes a job to another reservation ID.

Before you begin

You must be an administrator to modify a job reservation ID.

Procedure

1. Use the `-U` option of the **bmod** command to change a job to another reservation ID.

```
bmod -U user1#0 1234
```

2. To cancel the reservation, use the `-Un` option of the **bmod** command.

```
bmod -Un 1234
```

Use the **bmod -Un** option to detach a running job from an *inactive* open reservation. After the job is detached from the reservation, it is scheduled like a normal job.

Viewing jobs that are associated with an advance reservation

Procedure

- To view all jobs that are associated with a particular reservation ID, use the **bjobs -U** option with 0 as the job ID. For example, to display all jobs that are associated with the reservation ID `user1#2`:

```
bjobs -U user1#2 0
```

- To see if a single job is associated with a particular reservation ID, use the **bjobs -U** option with a specific job ID. For example, to display one job with ID 101 if it is associated with the reservation ID `user1#2`:

```
bjobs -U user1#2 101
```

- To see which jobs in a specified list are associated with a particular reservation ID, use the **bjobs -U** and **-l** options with multiple job IDs.
Displays any jobs from a specified list (with IDs 101, 102, 103, and 104) that are associated with the reservation ID `user1#2`:

```
bjobs -U user1#2 -l 101 102 103 104
```

Viewing the reservation ID for a job

Procedure

Use the `-o` option of the **bjobs** command and include the `rsvld` column to view the reservation ID for the specified jobs. For more information on customizing job information with the **bjobs -o** option, see [Customize job information output](#).

Advance reservation behavior and operations

A job that uses a reservation is subject to all job resource usage limits. Advance reservation preemption allows advance reservation jobs to use the slots that are reserved by the reservation. You can create and use advance reservations for the LSF multicluster capability job forwarding model. Resizable jobs and jobs with compute unit resource requirements can be dispatched only after the advance reservation becomes active.

Job resource usage limits and job chunking

A job that uses a reservation is subject to all job resource usage limits. If a limit is reached on a particular host in a reservation, jobs that use that reservation cannot start on that host.

An advance reservation job is dispatched to its reservation even if the run limit or estimated run time of the job exceeds the remaining active time of the reservation. For example, if a job has a run limit of 1 hour, and a reservation has a remaining active time of 1 minute, the job is still dispatched to the reservation. If the reservation is closed, the job is terminated when the reservation expires.

Similarly, when your job uses chunk job scheduling, advance reservation jobs are chunked together as usual when dispatched to a host of the reservation without regard to the expiry time of the reservation. Chunking occurs even when the jobs are given a run limit or estimated run time. If the reservation is closed, the jobs in `WAIT` state are terminated when the reservation expires.

Advance reservation preemption

Advance reservation preemption allows advance reservation jobs to use the slots that are reserved by the reservation. Slots that are occupied by non-advance jobs might be preempted when the reservation becomes active.

Without modification with **brsvmod**, advance reservation preemption is triggered at most once per reservation period (a non-recurring reservation has only one period) whenever *both* of the following conditions are met:

- The reservation is active.
- At least one job associated with the advance reservation is pending or suspended.

If an advance reservation is modified, preemption is done for an active advance reservation after every modification of the reservation when there is at least one pending or suspended job that is associated with the reservation.

When slots are added to an advance reservation with **brsvmod**, LSF preempts running non-reservation jobs if necessary to provide slots for jobs that belong to the reservation. Preemption is triggered if pending or suspended jobs belong to the reservation in the system.

When preemption is triggered, non-advance reservation jobs are suspended and their slots that are given to the advance reservation on the hosts that belong to the reservation. On each host, enough non-advance reservation jobs are suspended so that all of slots required by the advance reservation are obtained. The number of slots obtained does not depend on the number of jobs submitted to the advance reservation. Non-advance reservation jobs on a host can only use slots not assigned to the advance reservation.

When a job is preempted for an advance reservation, it can only resume on the host when either the advance reservation finishes, or some other non-advance reservation job finishes on the host.

For example, a single-host cluster has 10 slots, with 9 non-advance reservation jobs dispatched to the host (each requiring one slot). An advance reservation that uses 5 slots on the host is created, and a single job is submitted to the reservation. When the reservation becomes active, 4 of the non-advance reservation jobs are suspended, and the advance reservation job will start.

Force a job to run before a reservation is active

LSF administrators can use **brun** to force jobs to run before the reservation is active, but the job must finish running before the time window of the reservation expires.

For example, if the administrator forces a job with a reservation to run one hour before the reservation is active, and the reservation period is 3 hours, a 4 hour run limit takes effect.

Host intersection and advance reservation

When the `ENABLE_HOST_INTERSECTION=Y` parameter is specified in the `lsb.params` file, LSF finds any existing intersection with hosts specified in the queue and those specified at job submission by **bsub -m** and/or hosts with advance reservation. When specifying keywords such as `all`, `allremote`, and `others`, LSF finds an existing intersection of hosts available and the job runs rather than being rejected.

Advance reservations across clusters

You can create and use advance reservation for the LSF multicluster capability job forwarding model. To enable this feature, you must upgrade all clusters to LSF 10 or later.

See [IBM Spectrum LSF multicluster capability](#) for more information.

Resizable jobs and advance reservations

Like regular jobs, resizable jobs associated with an advance reservation can be dispatched only after the reservation becomes active, and the minimum processor request can be satisfied. The allocation request is treated like a regular advance reservation job, which relies on slots available to the reservation. If an advance reservation gets more resources by modification (**brsvmmod addhost**), those resources can be used by pending allocation requests immediately.

The following table summarizes the relationship of the advance reservation lifecycle and resizable job requests:

Advance Reservation		Resizable job	Allocation request
One-time expired/deleted	Open	RUN->SSUSP->RUN	Postponed until the job runs
	Closed	Removed	Removed
Recurrent expired/deleted	Open	SSUSP till next instance	Postponed until the job runs again in next instance
	Closed	Removed	Removed

By the time a reservation has expired or deleted, the status change of the resizable job to SSUSP blocks a resizable job allocation request from being scheduled.

Released slots from a resizable job can be reused by other jobs in the reservation.

Resizable advance reservation jobs can preempt non-advance reservation jobs that are consuming the slots that belong to the reservation. Higher priority advance reservation jobs can preempt low priority advance reservation jobs, regardless of whether both are resizable jobs.

Allocation requests of resizable AR jobs honor limits configuration. They cannot preempt any limit tokens from other jobs.

Compute units and advance reservations

Like regular jobs, jobs with compute unit resource requirements and an advance reservation can be dispatched only after the reservation becomes active, and the minimum processor request can be satisfied.

In the case of exclusive compute unit jobs (with the resource requirement `cu[excl]`), the advance reservation can affect hosts outside the advance reservation but in the same compute unit as follows:

- An exclusive compute unit job dispatched to a host inside the advance reservation will lock the entire compute unit, including any hosts outside the advance reservation.

- An exclusive compute unit job dispatched to a host outside the advance reservation will lock the entire compute unit, including any hosts inside the advance reservation.

Ideally, all hosts belonging to a compute unit should be inside or outside of an advance reservation.

Fair share scheduling

fair share scheduling divides the processing power of the LSF cluster among users and queues to provide fair access to resources, so that no user or queue can monopolize the resources of the cluster and no queue will be starved.

To configure any kind of fair share scheduling, you should understand the following concepts:

- User share assignments
- Dynamic share priority
- Job dispatch order

You can configure fair share at either host level or queue level. If you require more control, you can implement hierarchical fair share. You can also set some additional restrictions when you submit a job.

Note: If you enable the RELAX_JOB_DISPATCH_ORDER parameter in the lsb.params file, which allows LSF to deviate from standard job prioritization policies, LSF might break the job dispatch order as specified by the fair share policy.

- [Fair share scheduling](#)
By default, LSF considers jobs for dispatch in the same order as they appear in the queue (which is not necessarily the order in which they are submitted to the queue). This is called first-come, first-served (FCFS) scheduling.
- [Ways to configure fair share](#)
- [User-based fair share](#)
User-based fair share lets you allocate resources to users in a hierarchical manner.
- [Use time decay and committed run time](#)
By default, as a job is running, the dynamic priority decreases gradually until the job has finished running, then increases immediately when the job finishes.
- [How fair share affects job dispatch order](#)
Within a queue, jobs are dispatched according to the queue's scheduling policy.
- [Host partition user-based fair share](#)
User-based fair share policies that are configured at the host level handle resource contention across multiple queues. You can define a different fair share policy for every host partition. If multiple queues use the host partition, a user has the same priority across multiple queues.
- [Queue-level user-based fair share](#)
User-based fair share policies configured at the queue level handle resource contention among users in the same queue. You can define a different fair share policy for every queue, even if they share the same hosts. A user's priority is calculated separately for each queue.
- [Queue-based fair share](#)
When a priority is set in a queue configuration, a high priority queue tries to dispatch as many jobs as it can before allowing lower priority queues to dispatch any job. Lower priority queues are blocked until the higher priority queue cannot dispatch any more jobs. However, it may be desirable to give some preference to lower priority queues and regulate the flow of jobs from the queue.
- [Slot allocation per queue](#)
- [Users affected by multiple fair share policies](#)
If you belong to multiple user groups, which are controlled by different fair share policies, each group probably has a different dynamic share priority at any given time. By default, if any one of these groups becomes the highest priority user, you could be the highest priority user in that group, and LSF would attempt to place your job.
- [Re-sizable jobs and fair share](#)
Re-sizable jobs submitting into fair share queues or host partitions are subject to fair share scheduling policies. The dynamic priority of the user who submitted the job is the most important criterion. LSF treats pending resize allocation requests as a regular job and enforces the fair share user priority policy to schedule them.
- [Parallel fair share](#)
LSF can consider the number of CPUs when using fair share scheduling with parallel jobs.
- [User share assignments](#)
Both queue-level and host partition fair share use the following syntax to define how shares are assigned to users or

- user groups
- **[Dynamic user priority](#)**
LSF calculates a dynamic user priority for individual users or for a group, depending on how the shares are assigned. The priority is dynamic because it changes as soon as any variable in formula changes. By default, a user's dynamic priority gradually decreases after a job starts, and the dynamic priority immediately increases when the job finishes.
- **[Cross-queue user-based fair share](#)**
User-based fair share policies configured at the queue level handle resource contention across multiple queues.
- **[Slot allocation per queue](#)**
- **[Typical slot allocation scenarios](#)**
- **[Job count based fair share scheduling](#)**
Job count based fair share scheduling uses the number of jobs in the fair share scheduling algorithm instead of the number of job slots.

Fair share scheduling

By default, LSF considers jobs for dispatch in the same order as they appear in the queue (which is not necessarily the order in which they are submitted to the queue). This is called first-come, first-served (FCFS) scheduling.

Fair share scheduling divides the processing power of the LSF cluster among users and queues to provide fair access to resources, so that no user or queue can monopolize the resources of the cluster and no queue will be starved.

If your cluster has many users competing for limited resources, the FCFS policy might not be enough. For example, one user could submit many long jobs at once and monopolize the cluster's resources for a long time, while other users submit urgent jobs that must wait in queues until all the first user's jobs are all done. To prevent this, use fair share scheduling to control how resources should be shared by competing users.

Fair sharing is not necessarily equal sharing: you can assign a higher priority to the most important users. If there are two users competing for resources, you can:

- Give all the resources to the most important user
- Share the resources so the most important user gets the most resources
- Share the resources so that all users have equal importance

Queue-level vs. host partition fair share

You can configure fair share at either the queue level or the host level. However, these types of fair share scheduling are mutually exclusive. You cannot configure queue-level fair share and host partition fair share in the same cluster.

If you want a user's priority in one queue to depend on their activity in another queue, you must use cross-queue fair share or host-level fair share.

Fair share policies

A fair share policy defines the order in which LSF attempts to place jobs that are in a queue or a host partition. You can have multiple fair share policies in a cluster, one for every different queue or host partition. You can also configure some queues or host partitions with fair share scheduling, and leave the rest using FCFS scheduling.

How fair share scheduling works

Each fair share policy assigns a fixed number of shares to each user or group. These shares represent a fraction of the resources that are available in the cluster. The most important users or groups are the ones with the most shares. Users who have no shares cannot run jobs in the queue or host partition.

A user's dynamic priority depends on their share assignment, the dynamic priority formula, and the resources their jobs have already consumed.

The order of jobs in the queue is secondary. The most important thing is the dynamic priority of the user who submitted the job. When fair share scheduling is used, LSF tries to place the first job in the queue that belongs to the user with the highest

dynamic priority.

Parallel fair share

LSF can consider the number of CPUs when using fair share scheduling with parallel jobs.

If the job is submitted with **bsub -n**, the following formula is used to calculate dynamic priority:

dynamic priority = number_shares / A

where:

A

Calculated as:

```
(cpu_time * CPU_TIME_FACTOR + run_time * number_CPUs * RUN_TIME_FACTOR + (1 + job_slots) * RUN_JOB_FACTOR + fair_share_adjustment(struc* shareAdjustPair) * FAIRSHARE_ADJUSTMENT_FACTOR + (historical_gpu_run_time + gpu_run_time) * ngpus_physical * GPU_RUN_TIME_FACTOR)
```

number_CPUs

The number of CPUs used by the job.

- [Configuring parallel fair share](#)

You can configure parallel fair share so that the number of CPUs is considered when calculating dynamic priority for queue-level user-based fair share.

Configuring parallel fair share

You can configure parallel fair share so that the number of CPUs is considered when calculating dynamic priority for queue-level user-based fair share.

About this task

Note: LSB_NCPU_ENFORCE does not apply to host-partition user-based fair share. For host-partition user-based fair share, the number of CPUs is automatically considered.

Procedure

1. Configure fair share at the queue level.
2. Enable parallel fair share: LSB_NCPU_ENFORCE=1 in lsf.conf.
3. Run the following commands to restart all LSF daemons:

```
# lsadmin reconfig
# bctrld restart res all
# bctrld restart sbd all
# badmin mbdrestart
```

User share assignments

Both queue-level and host partition fair share use the following syntax to define how shares are assigned to users or user groups

Syntax

`[user, number_shares]`

Enclose each user share assignment in square brackets, as shown. Separate multiple share assignments with a space between each set of square brackets.

user

Specify users of the queue or host partition. You can assign the shares:

- to a single user (specify *user_name*)
- to users in a group, individually (specify *group_name@*) or collectively (specify *group_name*)
- to users not included in any other share assignment, individually (specify the keyword *default*) or collectively (specify the keyword *others*)

By default, when resources are assigned collectively to a group, the group members compete for the resources according to FCFS scheduling. You can use hierarchical fair share to further divide the shares among the group members.

When resources are assigned to members of a group individually, the share assignment is recursive. Members of the group and of all subgroups always compete for the resources according to FCFS scheduling, regardless of hierarchical fair share policies.

number_shares

Specify a positive integer representing the number of shares of cluster resources assigned to the user.

The number of shares assigned to each user is only meaningful when you compare it to the shares assigned to other users, or to the total number of shares. The total number of shares is just the sum of all the shares assigned in each share assignment.

Examples

```
[User1, 1] [GroupB, 1]
```

Assigns two shares: 1 to *User1*, and 1 to be shared by the users in *GroupB*. Each user in *GroupB* has equal importance. *User1* is as important as all the users in *GroupB* put together. In this example, it does not matter if the number of shares is one, six, or 600. As long as *User1* and *GroupB* are both assigned the same number of shares, the relationship stays the same.

```
[User1, 10] [GroupB@, 1]
```

If *GroupB* contains ten users, assigns 20 shares in total: ten to *User1*, and 1 to each user in *GroupB*. Each user in *GroupB* has equal importance. *User1* is ten times as important as any user in *GroupB*.

```
[User1, 10] [User2, 9] [others, 8]
```

Assigns 27 shares: ten to *User1*, none to *User2*, and eight to the remaining users, as a group. *User1* is slightly more important than *User2*. Each of the remaining users has equal importance.

- If there are three users in total, the single remaining user has all eight shares, and is almost as important as *User1* and *User2*.
- If there are 12 users in total, then ten users compete for those eight shares, and each of them is significantly less important than *User1* and *User2*.

```
[User1, 10] [User2, 6] [default, 4]
```

The relative percentage of shares held by a user will change, depending on the number of users who are granted shares by default.

- If there are three users in total, assigns 20 shares: ten to *User1*, six to *User2*, and four to the remaining user. *User1* has half of the available resources (ten shares out of 20).
- If there are 12 users in total, assigns 56 shares: ten to *User1*, six to *User2*, and 4 to each of the remaining ten users. *User1* has about a fifth of the available resources (ten shares out of 56).

Dynamic user priority

LSF calculates a dynamic user priority for individual users or for a group, depending on how the shares are assigned. The priority is dynamic because it changes as soon as any variable in formula changes. By default, a user's dynamic priority gradually decreases after a job starts, and the dynamic priority immediately increases when the job finishes.

How LSF calculates dynamic priority

By default, LSF calculates the dynamic priority for each user based on the following criteria:

- The number of shares assigned to the user
- The resources used by jobs belonging to the user:
 - Number of job slots reserved and in use
 - Run time of running jobs
 - Cumulative actual CPU time (not normalized), adjusted so that recently used CPU time is weighted more heavily than CPU time used in the distant past

If you enable additional functionality, the formula can also involve additional resources used by jobs belonging to the user:

- Decayed run time of running jobs
- Historical run time of finished jobs
- Committed run time, specified at job submission with the **-W** option of **bsub**, or in the queue with the RUNLIMIT parameter in **lsb.queues**
- Memory usage adjustment made by the fair share plug-in (**libfairshareadjust.***).

How LSF measures fair share resource usage

LSF resource usage differently, depending on the type of fair share:

- For user-based fair share:
 - For queue-level fair share, LSF measures the resource consumption of all the user's jobs in the queue. This means a user's dynamic priority can be different in every queue.
 - For host partition fair share, LSF measures resource consumption for all the user's jobs that run on hosts in the host partition. This means a user's dynamic priority is the same in every queue that uses hosts in the same partition.
- For queue-based fair share, LSF measures the resource consumption of all jobs in each queue.

Default dynamic priority formula

By default, LSF calculates dynamic priority according to the following formula:

$$\text{dynamic priority} = \text{number_shares} / (\text{cpu_time} * \text{CPU_TIME_FACTOR} + \text{run_time} * \text{RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR} + (1 + \text{fwd_job_slots}) * \text{FWD_JOB_FACTOR} + \text{fairshare_adjustment} * \text{FAIRSHARE_ADJUSTMENT_FACTOR}) + ((\text{historical_gpu_run_time} + \text{gpu_run_time}) * \text{ngpus_physical}) * \text{GPU_RUN_TIME_FACTOR}$$

Note: The maximum value of dynamic user priority is 100 times the number of user shares (if the denominator in the calculation is less than 0.01, LSF rounds up to 0.01).

For *cpu_time*, *run_time*, and *job_slots*, LSF uses the total resource consumption of all the jobs in the queue or host partition that belong to the user or group.

number_shares

The number of shares assigned to the user.

cpu_time

The cumulative CPU time used by the user (measured in hours). LSF calculates the cumulative CPU time using the actual (not normalized) CPU time and a decay factor such that 1 hour of recently-used CPU time decays to 0.1 hours after an interval of time specified by HIST_HOURS in **lsb.params** (5 hours by default).

run_time

The total run time of running jobs (measured in hours).

job_slots

The number of job slots reserved and in use.

fairshare_adjustment

The adjustment calculated by the fair share adjustment plug-in (libfairshareadjust.*).

Configure the default dynamic priority

You can give additional weight to the various factors in the priority calculation by setting the following parameters for the queue in `lsb.queues` or for the cluster in `lsb.params`. When the queue value is not defined, the cluster-wide value from `lsb.params` is used.

- CPU_TIME_FACTOR
- RUN_TIME_FACTOR
- RUN_JOB_FACTOR
- FWD_JOB_FACTOR
- FAIRSHARE_ADJUSTMENT_FACTOR
- HIST_HOURS
- GPU_RUN_TIME_FACTOR

If you modify the parameters used in the dynamic priority formula, it affects every fair share policy in the cluster:

CPU_TIME_FACTOR

The CPU time weighting factor.

Default: 0.7

FWD_JOB_FACTOR

The forwarded job slots weighting factor when using the LSF multicluster capability.

Default: Not defined

RUN_TIME_FACTOR

The run time weighting factor.

Default: 0.7

RUN_JOB_FACTOR

The job slots weighting factor.

Default: 3

FAIRSHARE_ADJUSTMENT_FACTOR

The fair share plug-in (libfairshareadjust.*) weighting factor.

Default: 0

HIST_HOURS

Interval for collecting resource consumption history.

Default: 5

GPU_RUN_TIME_FACTOR

GPU run time weighting factor.

Default: 0

Customize the dynamic priority

In some cases the dynamic priority equation may require adjustments beyond the run time, CPU time, and job slot dependencies provided by default. The fair share adjustment plug-in is open source and can be customized once you identify specific requirements for dynamic priority.

All information used by the default priority equation (except the user shares) is passed to the fair share plug-in. In addition, the fair share plug-in is provided with current memory use over the entire cluster and the average memory that is allocated to a slot in the cluster.

Note: If you modify the parameters used in the dynamic priority formula, it affects every fair share policy in the cluster. The fair share adjustment plug-in (libfairshareadjust.*) is not queue-specific. Parameter settings passed to the fair share adjustment plug-in are those defined in `lsb.params`.

Example

Jobs assigned to a single slot on a host can consume host memory to the point that other slots on the hosts are left unusable. The default dynamic priority calculation considers job slots used, but doesn't account for unused job slots effectively blocked by another job.

The fair share adjustment plug-in example code provided by LSF is found in the examples directory of your installation, and implements a memory-based dynamic priority adjustment as follows:

```
fair share adjustment= (1+slots)*((used_memory /used_slots)/(slot_memory*THRESHOLD))
```

used_slots

The number of job slots in use by started jobs.

used_memory

The total memory in use by started jobs.

slot_memory

The average amount of memory that exists per slot in the cluster.

THRESHOLD

The memory threshold set in the fair share adjustment plug-in.

Use time decay and committed run time

By default, as a job is running, the dynamic priority decreases gradually until the job has finished running, then increases immediately when the job finishes.

In some cases this can interfere with fair share scheduling if two users who have the same priority and the same number of shares submit jobs at the same time.

To avoid these problems, you can modify the dynamic priority calculation by using one or more of the following weighting factors:

- Run time decay
- Historical run time decay
- Committed run time
- [Historical run time decay](#)
By default, historical run time does not affect the dynamic priority. You can configure LSF so that the user's dynamic priority increases *gradually* after a job finishes. After a job is finished, its run time is saved as the historical run time of the job and the value can be used in calculating the dynamic priority, the same way LSF considers historical CPU time in calculating priority. LSF applies a decaying algorithm to the historical run time to gradually increase the dynamic priority over time after a job finishes.
- [Run time decay](#)
In a cluster running jobs of varied length, a user running only short jobs may always have a higher priority than a user running a long job. This can happen when historical run time decay is applied, decreasing the impact of the completed short jobs but not the longer job that is still running. To correct this, you can configure LSF to decay the run time of a job that is still running in the same manner historical run time decays.
- [Committed run time weighting factor](#)
The committed run time is the run time requested at job submission with the **-W** option for the **bsub** command, or in the queue configuration with the RUNLIMIT parameter. By default, committed run time does not affect the dynamic priority.

Historical run time decay

By default, historical run time does not affect the dynamic priority. You can configure LSF so that the user's dynamic priority increases *gradually* after a job finishes. After a job is finished, its run time is saved as the historical run time of the job and the value can be used in calculating the dynamic priority, the same way LSF considers historical CPU time in calculating priority.

LSF applies a decaying algorithm to the historical run time to gradually increase the dynamic priority over time after a job finishes.

- [Configuring historical run time](#)
- [How mbatchd reconfiguration and restart affects historical run time](#)

After restarting or reconfiguring the **mbatchd** daemon, the historical run time of finished jobs might be different, since it includes jobs that may have been cleaned from **mbatchd** before the restart. **mbatchd** restart only reads recently finished jobs from `lsb.events`, according to the value of `CLEAN_PERIOD` in `lsb.params`. Any jobs cleaned before restart are lost and are not included in the new calculation of the dynamic priority.

Configuring historical run time

Procedure

Specify `ENABLE_HIST_RUN_TIME=Y` for the queue in `lsb.queues` or for the cluster in `lsb.params`.

Historical run time is added to the calculation of the dynamic priority so that the formula becomes the following:

dynamic priority = *number_shares* / *A*

where *A* is (*cpu_time* * `CPU_TIME_FACTOR` + *run_time* * *number_CPUs* * `RUN_TIME_FACTOR` + (1 + *job_slots*) * `RUN_JOB_FACTOR` + *fairshare_adjustment*(*struc* * *shareAdjustPair*) * `FAIRSHARE_ADJUSTMENT_FACTOR` + (*historical_gpu_run_time* + *gpu_run_time*) * *ngpus_physical* * `GPU_RUN_TIME_FACTOR`)

historical_run_time—(measured in hours) of finished jobs accumulated in the user's share account file. LSF calculates the historical run time using the actual run time of finished jobs and a decay factor such that 1 hour of recently-used run time decays to 0.1 hours after an interval of time specified by `HIST_HOURS` in `lsb.params` (5 hours by default).

How mbatchd reconfiguration and restart affects historical run time

After restarting or reconfiguring the **mbatchd** daemon, the historical run time of finished jobs might be different, since it includes jobs that may have been cleaned from **mbatchd** before the restart. **mbatchd** restart only reads recently finished jobs from `lsb.events`, according to the value of `CLEAN_PERIOD` in `lsb.params`. Any jobs cleaned before restart are lost and are not included in the new calculation of the dynamic priority.

Example

The following fair share parameters are configured in `lsb.params`:

```
CPU_TIME_FACTOR = 0
RUN_JOB_FACTOR  = 0
RUN_TIME_FACTOR = 1
FAIRSHARE_ADJUSTMENT_FACTOR = 0
```

Note that in this configuration, only run time is considered in the calculation of dynamic priority. This simplifies the formula to the following:

dynamic priority = *number_shares* / (*run_time* * `RUN_TIME_FACTOR`)

Without the historical run time, the dynamic priority increases suddenly as soon as the job finishes running because the run time becomes zero, which gives no chance for jobs pending for other users to start.

When historical run time is included in the priority calculation, the formula becomes:

dynamic priority = *number_shares* / (*historical_run_time* + *run_time*) * `RUN_TIME_FACTOR`)

Now the dynamic priority increases gradually as the historical run time decays over time.

Run time decay

In a cluster running jobs of varied length, a user running only short jobs may always have a higher priority than a user running a long job. This can happen when historical run time decay is applied, decreasing the impact of the completed short jobs but not the longer job that is still running. To correct this, you can configure LSF to decay the run time of a job that is still running in the same manner historical run time decays.

Once a job is complete, the decayed run time is transferred to the historical run time where the decay continues. This equalizes the effect of short and long running jobs on user dynamic priority.

Note: Running **badmin reconfig** or restarting **mbatchd** during a job's run time results in the decayed run time being recalculated. When a suspended job using run time decay is resumed, the decay time is based on the elapsed time.

- [Configuring run time decay](#)

Configuring run time decay

Procedure

1. Specify `HIST_HOURS` for the queue in `lsb.queues` or for the cluster in `lsb.params`.
2. Specify `RUN_TIME_DECAY=Y` for the queue in `lsb.queues` or for the cluster in `lsb.params`.
The run time used in the calculation of the dynamic priority so that the formula becomes the following:

$$\text{dynamic priority} = \text{number_shares} / A$$

where *A* is:

```
(cpu_time * CPU_TIME_FACTOR + run_time * number_CPUs * RUN_TIME_FACTOR + (1 + job_slots) * RUN_JOB_FACTOR + fair_share_adjustment(struc* shareAdjustPair) * FAIRSHARE_ADJUSTMENT_FACTOR + (historical_gpu_run_time + gpu_run_time) * ngpus_physical * GPU_RUN_TIME_FACTOR)
```

Committed run time weighting factor

The committed run time is the run time requested at job submission with the **-W** option for the **bsub** command, or in the queue configuration with the `RUNLIMIT` parameter. By default, committed run time does not affect the dynamic priority.

While the job is running, the actual run time is subtracted from the committed run time. The user's dynamic priority decreases immediately to its lowest expected value, and is maintained at that value until the job finishes. Job run time is accumulated as usual, and historical run time, if any, is decayed.

When the job finishes, the committed run time is set to zero and the actual run time is added to the historical run time for future use. The dynamic priority increases gradually until it reaches its maximum value.

Providing a weighting factor in the run time portion of the dynamic priority calculation prevents a job dispatching burst, where one user monopolizes job slots because of the latency in computing run time.

Limitation

If you use queue-level fair share, and a running job has a committed run time, you should not switch that job to or from a fair share queue (using **bswitch**). The fair share calculations will not be correct.

Run time displayed by `bqueues` and `bhpart`

The run time displayed by **bqueues** and **bhpart** is the sum of the actual, accumulated run time and the historical run time, but does not include the committed run time.

- [Configuring committed run time](#)

Configuring committed run time

Procedure

Set a value for the `COMMITTED_RUN_TIME_FACTOR` parameter for the queue in `lsb.queues` or for the cluster in `lsb.params`. You should also specify a `RUN_TIME_FACTOR`, to prevent the user's dynamic priority from increasing as the run time increases.

If you have also enabled the use of historical run time, the dynamic priority is calculated according to the following formula:

```
dynamic priority = number_shares / (cpu_time * CPU_TIME_FACTOR + (historical_run_time +  
run_time) * RUN_TIME_FACTOR + (committed_run_time - run_time) * COMMITTED_RUN_TIME_FACTOR + (1  
+ job_slots) * RUN_JOB_FACTOR + fair_share_adjustment(struct*  
shareAdjustPair)*FAIRSHARE_ADJUSTMENT_FACTOR) + ((historical_gpu_run_time + gpu_run_time) *  
ngpus_physical) * GPU_RUN_TIME_FACTOR
```

where *committed_run_time* is the run time requested at job submission with the **-W** option of **bsub**, or in the queue configuration with the `RUNLIMIT` parameter. This calculation measures the committed run time in hours.

In the calculation of a user's dynamic priority, `COMMITTED_RUN_TIME_FACTOR` determines the relative importance of the committed run time in the calculation. If the **-W** option of **bsub** is not specified at job submission and a `RUNLIMIT` has not been set for the queue, the committed run time is not considered.

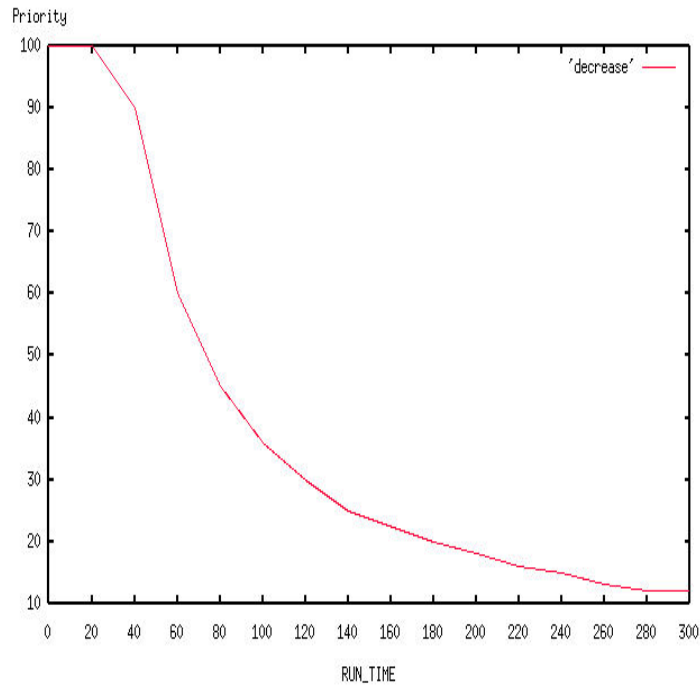
`COMMITTED_RUN_TIME_FACTOR` can be any positive value between 0.0 and 1.0. The default value set in `lsb.params` is 0.0. As the value of `COMMITTED_RUN_TIME_FACTOR` approaches 1.0, more weight is given to the committed run time in the calculation of the dynamic priority.

Example

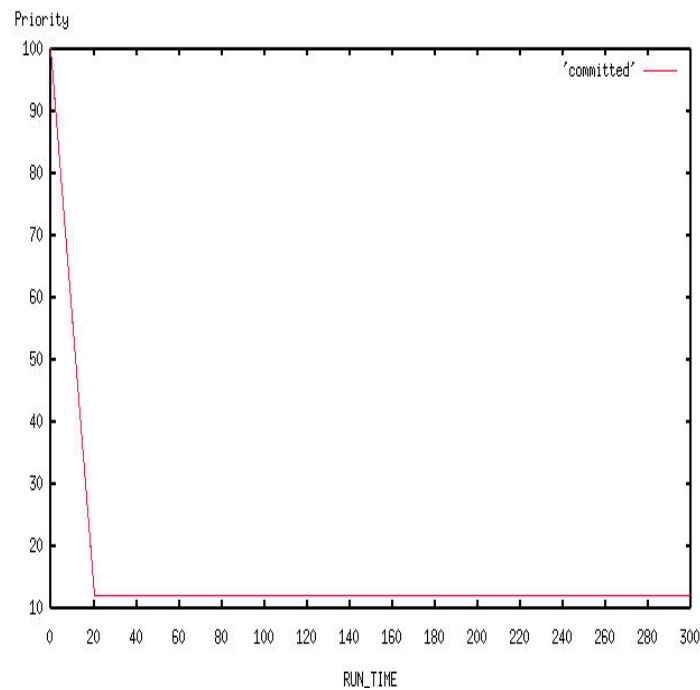
The following fair share parameters are configured in `lsb.params`:

```
CPU_TIME_FACTOR = 0  
RUN_JOB_FACTOR = 0  
RUN_TIME_FACTOR = 1  
FAIRSHARE_ADJUSTMENT_FACTOR = 0  
GPU_RUN_TIME_FACTOR = 0  
COMMITTED_RUN_TIME_FACTOR = 1
```

Without a committed run time factor, dynamic priority for the job owner drops gradually while a job is running:



When a committed run time factor is included in the priority calculation, the dynamic priority drops as soon as the job is dispatched, rather than gradually dropping as the job runs:



How fair share affects job dispatch order

Within a queue, jobs are dispatched according to the queue's scheduling policy.

- For FCFS queues, the dispatch order depends on the order of jobs in the queue (which depends on job priority and submission time, and can also be modified by the job owner).
- For fair share queues, the dispatch order depends on dynamic share priority, then order of jobs in the queue (which is not necessarily the order in which they are submitted to the queue).

A user's priority gets higher when they use less than their fair share of the cluster's resources. When a user has the highest priority, LSF considers one of their jobs first, even if other users are ahead of them in the queue.

If there are only one user's jobs pending, and you do not use hierarchical fair share, then there is no resource contention between users, so the fair share policies have no effect and jobs are dispatched as usual.

Job dispatch order among queues of equivalent priority

The order of dispatch depends on the order of the queues in the queue configuration file. The first queue in the list is the first to be scheduled.

Jobs in a fair share queue are always considered as a group, so the scheduler attempts to place all jobs in the queue before beginning to schedule the next queue.

Jobs in an FCFS queue are always scheduled along with jobs from other FCFS queues of the same priority (as if all the jobs belonged to the same queue).

Example

In a cluster, queues A, B, and C are configured in that order and have equal queue priority.

Jobs with equal job priority are submitted to each queue in this order: C, B, A, B, and A.

- If all queues are FCFS queues, order of dispatch is C, B, A, B, and A (queue A is first; queues B and C are the same priority as A; all jobs are scheduled in FCFS order).
- If all queues are fair share queues, order of dispatch is AA, BB, and C (queue A is first; all jobs in the queue are scheduled; then queue B, then C).
- If A and C are fair share, and B is FCFS, order of dispatch is AA, B, B, and C (queue A jobs are scheduled according to user priority; then queue B jobs are scheduled in FCFS order; then queue C jobs are scheduled according to user priority)
- If A and C are FCFS, and B is fair share, order of dispatch is C, A, A, and BB (queue A is first; queue A and C jobs are scheduled in FCFS order, then queue B jobs are scheduled according to user priority)
- If any of these queues uses cross-queue fair share, the other queues must also use cross-queue fair share and belong to the same set, or they cannot have the same queue priority.

Host partition user-based fair share

User-based fair share policies that are configured at the host level handle resource contention across multiple queues. You can define a different fair share policy for every host partition. If multiple queues use the host partition, a user has the same priority across multiple queues.

To run a job on a host that has fair share, users must have a share assignment (USER_SHARES in the **HostPartition** section of `lsb.hosts`). Even cluster administrators cannot submit jobs to a fair share host if they do not have a share assignment.

- [Configuring host partition fair share scheduling](#)

Configuring host partition fair share scheduling

Procedure

To configure host partition fair share, define a host partition in `lsb.hosts`. Use the following format:

```
Begin HostPartition
HPART_NAME = Partition1
HOSTS = hostA hostB ~hostC
```

```
USER_SHARES = [groupA@, 3] [groupB, 7] [default, 1]
End HostPartition
```

- A host cannot belong to multiple partitions.
 - Optional: Use the reserved host name *all* to configure a single partition that applies to all hosts in a cluster.
 - Optional: Use the not operator (~) to exclude hosts or host groups from the list of hosts in the host partition.
 - Hosts in a host partition cannot participate in queue-based fair share.
- Hosts that are not included in any host partition are controlled by FCFS scheduling policy instead of fair share scheduling policy.

Queue-level user-based fair share

User-based fair share policies configured at the queue level handle resource contention among users in the same queue. You can define a different fair share policy for every queue, even if they share the same hosts. A user's priority is calculated separately for each queue.

To submit jobs to a fair share queue, users must be allowed to use the queue (USERS in the `lsb.queues` file) and must have a share assignment (a value of **fairshare** in the `lsb.queues` file). Even cluster and queue administrators cannot submit jobs to a fair share queue if they do not have a share assignment.

If the default user group set in `DEFAULT_USER_GROUP` (in the `lsb.params` file) does not have shares assigned in a fair share queue, jobs can still run from the default user group, and are charged to the highest priority account the user can access in the queue. The default user group should have shares assigned in most fair share queues to ensure jobs run smoothly.

Job submitted with a user group (**bsub -G**) which is no longer valid when the job runs charge the default user group (if defined) or the highest priority account the user can access in the queue (if no default user group is defined). In such cases **bjobs -l** output shows the submission user group, along with the updated SAAP (share attribute account path).

By default, user share accounts are created for users in each user group, whether they have active jobs or not. When many user groups in the fair share policy have all as a member, the memory used creating user share accounts on startup of the `mbatchd` daemon can be noticeable. Limit the number of share accounts created to active users (and all members of the default user group) by setting `LSB_SACCT_ONE_UG=Y` in the `lsf.conf` file.

- [Configuring queue-level fair share](#)
- [Cross-queue user-based fair share](#)

User-based fair share policies configured at the queue level handle resource contention across multiple queues.

Configuring queue-level fair share

Procedure

To configure a fair share queue, define fair share in `lsb.queues` and specify a share assignment for all users of the queue:

```
fair share = USER_SHARES[[user, number_shares]...]
```

- You must specify at least one user share assignment.
- Enclose the list in square brackets, as shown.
- Enclose each user share assignment in square brackets, as shown.

Cross-queue user-based fair share

User-based fair share policies configured at the queue level handle resource contention across multiple queues.

Apply the same fair share policy to several queues

With cross-queue fair share, the same user-based fair share policy can apply to several queues at the same time. You define the fair share policy in a *parent_queue* and list *child_queues* to which the same fair share policy applies; child queues inherit the same fair share policy as your parent queue. For job scheduling purposes, this is equivalent to having one queue with one fair share tree.

In this way, if a user submits jobs to different queues, user priority is calculated by taking into account all the jobs the user has submitted across the defined queues.

To submit jobs to a fair share queue, users must be allowed to use the queue (USERS in *lsb.queues*) and must have a share assignment (FAIRSHARE in *lsb.queues*). Even cluster and queue administrators cannot submit jobs to a fair share queue if they do not have a share assignment.

User and queue priority

By default, a user has the same priority across the parent and child queues. If the same user submits several jobs to these queues, user priority is calculated by taking into account all the jobs the user has submitted across the parent-child set.

If DISPATCH_ORDER=QUEUE is set in the parent queue, jobs are dispatched according to queue priorities first, then user priority. This avoids having users with higher fair share priority getting jobs dispatched from low-priority queues.

Jobs from users with lower fair share priorities who have pending jobs in higher priority queues are dispatched before jobs in lower priority queues. Jobs in queues having the same priority are dispatched according to user priority.

Queues that are not part of the ordered cross-queue fair share can have any priority. Their priority can fall within the priority range of cross-queue fair share queues and they can be inserted between two queues using the same fair share tree.

- [Configuring cross-queue fair share](#)
- [Control job dispatch order in cross-queue fair share](#)

Configuring cross-queue fair share

About this task

- FAIRSHARE must be defined in the parent queue. If it is also defined in the queues that are listed in FAIRSHARE_QUEUES, it will be ignored.
- Cross-queue fair sharing can be defined more than once within *lsb.queues*. You can define several sets of parent-child queues. However, a queue cannot belong to more than one parent-child set. For example, you can define:
 - In parent queue *normal*: **FAIRSHARE_QUEUES=short**
 - In parent queue *priority*: **FAIRSHARE_QUEUES=night owners**You cannot, however, define *night*, *owners*, or *priority* as children in the *normal* queue; or *normal*, *short* as children in the *priority* queue; or *short*, *night*, *owners* as parent queues of their own.
- Cross-queue fair sharing cannot be used with host partition fair sharing. It is part of queue-level fair sharing.

Procedure

1. Decide to which queues in your cluster cross-queue fair sharing will apply.
For example, in your cluster you may have the queues *normal*, *priority*, *short*, and you want cross-queue fair sharing to apply only to *normal*, and *short*.
2. Define fair share policies in your parent queue.
In the queue you want to be the parent, for example *normal*, define the following in *lsb.queues*:
 - FAIRSHARE and specify a share assignment for all users of the queue.
 - FAIRSHARE_QUEUES and list child queues to which the defined fair share policy will also apply
 - PRIORITY to indicate the priority of the queue.

```

Begin Queue
QUEUE_NAME    = queue1
PRIORITY      = 30
NICE          = 20
FAIRSHARE     = USER_SHARES[[user1,100] [default,1]]
FAIRSHARE_QUEUES = queue2 queue3
DESCRIPTION   = For normal low priority jobs, running only if hosts are lightly loaded.
End Queue

```

3. In all the child queues listed in FAIRSHARE_QUEUES, define all queue values as desired.
For example:

```

Begin Queue
QUEUE_NAME    = queue2
PRIORITY      = 40
NICE          = 20
UJOB_LIMIT    = 4
PJOB_LIMIT    = 2
End Queue

Begin Queue
QUEUE_NAME    = queue3
PRIORITY      = 50
NICE          = 10
PREEMPTION    = PREEMPTIVE
QJOB_LIMIT    = 10
UJOB_LIMIT    = 1
PJOB_LIMIT    = 1
End Queue

```

Control job dispatch order in cross-queue fair share

DISPATCH_ORDER parameter (lsb.queues)

Use DISPATCH_ORDER=QUEUE in the parent queue to define an ordered cross-queue fair share set. DISPATCH_ORDER indicates that jobs are dispatched according to the order of queue priorities, not user fair share priority.

Priority range in cross-queue fair share

By default, the range of priority defined for queues in cross-queue fair share cannot be used with any other queues. The priority of queues that are not part of the cross-queue fair share cannot fall between the priority range of cross-queue fair share queues.

For example, you have four queues: queue1, queue2, queue3, and queue4. You configure cross-queue fair share for queue1, queue2, and queue3, and assign priorities of 30, 40, 50 respectively. The priority of queue4 (which is not part of the cross-queue fair share) cannot fall between 30 and 50, but it can be any number up to 29 or higher than 50. It does not matter if queue4 is a fair share queue or FCFS queue.

If DISPATCH_ORDER=QUEUE is set in the parent queue, queues that are not part of the ordered cross-queue fair share can have any priority. Their priority can fall within the priority range of cross-queue fair share queues and they can be inserted between two queues using the same fair share tree. In the example above, queue4 can have any priority, including a priority falling between the priority range of the cross-queue fair share queues (30-50).

Jobs from equal priority queues

- If two or more non-fair share queues have the same priority, their jobs are dispatched first-come, first-served based on submission time or job ID as if they come from the same queue.
- If two or more fair share queues have the same priority, jobs are dispatched in the order the queues are listed in lsb.queues.

User-based fair share

User-based fair share lets you allocate resources to users in a hierarchical manner.

By default, when shares are assigned to a group, group members compete for resources according to FCFS policy. If you use hierarchical fair share, you control the way shares that are assigned collectively are divided among group members. If groups have subgroups, you can configure additional levels of share assignments, resulting in a multi-level share tree that becomes part of the fair share policy.

How hierarchical user-based fair share affects dynamic share priority

When you use hierarchical fair share, the dynamic share priority formula does not change, but LSF measures the resource consumption for all levels of the share tree. To calculate the dynamic priority of a group, LSF uses the resource consumption of all the jobs in the queue or host partition that belong to users in the group and all its subgroups, recursively.

How hierarchical user-based fair share affects job dispatch order

LSF uses the dynamic share priority of a user or group to find out which user's job to run next. If you use hierarchical fair share, LSF works through the share tree from the top level down, and compares the dynamic priority of users and groups at each level until the user with the highest dynamic priority is a single user, or a group that has no subgroups.

- [Configure hierarchical fair share](#)
To define a hierarchical fair share policy, configure the top-level share assignment in `lsb.queues` or `lsb.hosts`, as usual. Then, for any group of users affected by the fair share policy, configure a share tree in the **UserGroup** section of `lsb.users`. This specifies how shares assigned to the group, collectively, are distributed among the individual users or subgroups.
- [Configuring a share tree](#)
- [User share assignments](#)
Both queue-level and host partition fair share use the following syntax to define how shares are assigned to users or user groups
- [Dynamic user priority](#)
LSF calculates a dynamic user priority for individual users or for a group, depending on how the shares are assigned. The priority is dynamic because it changes as soon as any variable in formula changes. By default, a user's dynamic priority gradually decreases after a job starts, and the dynamic priority immediately increases when the job finishes.

Configure hierarchical fair share

To define a hierarchical fair share policy, configure the top-level share assignment in `lsb.queues` or `lsb.hosts`, as usual. Then, for any group of users affected by the fair share policy, configure a share tree in the **UserGroup** section of `lsb.users`. This specifies how shares assigned to the group, collectively, are distributed among the individual users or subgroups.

If shares are assigned to members of any group individually, using `@`, there can be no further hierarchical fair share within that group. The shares are assigned recursively to all members of all subgroups, regardless of further share distributions defined in `lsb.users`. The group members and members of all subgroups compete for resources according to FCFS policy.

You can choose to define a hierarchical share tree for some groups but not others. If you do not define a share tree for any group or subgroup, members compete for resources according to FCFS policy.

Configuring a share tree

Procedure

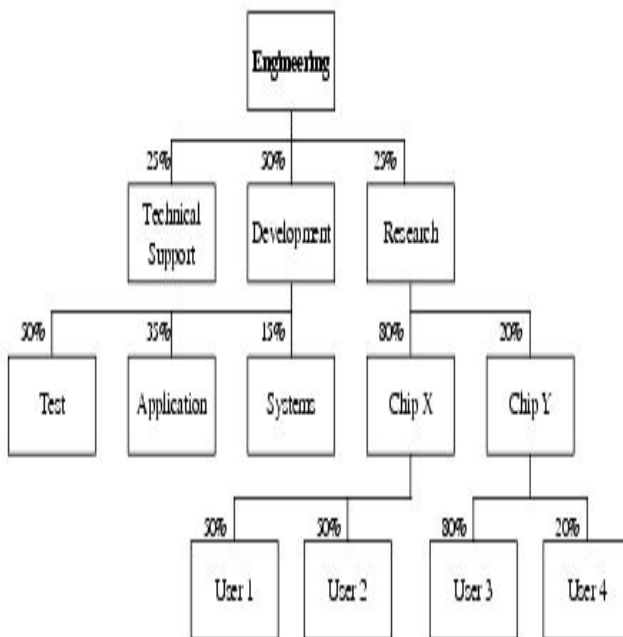
Group membership is already defined in the **UserGroup** section of lsb.users. To configure a share tree, use the **USER_SHARES** column to describe how the shares are distributed in a hierarchical manner. Use the following format.

```
Begin UserGroup
GROUP_NAME    GROUP_MEMBER    USER_SHARES
GroupB        (User1 User2)    ()
GroupC        (User3 User4)    ([User3, 3] [User4, 4])
GroupA        (GroupB GroupC User5) ([User5, 1] [default, 10])
```

- User groups must be defined before they can be used (in the **GROUP_MEMBER** column) to define other groups.
- Shares (in the **USER_SHARES** column) can only be assigned to user groups in the **GROUP_MEMBER** column.
- The keyword all refers to all users, not all user groups.
- Enclose the share assignment list in parentheses, as shown, even if you do not specify any user share assignments.

Example

An **Engineering** queue or host partition organizes users hierarchically, and divides the shares as shown; the actual number of shares assigned at each level do not matter:



The **Development** group gets the largest share (50%) of the resources in the event of contention. Shares that are assigned to the **Development** group can be further divided among the **Systems**, **Application**, and **Test** groups, which receive 15%, 35%, and 50%, respectively. At the lowest level, individual users compete for these shares as usual.

One way to measure a user's importance is to multiply their percentage of the resources at every level of the share tree. For example, **User1** is entitled to 10% of the available resources ($.50 \times .80 \times .25 = .10$) and **User3** is entitled to 4% ($.80 \times .20 \times .25 = .04$). However, if **Research** has the highest dynamic share priority among the 3 groups at the top level, and **ChipY** has a higher dynamic priority than **ChipX**, the next comparison is between **User3** and **User4**, so the importance of **User1** is not relevant. The dynamic priority of **User1** is not even calculated at this point.

Queue-based fair share

When a priority is set in a queue configuration, a high priority queue tries to dispatch as many jobs as it can before allowing lower priority queues to dispatch any job. Lower priority queues are blocked until the higher priority queue cannot dispatch any more jobs. However, it may be desirable to give some preference to lower priority queues and regulate the flow of jobs from the queue.

Queue-based fair sharing allows flexible slot allocation per queue as an alternative to absolute queue priorities by enforcing a soft job slot limit on a queue. This allows you to organize the priorities of your work and tune the number of jobs dispatched

from a queue so that no single queue monopolizes cluster resources, leaving other queues waiting to dispatch jobs.

You can balance the distribution of job slots among queues by configuring a ratio of jobs waiting to be dispatched from each queue. LSF then attempts to dispatch a certain percentage of jobs from each queue, and does not attempt to drain the highest priority queue entirely first.

When queues compete, the allocated slots per queue are kept within the limits of the configured share. If only one queue in the pool has jobs, that queue can use all the available resources and can span its usage across all hosts it could potentially run jobs on.

Manage pools of queues

You can configure your queues into a pool, which is a named group of queues using the same set of hosts. A pool is entitled to a slice of the available job slots. You can configure as many pools as you need, but each pool must use the same set of hosts. There can be queues in the cluster that do not belong to any pool yet share some hosts that are used by a pool.

How LSF allocates slots for a pool of queues

During job scheduling, LSF orders the queues within each pool based on the shares the queues are entitled to. The number of running jobs (or job slots in use) is maintained at the percentage level that is specified for the queue. When a queue has no pending jobs, leftover slots are redistributed to other queues in the pool with jobs pending.

The total number of slots in each pool is constant; it is equal to the number of slots in use plus the number of free slots to the maximum job slot limit configured either in `lsb.hosts (MAX)` or in `lsb.resources` for a host or host group. The accumulation of slots in use by the queue is used in ordering the queues for dispatch.

Job limits and host limits are enforced by the scheduler. For example, if LSF determines that a queue is eligible to run 50 jobs, but the queue has a job limit of 40 jobs, no more than 40 jobs will run. The remaining 10 job slots are redistributed among other queues belonging to the same pool, or make them available to other queues that are configured to use them.

Accumulated slots in use

As queues run the jobs allocated to them, LSF accumulates the slots each queue has used and decays this value over time, so that each queue is not allocated more slots than it deserves, and other queues in the pool have a chance to run their share of jobs.

Interaction with other scheduling policies

- Queues participating in a queue-based fair share pool cannot be preemptive or pre-emptable.
- You should not configure slot reservation (`SLOT_RESERVE`) in queues that use queue-based fair share.
- Cross-queue user-based fair share (`FAIRSHARE_QUEUES`) can undo the dispatching decisions of queue-based fair share. Cross-queue user-based fair share queues should not be part of a queue-based fair share pool.
- When `SLOT_RESERVE` and `BACKFILL` are defined (in `lsb.queues`) for the same queue, jobs in the queue cannot backfill using slots reserved by other jobs in the same queue.

Examples

Three queues using two hosts each with maximum job slot limit of 6 for a total of 12 slots to be allocated:

- `queue1` shares 50% of slots to be allocated = $2 * 6 * 0.5 = 6$
slots
- `queue2` shares 30% of slots to be allocated = $2 * 6 * 0.3 = 3.6 = 4$
slots
- `queue3` shares 20% of slots to be allocated = $2 * 6 * 0.2 = 2.4 = 3$
slots; however, since the total cannot be more than 12, `queue3` is actually allocated only 2 slots.

Four queues using two hosts each with maximum job slot limit of 6 for a total of 12 slots; `queue4` does not belong to any pool:

- `queue1` shares 50% of slots to be allocated = $2 * 6 * 0.5 = 6$
- `queue2` shares 30% of slots to be allocated = $2 * 6 * 0.3 = 3.6 =$
4

- `queue3` shares 20% of slots to be allocated = $2 * 6 * 0.2 = 2.4 = 2$
- `queue4` shares no slots with other queues

`queue4` causes the total number of slots to be less than the total free and in use by the `queue1`, `queue2`, and `queue3` that do belong to the pool. It is possible that the pool may get all its shares used up by `queue4`, and jobs from the pool will remain pending.

`queue1`, `queue2`, and `queue3` belong to one pool, `queue6`, `queue7`, and `queue8` belong to another pool, and `queue4` and `queue5` do not belong to any pool:

LSF orders the queues in the two pools from higher-priority queue to lower-priority queue (`queue1` is highest and `queue8` is lowest):

`queue1 -> queue2 -> queue3 -> queue6 -> queue7 -> queue8`

If the queue belongs to a pool, jobs are dispatched from the highest priority queue first. Queues that do not belong to any pool (`queue4` and `queue5`) are merged into this ordered list according to their priority, but LSF dispatches as many jobs from the non-pool queues as it can:

`queue1 -> queue2 -> queue3 -> queue4 -> queue5 -> queue6 -> queue7 -> queue8`

Slot allocation per queue

Configure as many pools as you need in `lsb.queues`.

Note: This feature is deprecated and might be removed in a future version of LSF.

SLOT_SHARE parameter

The `SLOT_SHARE` parameter represents the percentage of running jobs (job slots) in use from the queue. `SLOT_SHARE` must be greater than zero and less than or equal to 100.

The sum of `SLOT_SHARE` for all queues in the pool does not need to be 100%. It can be more or less, depending on your needs.

SLOT_POOL parameter

The `SLOT_POOL` parameter is the name of the pool of job slots the queue belongs to. A queue can only belong to one pool. All queues in the pool must share the same set of hosts.

MAX_SLOTS_IN_POOL parameter

The optional parameter `MAX_SLOTS_IN_POOL` sets a limit on the number of slots available for a slot pool. This parameter is defined in the first queue of the slot pool in `lsb.queues`.

USE_PRIORITY_IN_POOL parameter

The optional parameter `USE_PRIORITY_IN_POOL` enables LSF scheduling to allocate any unused slots in the pool to jobs based on the job priority across the queues in the slot pool. This parameter is defined in the first queue of the slot pool in `lsb.queues`.

Host job slot limit

The hosts that are used by the pool must have a maximum job slot limit, configured either in `lsb.hosts` (MXJ) or `lsb.resources` (HOSTS and SLOTS).

- [Configuring slot allocation per queue](#)

- [Typical slot allocation scenarios](#)
- [Configuring slot allocation per queue](#)

Configuring slot allocation per queue

Procedure

1. For each queue that uses queue-based fair share, define the following in `lsb.queues`:
 - a. `SLOT_SHARE`
 - b. `SLOT_POOL`
2. Optional: Define the following in `lsb.queues` for each queue that uses queue-based fair share:
 - a. `HOSTS` to list the hosts that can receive jobs from the queue
If no hosts are defined for the queue, the default is all hosts.
 - b. `PRIORITY` to indicate the priority of the queue.

Tip: Hosts for queue-based fair share cannot be in a host partition.
3. For each host used by the pool, define a maximum job slot limit, either in `lsb.hosts` (`MXJ`) or `lsb.resources` (`HOSTS` and `SLOTS`).

Configure two pools

The following example configures pool **A** with three queues, with different shares, using the hosts in host group **groupA**:

```
Begin Queue
QUEUE_NAME = queue1
PRIORITY   = 50
SLOT_POOL  = poolA
SLOT_SHARE = 50
HOSTS      = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue2
PRIORITY   = 48
SLOT_POOL  = poolA
SLOT_SHARE = 30
HOSTS      = groupA
...
End Queue

Begin Queue
QUEUE_NAME = queue3
PRIORITY   = 46
SLOT_POOL  = poolA
SLOT_SHARE = 20
HOSTS      = groupA
...
End Queue
```

The following configures a pool named **poolB**, with three queues with equal shares, using the hosts in host group **groupB**, setting a maximum number of slots for the pool (`MAX_SLOTS_IN_POOL`) and enabling a second round of scheduling based on job priority across the queues in the pool (`USE_PRIORITY_IN_POOL=Y`):

```
Begin Queue
QUEUE_NAME = queue4
PRIORITY   = 44
SLOT_POOL  = poolB
SLOT_SHARE = 30
HOSTS      = groupB
MAX_SLOTS_IN_POOL=128
USE_PRIORITY_IN_POOL=Y
...
```

```

End Queue
Begin Queue
QUEUE_NAME = queue5
PRIORITY   = 43
SLOT_POOL  = poolB
SLOT_SHARE = 30
HOSTS      = groupB
...
End Queue
Begin Queue
QUEUE_NAME = queue6
PRIORITY   = 42
SLOT_POOL  = poolB
SLOT_SHARE = 30
HOSTS      = groupB
...
End Queue

```

Typical slot allocation scenarios

3 queues with SLOT_SHARE 50%, 30%, 20%, with 15 job slots

This scenario has three phases:

1. All three queues have jobs running, and LSF assigns the number of slots to queues as expected: 8, 5, 2. Though queue *Genova* deserves 3 slots, the total slot assignment must be 15, so *Genova* is allocated only 2 slots:

bqueues											
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
Roma	50	Open:Active	-	-	-	-	1000	992	8	0	
Verona	48	Open:Active	-	-	-	-	995	990	5	0	
Genova	48	Open:Active	-	-	-	-	996	994	2	0	

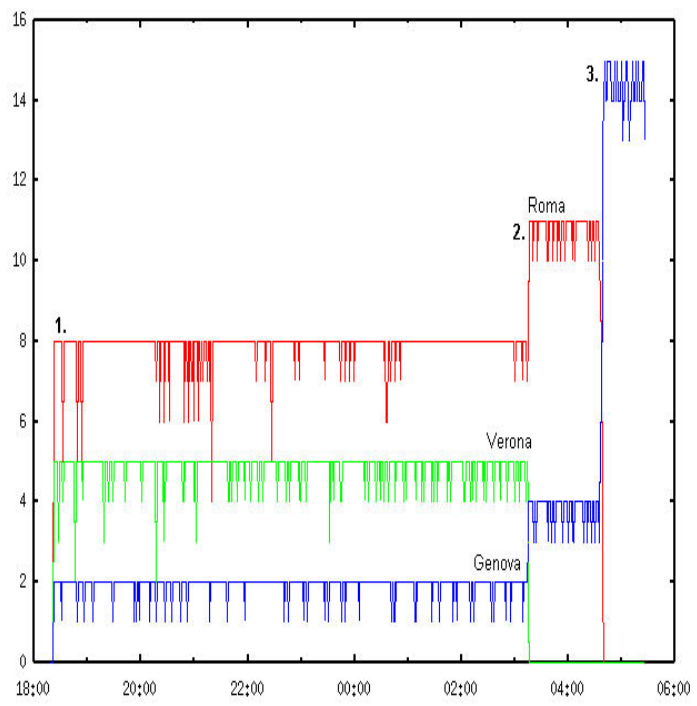
2. When queue *Verona* has done its work, queues *Roma* and *Genova* get their respective shares of 8 and 3. This leaves 4 slots to be redistributed to queues according to their shares: 50% (2 slots) to *Roma*, 20% (1 slot) to *Genova*. The one remaining slot is assigned to queue *Roma* again:

bqueues											
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
Roma	50	Open:Active	-	-	-	-	231	221	11	0	
Verona	48	Open:Active	-	-	-	-	0	0	0	0	
Genova	48	Open:Active	-	-	-	-	496	491	4	0	

3. When queues *Roma* and *Verona* have no more work to do, *Genova* can use all the available slots in the cluster:

bqueues											
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP	
Roma	50	Open:Active	-	-	-	-	0	0	0	0	
Verona	48	Open:Active	-	-	-	-	0	0	0	0	
Genova	48	Open:Active	-	-	-	-	475	460	15	0	

The following figure illustrates phases 1, 2, and 3:

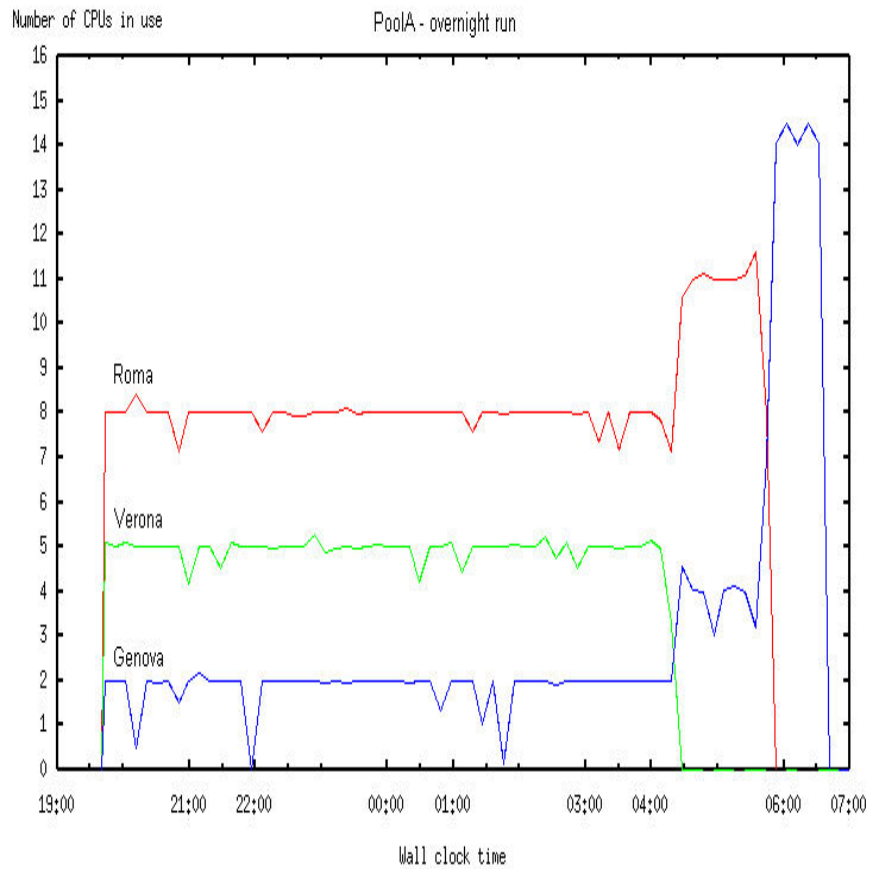


2 pools, 30 job slots, and 2 queues out of any pool

- `poolA` uses 15 slots and contains queues `Roma` (50% share, 8 slots), `Verona` (30% share, 5 slots), and `Genova` (20% share, 2 remaining slots to total 15).
- `poolB` with 15 slots containing queues `Pisa` (30% share, 5 slots), `Venezia` (30% share, 5 slots), and `Bologna` (30% share, 5 slots).
- Two other queues `Milano` and `Parma` do not belong to any pool, but they can use the hosts of `poolB`. The queues from `Milano` to `Bologna` all have the same priority.

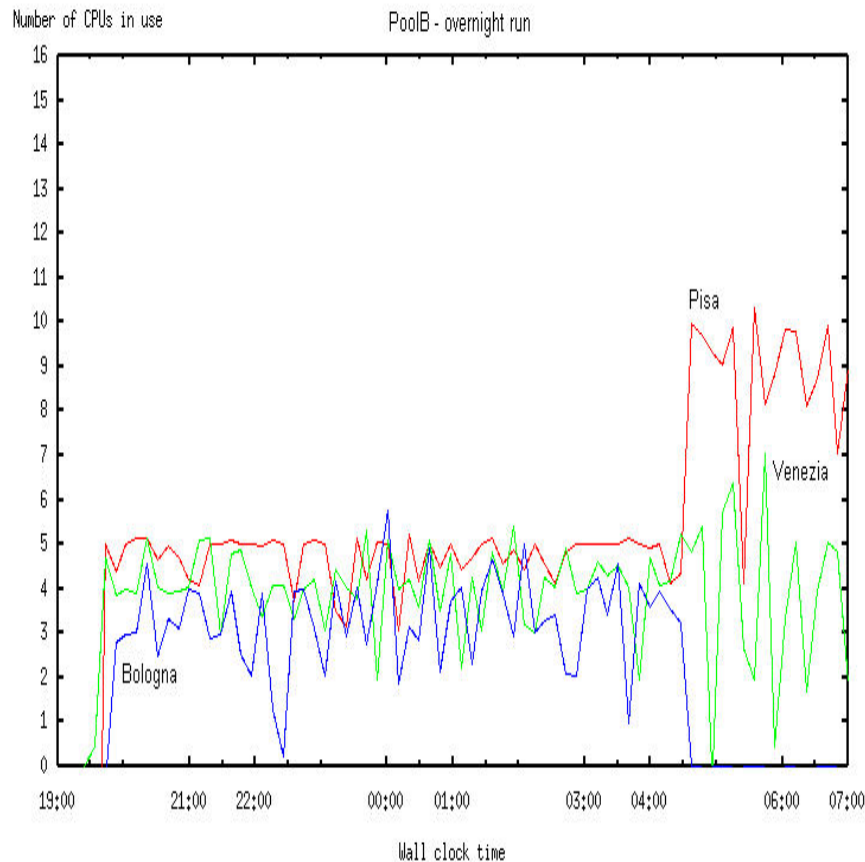
The queues `Milano` and `Parma` run very short jobs that get submitted periodically in bursts. When no jobs are running in them, the distribution of jobs looks like this:

QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open:Active	-	-	-	-	1000	992	8	0
Verona	48	Open:Active	-	-	-	-	1000	995	5	0
Genova	48	Open:Active	-	-	-	-	1000	998	2	0
Pisa	44	Open:Active	-	-	-	-	1000	995	5	0
Milano	43	Open:Active	-	-	-	-	2	2	0	0
Parma	43	Open:Active	-	-	-	-	2	2	0	0
Venezia	43	Open:Active	-	-	-	-	1000	995	5	0
Bologna	43	Open:Active	-	-	-	-	1000	995	5	0



When Milano and Parma have jobs, their higher priority reduces the share of slots free and in use by Venezia and Bologna:

QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open:Active	-	-	-	-	992	984	8	0
Verona	48	Open:Active	-	-	-	-	993	990	3	0
Genova	48	Open:Active	-	-	-	-	996	994	2	0
Pisa	44	Open:Active	-	-	-	-	995	990	5	0
Milano	43	Open:Active	-	-	-	-	10	7	3	0
Parma	43	Open:Active	-	-	-	-	11	8	3	0
Venezia	43	Open:Active	-	-	-	-	995	995	2	0
Bologna	43	Open:Active	-	-	-	-	995	995	2	0



Round-robin slot distribution: 13 queues and 2 pools

- Pool `poolA` has 3 hosts each with 7 slots for a total of 21 slots to be shared. The first 3 queues are part of the pool `poolA` sharing the CPUs with proportions 50% (11 slots), 30% (7 slots) and 20% (3 remaining slots to total 21 slots).
- The other 10 queues belong to pool `poolB`, which has 3 hosts each with 7 slots for a total of 21 slots to be shared. Each queue has 10% of the pool (3 slots).

The initial slot distribution looks like this:

bqueues										
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP
Roma	50	Open:Active	-	-	-	-	15	6	11	0
Verona	48	Open:Active	-	-	-	-	25	18	7	0
Genova	47	Open:Active	-	-	-	-	460	455	3	0
Pisa	44	Open:Active	-	-	-	-	264	261	3	0
Milano	43	Open:Active	-	-	-	-	262	259	3	0
Parma	42	Open:Active	-	-	-	-	260	257	3	0
Bologna	40	Open:Active	-	-	-	-	260	257	3	0
Sora	40	Open:Active	-	-	-	-	261	258	3	0
Ferrara	40	Open:Active	-	-	-	-	258	255	3	0
Napoli	40	Open:Active	-	-	-	-	259	256	3	0
Livorno	40	Open:Active	-	-	-	-	258	258	0	0
Palermo	40	Open:Active	-	-	-	-	256	256	0	0
Venezia	4	Open:Active	-	-	-	-	255	255	0	0

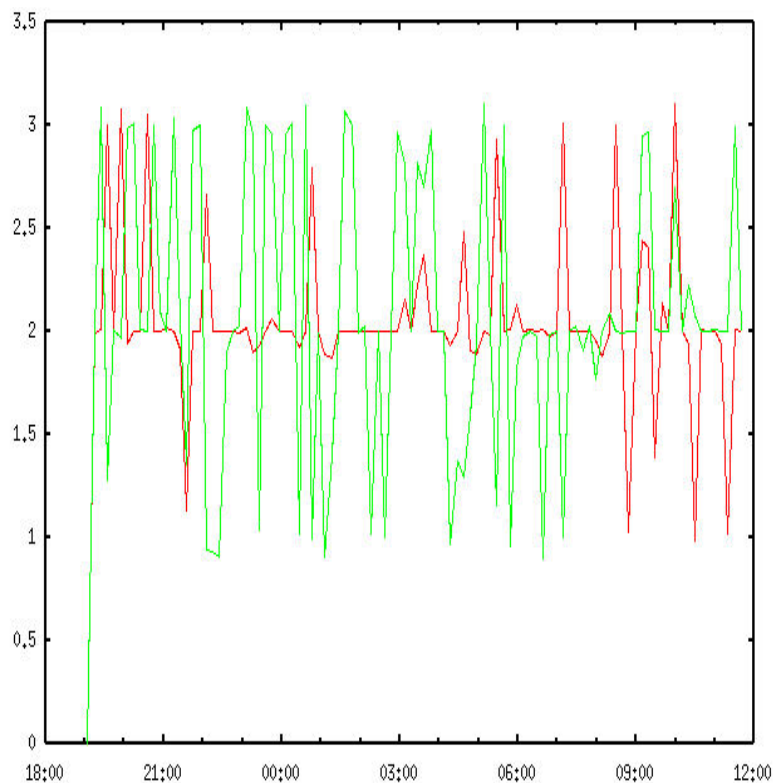
Initially, queues `Livorno`, `Palermo`, and `Venezia` in `poolB` are not assigned any slots because the first 7 higher priority queues have used all 21 slots available for allocation.

As jobs run and each queue accumulates used slots, LSF favors queues that have not run jobs yet. As jobs finish in the first 7 queues of `poolB`, slots are redistributed to the other queues that originally had no jobs (queues `Livorno`, `Palermo`, and `Venezia`). The total slot count remains 21 in all queues in `poolB`.

bqueues										
QUEUE_NAME	PRIO	STATUS	MAX	JL/U	JL/P	JL/H	NJOBS	PEND	RUN	SUSP

Roma	50	Open:Active	-	-	-	-	15	6	9	0
Verona	48	Open:Active	-	-	-	-	25	18	7	0
Genova	47	Open:Active	-	-	-	-	460	455	5	0
Pisa	44	Open:Active	-	-	-	-	263	261	2	0
Milano	43	Open:Active	-	-	-	-	261	259	2	0
Parma	42	Open:Active	-	-	-	-	259	257	2	0
Bologna	40	Open:Active	-	-	-	-	259	257	2	0
Sora	40	Open:Active	-	-	-	-	260	258	2	0
Ferrara	40	Open:Active	-	-	-	-	257	255	2	0
Napoli	40	Open:Active	-	-	-	-	258	256	2	0
Livorno	40	Open:Active	-	-	-	-	258	256	2	0
Palermo	40	Open:Active	-	-	-	-	256	253	3	0
Venezia	4	Open:Active	-	-	-	-	255	253	2	0

The following figure illustrates the round-robin distribution of slot allocations between queues Livorno and Palermo:



How LSF rebalances slot usage

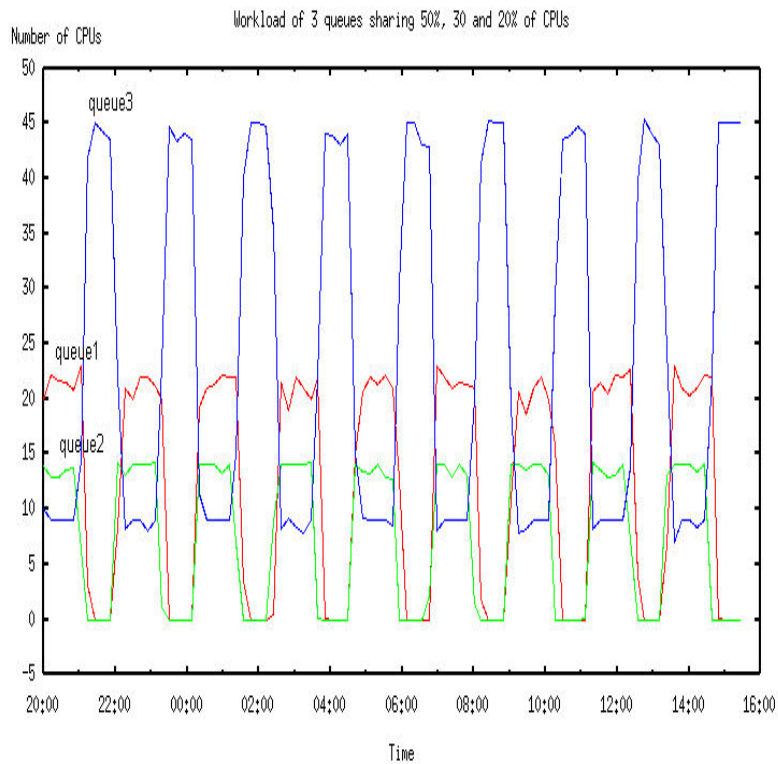
In the following examples, job runtime is not equal, but varies randomly over time.

3 queues in one pool with 50%, 30%, 20% shares

A pool configures 3 queues:

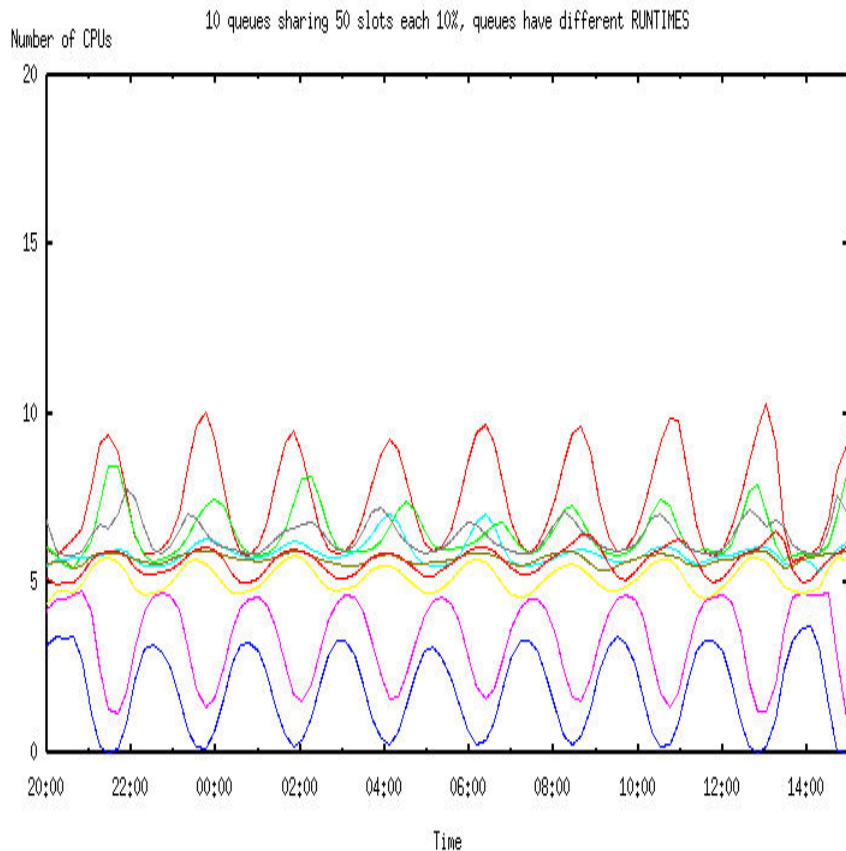
- queue1 50% with short-running jobs
- queue2 20% with short-running jobs
- queue3 30% with longer running jobs

As queue1 and queue2 finish their jobs, the number of jobs in queue3 expands, and as queue1 and queue2 get more work, LSF rebalances the usage:



10 queues sharing 10% each of 50 slots

In this example, *queue1* (the curve with the highest peaks) has the longer running jobs and so has less accumulated slots in use over time. LSF accordingly rebalances the load when all queues compete for jobs to maintain a configured 10% usage share.



Users affected by multiple fair share policies

If you belong to multiple user groups, which are controlled by different fair share policies, each group probably has a different dynamic share priority at any given time. By default, if any one of these groups becomes the highest priority user, you could be the highest priority user in that group, and LSF would attempt to place your job.

To restrict the number of fair share policies that will affect your job, submit your job and specify a single user group that your job will belong to, for the purposes of fair share scheduling. LSF will not attempt to dispatch this job unless the group you specified is the highest priority user. If you become the highest priority user because of some other share assignment, another one of your jobs might be dispatched, but not this one.

- [Submitting a job and specify a user group](#)
Associate a job with a user group for fair share scheduling.

Submitting a job and specify a user group

Associate a job with a user group for fair share scheduling.

Procedure

Use **bsub -G** and specify a group that you belong to.

For example, `User1` shares resources with `groupA` and `groupB`. `User1` is also a member of `groupA`, but not any other groups.

`User1` submits a job: **bsub sleep 100**.

By default, the job could be considered for dispatch if either `User1` or `GroupA` has highest dynamic share priority.

`User1` submits a job and associates the job with `GroupA`:

```
bsub -G groupA sleep 100
```

If `User1` is the highest priority user, this job will not be considered.

- `User1` can only associate the job with a group of which they are a member.
- `User1` cannot associate the job with his individual user account because **bsub -G** only accepts group names.

Example with hierarchical fair share

In the share tree, `User1` shares resources with `GroupA` at the top level. `GroupA` has 2 subgroups, `B` and `C`. `GroupC` has 1 subgroup, `GroupD`. `User1` also belongs to `GroupB` and `GroupC`.

`User1` submits a job:

```
bsub sleep 100
```

By default, the job could be considered for dispatch if either `User1`, `GroupB`, or `GroupC` has highest dynamic share priority.

`User1` submits a job and associates the job with `GroupB`:

```
bsub -G groupB sleep 100
```

If `User1` or `GroupC` is the highest priority user, this job will not be considered.

- `User1` cannot associate the job with `GroupC`, because `GroupC` includes a subgroup.
- `User1` cannot associate the job with his individual user account because **bsub -G** only accepts group names.

Ways to configure fair share

- [Chargeback fair share](#)
Chargeback fair share lets competing users share the same hardware resources according to a fixed ratio. Each user is entitled to a specified portion of the available resources.
- [Equal share](#)
Equal share balances resource usage equally between users.
- [Priority user and static priority fair share](#)
There are two ways to configure fair share so that a more important user's job always overrides the job of a less important user, regardless of resource use.
- [Host partition fair share](#)
Host partition fair share balances resource usage across the entire cluster according to one single fair share policy. Resources that are used in one queue affect job dispatch order in another queue.
- [GPU runtime fair share](#)
GPU runtime fair share lets you consider GPU run time and historical run time as weighting factors in the dynamic priority calculation.

Host partition fair share

Host partition fair share balances resource usage across the entire cluster according to one single fair share policy. Resources that are used in one queue affect job dispatch order in another queue.

If two users compete for resources, their dynamic share priority is the same in every queue.

- [Configuring host partition fair share](#)

Configuring host partition fair share

Procedure

Use the keyword `all` to configure a single partition that includes all the hosts in the cluster.

```
Begin HostPartition
HPART_NAME =GlobalPartition
HOSTS = all
USER_SHARES = [groupA@, 3] [groupB, 7] [default, 1]
End HostPartition
```

Chargeback fair share

Chargeback fair share lets competing users share the same hardware resources according to a fixed ratio. Each user is entitled to a specified portion of the available resources.

If two users compete for resources, the most important user is entitled to more resources.

- [Configuring chargeback fair share](#)

Configuring chargeback fair share

Procedure

To configure chargeback fair share, put competing users in separate user groups and assign a fair number of shares to each group.

Example

About this task

Suppose that two departments contributed to the purchase of a large system. The engineering department contributed 70 percent of the cost, and the accounting department 30 percent. Each department wants to get their money's worth from the system.

Procedure

1. Define two user groups in lsb.users, one listing all the engineers, and one listing all the accountants.

```
Begin UserGroup
Group_Name    Group_Member
eng_users     (user6 user4)
acct_users    (user2 user5)
End UserGroup
```

2. Configure a host partition for the host, and assign the shares appropriately.

```
Begin HostPartition
HPART_NAME = big_servers
HOSTS = hostH
USER_SHARES = [eng_users, 7] [acct_users, 3]
End HostPartition
```

Equal share

Equal share balances resource usage equally between users.

- [Configuring equal share](#)

Configuring equal share

Procedure

To configure equal share, use the keyword default to define an equal share for every user.

```
Begin HostPartition
HPART_NAME = equal_share_partition
HOSTS = all
USER_SHARES = [default, 1]
End HostPartition
```

Priority user and static priority fair share

There are two ways to configure fair share so that a more important user's job always overrides the job of a less important user, regardless of resource use.

Priority user fair share

Dynamic priority is calculated as usual, but more important and less important users are assigned a drastically different number of shares, so that resource use has virtually no effect on the dynamic priority: the user with the overwhelming majority of shares always goes first. However, if two users have a similar or equal number of shares, their resource use still determines which of them goes first. This is useful for isolating a group of high-priority or low-priority users, while allowing other fair share policies to operate as usual most of the time.

Static priority fair share

Dynamic priority is no longer dynamic because resource use is ignored. The user with the most shares always goes first. This is useful to configure multiple users in a descending order of priority.

- [Configuring priority user fair share](#)
A queue is shared by key users and other users. Priority user fair sharing gives priority to important users, so their jobs override the jobs of other users. You can still use fair share policies to balance resources among each group of users.
- [Configuring static priority fair share](#)
Static priority fair share assigns resources to the user with the most shares. Resource usage is ignored.

Configuring priority user fair share

A queue is shared by key users and other users. Priority user fair sharing gives priority to important users, so their jobs override the jobs of other users. You can still use fair share policies to balance resources among each group of users.

About this task

If two users compete for resources, and one of them is a priority user, the priority user's job always runs first.

Procedure

1. Define a user group for priority users in `lsb.users`, naming it accordingly.
For example, `key_users`.
2. Configure fair share and assign the overwhelming majority of shares to the key users:

```
Begin Queue
QUEUE_NAME = production
FAIRSHARE = USER_SHARES[ [key_users@, 2000] [others, 1] ]
...
End Queue
```

In the preceding example, key users have 2000 shares each, while other users together have only one share. This makes it virtually impossible for other users' jobs to get dispatched unless none of the users in the `key_users` group has jobs waiting to run.

If you want the same fair share policy to apply to jobs from all queues, configure host partition fair share in a similar way.

Configuring static priority fair share

Static priority fair share assigns resources to the user with the most shares. Resource usage is ignored.

Procedure

To implement static priority fair share, edit the `lsb.params` file and set all the weighting factors that are used in the dynamic priority formula to 0 (zero):

- Set `CPU_TIME_FACTOR` to 0
- Set `RUN_TIME_FACTOR` to 0
- Set `RUN_JOB_FACTOR` to 0
- Set `COMMITTED_RUN_TIME_FACTOR` to 0
- Set `FAIRSHARE_ADJUSTMENT_FACTOR` to 0
- Set `GPU_RUN_TIME_FACTOR` to 0

This results in `dynamic priority = number_shares / 0.01`; that is, if the denominator in the dynamic priority calculation is less than 0.01, IBM® Spectrum LSF rounds up to 0.01.

Results

If two users compete for resources, the most important user's job always runs first.

GPU runtime fair share

GPU runtime fair share lets you consider GPU run time and historical run time as weighting factors in the dynamic priority calculation.

- [Configuring GPU run time](#)

Configuring GPU run time

Procedure

1. Set a value for the GPU_RUN_TIME_FACTOR parameter for the queue in lsb.queues or for the cluster in lsb.params.
2. To enable historical GPU run time of finished jobs, specify `ENABLE_GPU_HIST_RUN_TIME=Y` for the queue in lsb.queues or for the cluster in lsb.params.

Enabling historical GPU time ensures that the user's priority does not increase significantly after a GPU job finishes.

Results

If you set the GPU run time factor and enabled the use of GPU historical run time, the dynamic priority is calculated according to the following formula:

$$\text{dynamic priority} = \text{number_shares} / (\text{cpu_time} * \text{CPU_TIME_FACTOR} + (\text{historical_run_time} + \text{run_time}) * \text{RUN_TIME_FACTOR} + (\text{committed_run_time} - \text{run_time}) * \text{COMMITTED_RUN_TIME_FACTOR} + (1 + \text{job_slots}) * \text{RUN_JOB_FACTOR} + \text{fairshare_adjustment}(\text{struct} * \text{shareAdjustPair}) * \text{FAIRSHARE_ADJUSTMENT_FACTOR}) + ((\text{historical_gpu_run_time} + \text{gpu_run_time}) * \text{ngpus_physical}) * \text{GPU_RUN_TIME_FACTOR}$$

gpu_run_time—The run time requested at GPU job submission with the **-gpu** option of **bsub**, the queue or application profile configuration with the GPU_REQ parameter, or the cluster configuration with the LSB_GPU_REQ parameter.

- For jobs that ask for exclusive use of a GPU, *gpu_run_time* is the same as the job's run time and *ngpus_physical* is the value of the requested *ngpus_physical* in the job's effective RES_REQ string.
- For jobs that ask for an exclusive host (with the **bsub -x** option), the *gpu_run_time* is the same as the job's run time and *ngpus_physical* is the number of GPUs on the execution host.
- For jobs that ask for an exclusive compute unit (**bsub -R "cu[excl]"** option), the *gpu_run_time* is the same as the job's run time and *ngpus_physical* is the number of GPUs or all the execution hosts in the compute unit.
- For jobs that ask for shared mode GPUs, the *gpu_run_time* is the job's run time divided by the number of jobs that are using the GPU, and *ngpus_physical* is the value of the requested *ngpus_physical* in the job's effective RES_REQ string.

historical_run_time— If `ENABLE_GPU_HIST_RUN_TIME` is defined in the lsb.params file, the *historical_run_time* is the same as the job's run time (measured in hours) of finished GPU jobs, and a decay factor from time to time based on HIST_HOURS in the lsb.params file (5 hours by default).

Re-sizable jobs and fair share

Re-sizable jobs submitting into fair share queues or host partitions are subject to fair share scheduling policies. The dynamic priority of the user who submitted the job is the most important criterion. LSF treats pending resize allocation requests as a regular job and enforces the fair share user priority policy to schedule them.

The dynamic priority of users depends on the following:

- Their share assignment
- The slots their jobs are currently consuming
- The resources their jobs consumed in the past
- The adjustment made by the fair share plug-in (`libfairshareadjust.*`)

Re-sizable job allocation changes affect the user priority calculation if the `RUN_JOB_FACTOR` or `FAIRSHARE_ADJUSTMENT_FACTOR` is greater than zero. Resize add requests increase number of slots in use and decrease user priority. Resize release requests decrease number of slots in use, and increase user priority. The faster a re-sizable job grows, the lower the user priority is, the less likely a pending allocation request can get more slots.

Note: The effect of re-sizable job allocation changes when the `Fairshare_adjustment_factor` is greater than zero, and depends on the user-defined fair share adjustment plug-in (`libfairshareadjust.*`).

After job allocation changes, running the `bqueues` and `bhpart` display updated user priority.

Job count based fair share scheduling

Job count based fair share scheduling uses the number of jobs in the fair share scheduling algorithm instead of the number of job slots.

You can potentially use job count based fair share scheduling if you set up a queue to run one job per host. In this situation, a job can use the entire job slots, memory, or other resources that are available in the host. Therefore, only the number of jobs that a user is running is under consideration for the fair share scheduling algorithm instead of job slots because each job can use all the job slots on a host.

The `FAIRSHARE_JOB_COUNT` parameter in the `lsb.params` file configures job count based fair share scheduling. If this parameter is set to `Y` or `y`, LSF uses the number of jobs in the fair share scheduling algorithm instead of job slots. This setting only affects the value that is multiplied by the job slots weighting factor in the fair share algorithm (`RUN_JOB_FACTOR` parameter in the `lsb.params` file).

Job checkpoint and restart

Optimize resource usage with job checkpoint and restart to stop jobs and then restart them from the point at which they stopped.

LSF can periodically capture the state of a running job and the data required to restart it. This feature provides fault tolerance and allows LSF administrators and users to migrate jobs from one host to another to achieve load balancing.

- [About job checkpoint and restart](#)
Checkpointing enables LSF users to restart a job on the same execution host or to migrate a job to a different execution host. LSF controls checkpointing and restart by means of interfaces named `echkpt` and `erestart`.
- [Configuration to enable job checkpoint and restart](#)
The job checkpoint and restart feature requires that a job be made checkpoint-able at the job or queue level. LSF users can make jobs checkpoint-able by submitting jobs using `bsub -k` and specifying a checkpoint directory. Queue administrators can make all jobs in a queue checkpoint-able by specifying a checkpoint directory for the queue.
- [Job checkpoint and restart behavior](#)
LSF invokes the `echkpt` interface when a job meets these conditions:
- [Configuration to modify job checkpoint and restart](#)
There are configuration parameters that modify various aspects of job checkpoint and restart behavior by:
- [Job checkpoint and restart commands](#)

About job checkpoint and restart

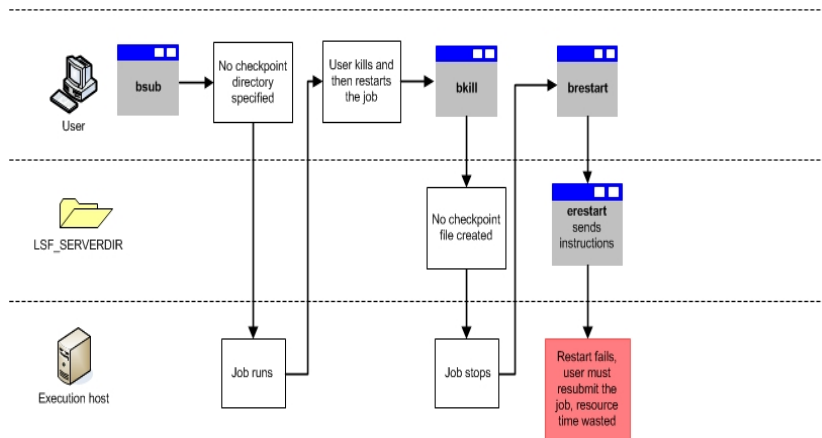
Checkpointing enables LSF users to restart a job on the same execution host or to migrate a job to a different execution host. LSF controls checkpointing and restart by means of interfaces named **echkptnt** and **erestart**.

When LSF checkpoints a job, the **echkptnt** interface creates a checkpoint file in the directory *checkpoint_dir/job_ID*, and then checkpoints and resumes the job. The job continues to run, even if checkpointing fails.

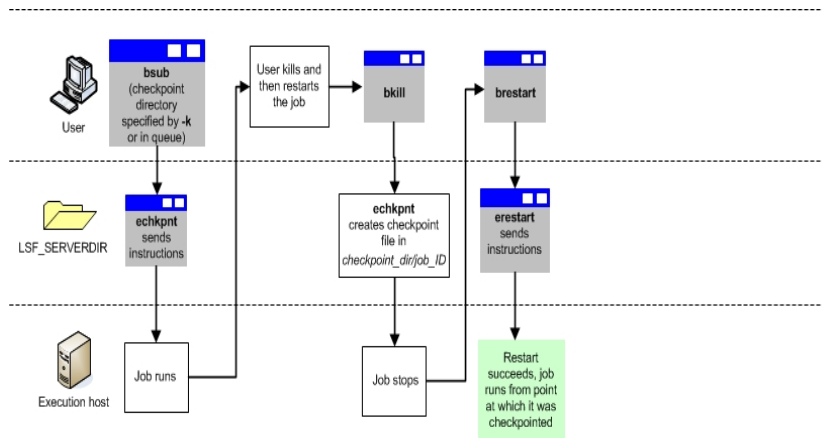
When LSF restarts a stopped job, the **erestart** interface recovers job state information from the checkpoint file, including information about the execution environment, and restarts the job from the point at which the job stopped. At job restart, LSF

1. Resubmits the job to its original queue and assigns a new job ID
2. Dispatches the job when a suitable host becomes available (not necessarily the original execution host)
3. Re-creates the execution environment based on information from the checkpoint file
4. Restarts the job from its most recent checkpoint

Default behavior (job checkpoint and restart not enabled)



With job checkpoint and restart enabled



Application-level checkpoint and restart

Different applications have different checkpointing implementations that require the use of customized external executables (**echkptnt.application** and **erestart.application**). Application-level checkpoint and restart enables you to configure LSF to use specific **echkptnt.application** and **erestart.application** executables for a job, queue, or cluster. You can write customized checkpoint and restart executables for each application that you use.

LSF uses a combination of corresponding checkpoint and restart executables. For example, if you use **echkptnt.fluent** to checkpoint a particular job, LSF will use **erestart.fluent** to restart the checkpointed job. You cannot override this behavior or configure LSF to use a specific restart executable.

Scope

Applicability	Details
Job types	<ul style="list-style-type: none">• Non-interactive batch jobs submitted with bsub or bmod• Non-interactive batch jobs, including chunk jobs, checkpointed with bchkpnt• Non-interactive batch jobs migrated with bmig• Non-interactive batch jobs restarted with brestart
Dependencies	<ul style="list-style-type: none">• UNIX and Windows user accounts must be valid on all hosts in the cluster, or the correct type of account mapping must be enabled.<ul style="list-style-type: none">◦ For a mixed UNIX/Windows cluster, UNIX/Windows user account mapping must be enabled.◦ For a cluster with a non-uniform user name space, between-host account mapping must be enabled.◦ For a MultiCluster environment with a non-uniform user name space, cross-cluster user account mapping must be enabled.• The checkpoint and restart executables run under the user account of the user who submits the job. User accounts must have the correct permissions to<ul style="list-style-type: none">◦ Successfully run executables located in LSF_SERVERDIR or LSB_ECHKPNT_METHOD_DIR◦ Write to the checkpoint directory• The erestart.application executable must have access to the original command line used to submit the job.• To allow restart of a checkpointed job on a different host than the host on which the job originally ran, both the original and the new hosts must:<ul style="list-style-type: none">◦ Be binary compatible◦ Have network connectivity and read/execute permissions to the checkpoint and restart executables (in LSF_SERVERDIR by default)◦ Have network connectivity and read/write permissions to the checkpoint directory and the checkpoint file◦ Have access to all files open during job execution so that LSF can locate them using an absolute path name
Limitations	<ul style="list-style-type: none">• bmod cannot change the echkpnt and erestart executables associated with a job.• Linux 32, AIX, and HP platforms with NFS (network file systems), checkpoint directories (including path and file name) must be shorter than 1000 characters.• Linux 64 with NFS (network file systems), checkpoint directories (including path and file name) must be shorter than 2000 characters.

Configuration to enable job checkpoint and restart

The job checkpoint and restart feature requires that a job be made checkpoint-able at the job or queue level. LSF users can make jobs checkpoint-able by submitting jobs using **bsub -k** and specifying a checkpoint directory. Queue administrators can make all jobs in a queue checkpoint-able by specifying a checkpoint directory for the queue.

Configuration file	Parameter and syntax	Behavior
lsb.queues	CHKPNT= <i>chkpnt_dir</i> [<i>chkpnt_period</i>]	<ul style="list-style-type: none"> All jobs submitted to the queue are checkpointable. LSF writes the checkpoint files, which contain job state information, to the checkpoint directory. The checkpoint directory can contain checkpoint files for multiple jobs. <ul style="list-style-type: none"> The specified checkpoint directory must already exist. LSF will not create the checkpoint directory. The user account that submits the job must have read and write permissions for the checkpoint directory. For the job to restart on another execution host, both the original and new hosts must have network connectivity to the checkpoint directory. If the queue administrator specifies a checkpoint period, in minutes, LSF creates a checkpoint file every <i>chkpnt_period</i> during job execution. Note: There is no default value for checkpoint period. You must specify a checkpoint period if you want to enable periodic checkpointing. If a user specifies a checkpoint directory and checkpoint period at the job level with bsub -k, the job-level values override the queue-level values. The file path of the checkpoint directory can contain up to 4000 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.
lsb.applications		

Configuration to enable kernel-level checkpoint and restart

Kernel-level checkpoint and restart is enabled by default. LSF users make a job checkpointable by either submitting a job using **bsub -k** and specifying a checkpoint directory or by submitting a job to a queue that defines a checkpoint directory for the CHKPNT parameter.

Configuration to enable application-level checkpoint and restart

Application-level checkpointing requires the presence of at least one **echkpnt.application** executable in the directory specified by the parameter LSF_SERVERDIR in lsf.conf. Each **echkpnt.application** must have a corresponding **erestart.application**.

Important:

The **erestart.application** executable must:

- Have access to the command line used to submit or modify the job
- Exit with a return value without running an application; the **erestart** interface runs the application to restart the job

Executable file	UNIX naming convention	Windows naming convention
echkpnt	LSF_SERVERDIR/ <i>echkpnt.application</i>	LSF_SERVERDIR\echkpnt. <i>application.exe</i> LSF_SERVERDIR\echkpnt. <i>application.bat</i>
erestart	LSF_SERVERDIR/ <i>erestart.application</i>	LSF_SERVERDIR\erestart. <i>application.exe</i> LSF_SERVERDIR\erestart. <i>application.bat</i>

Restriction:

The names **echkpnt.default** and **erestart.default** are reserved. Do not use these names for application-level checkpoint and restart executables.

Valid file names contain only alphanumeric characters, underscores (_), and hyphens (-).

For application-level checkpoint and restart, once the LSF_SERVERDIR contains one or more checkpoint and restart executables, users can specify the external checkpoint executable associated with each checkpointable job they submit. At restart, LSF invokes the corresponding external restart executable.

Requirements for application-level checkpoint and restart executables

- The executables must be written in C or Fortran.
- The directory/name combinations must be unique within the cluster. For example, you can write two different checkpoint executables with the name **echkptn.fluent** and save them as LSF_SERVERDIR/echkptn.fluent and my_execs/echkptn.fluent. To run checkpoint and restart executables from a directory other than LSF_SERVERDIR, you must configure the parameter LSB_ECHKPNT_METHOD_DIR in lsf.conf.
- Your executables must return the following values.
 - An **echkptn.application** must return a value of 0 when checkpointing succeeds and a non-zero value when checkpointing fails.
 - The **erestart** interface provided with LSF restarts the job using a restart command that **erestart.application** writes to a file. The return value indicates whether **erestart.application** successfully writes the parameter definition LSB_RESTART_CMD=restart_command to the file *checkpoint_dir/job_ID/.restart_cmd*.
 - A non-zero value indicates that **erestart.application** failed to write to the .restart_cmd file.
 - A return value of 0 indicates that **erestart.application** successfully wrote to the .restart_cmd file, or that the executable intentionally did not write to the file.
- Your executables must recognize the syntax used by the **echkptn** and **erestart** interfaces, which communicate with your executables by means of a common syntax.

- **echkptn.application** syntax:

```
echkptn [-c] [-f] [-k | -s] [-d checkpoint_dir] [-x] process_group_ID
```

Restriction:

The -k and -s options are mutually exclusive.

- **erestart.application** syntax:

```
erestart [-c] [-f] checkpoint_dir
```

Option or variable	Description	Operating systems
-c	Copies all files in use by the checkpointed process to the checkpoint directory.	Some
-f	Forces a job to be checkpointed even under non-checkpointable conditions, which are specific to the checkpoint implementation used. This option could create checkpoint files that do not provide for successful restart.	Some
-k	Kills a job after successful checkpointing. If checkpoint fails, the job continues to run.	All operating systems that LSF supports
-s	Stops a job after successful checkpointing. If checkpoint fails, the job continues to run.	Some
-d <i>checkpoint_dir</i>	Specifies the checkpoint directory as a relative or absolute path.	All operating systems that LSF supports
-x	Identifies the cpr (checkpoint and restart) process as type HID. This identifies the set of processes to checkpoint as a process hierarchy (tree) rooted at the current PID.	Some
<i>process_group_ID</i>	ID of the process or process group to checkpoint.	All operating systems that LSF supports

Job checkpoint and restart behavior

LSF invokes the **echkptnt** interface when a job meets these conditions:

- Automatically check-pointed based on a configured checkpoint period
- Manually check-pointed with **bchkptnt**
- Migrated to a new host with **bmig**

After checkpointing, LSF invokes the **erestart** interface to restart the job. LSF also invokes the **erestart** interface when a user

- Manually restarts a job using **brestart**
- Migrates the job to a new host using **bmig**

All checkpoint and restart executables run under the user account of the user who submits the job.

Note: By default, LSF redirects standard error and standard output to /dev/null and discards the data.

Checkpoint directory and files

LSF identifies checkpoint files by the checkpoint directory and job ID. For example:

```
bsub -k my_dir
Job <123> is submitted to default queue <default>
```

LSF writes the checkpoint file to my_dir/123.

LSF maintains all of the checkpoint files for a single job in one location. When a job restarts, LSF creates both a new subdirectory based on the new job ID and a symbolic link from the old to the new directory. For example, when job 123 restarts on a new host as job 456, LSF creates my_dir/456 and a symbolic link from my_dir/123 to my_dir/456.

The file path of the checkpoint directory can contain up to 4000 characters for UNIX and Linux, or up to 255 characters for Windows, including the directory and file name.

Precedence of job, queue, application, and cluster-level checkpoint values

LSF handles checkpoint and restart values as follows:

1. *Checkpoint directory and checkpoint period*—values specified at the job level override values for the queue. Values specified in an application profile setting overrides queue level configuration.
If checkpoint-related configuration is specified in the queue, application profile, and at job level:
 - Application-level and job-level parameters are merged. If the same parameter is defined at both job-level and in the application profile, the job-level value overrides the application profile value.
 - The merged result of job-level and application profile settings override queue-level configuration.
2. *Checkpoint and restart executables*—the value for *checkpoint_method* specified at the job level overrides the application-level *CHKPNT_METHOD*, and the cluster-level value for *LSB_ECHKPNT_METHOD* specified in *lsf.conf* or as an environment variable.
3. *Configuration parameters and environment variables*—values specified as environment variables override the values specified in *lsf.conf*

If the command line is...	And...	Then...
bsub -k "my_dir 240"	In lsb.queues, CHKPNT=other_dir 360	<ul style="list-style-type: none">• LSF saves the checkpoint file to my_dir/job_ID every 240 minutes

If the command line is...	And...	Then...
<code>bsub -k "my_dir fluent"</code>	In <code>lsf.conf</code> , <code>LSB_ECHKPNT_METHOD=</code> <code>myapp</code>	<ul style="list-style-type: none"> LSF invokes echkpnt.fluent at job checkpoint and erestart.fluent at job restart
<code>bsub -k "my_dir"</code>	In <code>lsb.applications</code> , <code>CHKPNT_PERIOD=360</code>	<ul style="list-style-type: none"> LSF saves the checkpoint file to <code>my_dir/job_ID</code> every 360 minutes
<code>bsub -k "240"</code>	In <code>lsb.applications</code> , <code>CHKPNT_DIR=app_dir</code> <code>CHKPNT_PERIOD=360</code> In <code>lsb.queue</code> s, <code>CHKPNT=other_dir</code>	<ul style="list-style-type: none"> LSF saves the checkpoint file to <code>app_dir/job_ID</code> every 240 minutes

Configuration to modify job checkpoint and restart

There are configuration parameters that modify various aspects of job checkpoint and restart behavior by:

- Specifying mandatory application-level checkpoint and restart executables that apply to all checkpoint-able batch jobs in the cluster
- Specifying the directory that contains customized application-level checkpoint and restart executables
- Saving standard output and standard error to files in the checkpoint directory
- Automatically checkpointing jobs before suspending or terminating them
- For Cray systems only, copying all open job files to the checkpoint directory

Configuration to specify mandatory application-level executables

You can specify mandatory checkpoint and restart executables by defining the parameter `LSB_ECHKPNT_METHOD` in `lsf.conf` or as an environment variable.

Configuration file	Parameter and syntax	Behavior
<code>lsf.conf</code>	<code>LSB_ECHKPNT_METHOD=</code> <code>"echkpnt_application"</code>	<ul style="list-style-type: none"> The specified echkpnt runs for all batch jobs submitted to the cluster. At restart, the corresponding erestart runs. For example, if <code>LSB_ECHKPNT_METHOD=fluent</code>, at checkpoint, LSF runs echkpnt.fluent and at restart, LSF runs erestart.fluent. If an LSF user specifies a different <code>echkpnt_application</code> at the job level using bsub -k or bmod -k, the job level value overrides the value in <code>lsf.conf</code>.

Configuration to specify the directory for application-level executables

By default, LSF looks for application-level checkpoint and restart executables in `LSF_SERVERDIR`. You can modify this behavior by specifying a different directory as an environment variable or in `lsf.conf`.

Configuration file	Parameter and syntax	Behavior
--------------------	----------------------	----------

Configuration file	Parameter and syntax	Behavior
lsf.conf	LSB_ECHKPNT_METHOD_DIR= <i>path</i>	<ul style="list-style-type: none"> Specifies the absolute path to the directory that contains the echkpnt.application and erestart.application executables User accounts that run these executables must have the correct permissions for the LSB_ECHKPNT_METHOD_DIR directory.

Configuration to save standard output and standard error

By default, LSF redirects the standard output and standard error from checkpoint and restart executables to /dev/null and discards the data. You can modify this behavior by defining the parameter LSB_ECHKPNT_KEEP_OUTPUT as an environment variable or in lsf.conf.

Configuration file	Parameter and syntax	Behavior
lsf.conf	LSB_ECHKPNT_KEEP_OUTPUT=Y y	<ul style="list-style-type: none"> The stdout and stderr for echkpnt.application or echkpnt.default are redirected to <i>checkpoint_dir/job_ID/</i> <ul style="list-style-type: none"> echkpnt.out echkpnt.err The stdout and stderr for erestart.application or erestart.default are redirected to <i>checkpoint_dir/job_ID/</i> <ul style="list-style-type: none"> erestart.out erestart.err

Configuration to checkpoint jobs before suspending or terminating them

LSF administrators can configure LSF at the queue level to checkpoint jobs before suspending or terminating them.

Configuration file	Parameter and syntax	Behavior
lsb.queue	JOB_CONTROLS=SUSPEND CHKPNT TERMINATE	<ul style="list-style-type: none"> LSF checkpoints jobs before suspending or terminating them When suspending a job, LSF checkpoints the job and then stops it by sending the SIGSTOP signal When terminating a job, LSF checkpoints the job and then kills it

Configuration to copy open job files to the checkpoint directory

For hosts that use the Cray operating system, LSF administrators can configure LSF at the host level to copy all open job files to the checkpoint directory every time the job is checkpointed.

Configuration file	Parameter and syntax	Behavior
lsb.hosts	HOST_NAME CHKPNT <i>host_name</i> C	<ul style="list-style-type: none"> LSF copies all open job files to the checkpoint directory when a job is checkpointed

Job checkpoint and restart commands

Commands for submission

Command	Description
bsub -k " <i>checkpoint_dir</i> [<i>checkpoint_period</i>] [<i>method=echkpt_application</i>]"	<ul style="list-style-type: none">• Specifies a relative or absolute path for the checkpoint directory and makes the job checkpointable.• If the specified checkpoint directory does not already exist, LSF creates the checkpoint directory.• If a user specifies a checkpoint period (in minutes), LSF creates a checkpoint file every <i>chkpt_period</i> during job execution.• The command-line values for the checkpoint directory and checkpoint period override the values specified for the queue.• If a user specifies an <i>echkpt_application</i>, LSF runs the corresponding restart executable when the job restarts. For example, for bsub -k "my_dir method=fluent" LSF runs echkpt.fluent at job checkpoint and erestart.fluent at job restart.• The command-line value for <i>echkpt_application</i> overrides the value specified by LSB_ECHKPNT_METHOD in lsf.conf or as an environment variable. Users can override LSB_ECHKPNT_METHOD and use the default checkpoint and restart executables by defining <i>method=default</i>.

Commands to monitor

Command	Description
bacct -l	<ul style="list-style-type: none">• Displays accounting statistics for finished jobs, including termination reasons. TERM_CHKPNT indicates that a job was checkpointed and killed.• If JOB_CONTROL is defined for a queue, LSF does not display the result of the action.
bhist -l	<ul style="list-style-type: none">• Displays the actions that LSF took on a completed job, including job checkpoint, restart, and migration to another host.
bjobs -l	<ul style="list-style-type: none">• Displays information about pending, running, and suspended jobs, including the checkpoint directory, the checkpoint period, and the checkpoint method (either <i>application</i> or default).

Commands to control

Command	Description
bmod -k " <i>checkpoint_dir</i> [<i>checkpoint_period</i>] [<i>method=echkpt_application</i>]"	<ul style="list-style-type: none">• Resubmits a job and changes the checkpoint directory, checkpoint period, and the checkpoint and restart executables associated with the job.
bmod -kn	<ul style="list-style-type: none">• Dissociates the checkpoint directory from a job, which makes the job no longer checkpointable.
bchkpnt	<ul style="list-style-type: none">• Checkpoints the most recently submitted checkpointable job. Users can specify particular jobs to checkpoint by including various bchkpnt options.

Command	Description
bchkpnt -p <i>checkpoint_period</i> <i>job_ID</i>	<ul style="list-style-type: none"> Checkpoint a job immediately and changes the checkpoint period for the job.
bchkpnt -k <i>job_ID</i>	<ul style="list-style-type: none"> Checkpoint a job immediately and kills the job.
bchkpnt -p 0 <i>job_ID</i>	<ul style="list-style-type: none"> Checkpoint a job immediately and disables periodic checkpointing.
brestart	<ul style="list-style-type: none"> Restarts a checkpointed job on the first available host.
brestart -m	<ul style="list-style-type: none"> Restarts a checkpointed job on the specified host or host group.
bmig	<ul style="list-style-type: none"> Migrates one or more running jobs from one host to another. The jobs must be checkpointable or rerunnable. Checkpoint, kills, and restarts one or more checkpointable jobs.

Commands to display configuration

Command	Description
bqueues -l	<ul style="list-style-type: none"> Displays information about queues configured in <code>lsb.queues</code>, including the values defined for checkpoint directory and checkpoint period. <p>Note: The bqueues command displays the checkpoint period in seconds; the <code>lsb.queues</code> <code>CHKPNT</code> parameter defines the checkpoint period in minutes.</p>
badmin showconf	<ul style="list-style-type: none"> Displays all configured parameters and their values set in <code>lsf.conf</code> or <code>ego.conf</code> that affect mbatchd and sbatchd. <p>Use a text editor to view other parameters in the <code>lsf.conf</code> or <code>ego.conf</code> configuration files.</p> <ul style="list-style-type: none"> In a MultiCluster environment, displays the parameters of daemons on the local cluster.

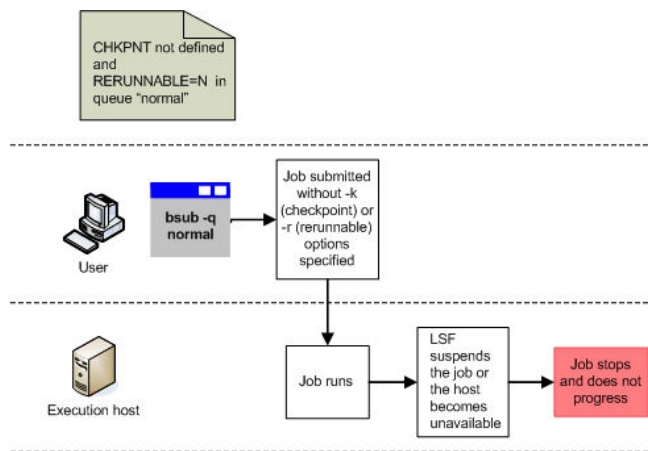
Job migration for checkpoint-able and re-runnable jobs

Use job migration to move checkpoint-able and re-runnable jobs from one host to another. Job migration makes use of job checkpoint and restart so that a migrated checkpoint-able job restarts on the new host from the point at which the job stopped on the original host.

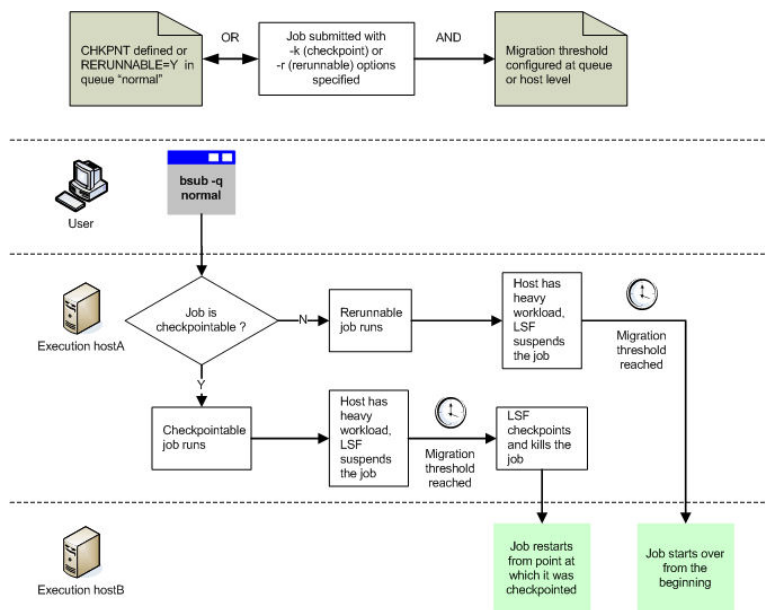
Job migration refers to the process of moving a checkpoint-able or re-runnable job from one host to another. This facilitates load balancing by moving jobs from a heavily-loaded host to a lightly-loaded host.

You can initiate job migration on demand (**bmig**) or automatically. To initiate job migration automatically, you configure a migration threshold at job submission, or at the host, queue, or application level.

Default behavior (job migration not enabled)



With automatic job migration enabled



Scope

Applicability	Details
Operating system	<ul style="list-style-type: none"> • UNIX • Linux • Windows
Job types	<ul style="list-style-type: none"> • Non-interactive batch jobs submitted with bsub or bmod, including chunk jobs

Applicability	Details
Dependencies	<ul style="list-style-type: none"> • UNIX and Windows user accounts must be valid on all hosts in the cluster, or the correct type of account mapping must be enabled: <ul style="list-style-type: none"> ◦ For a mixed UNIX/Windows cluster, UNIX/Windows user account mapping must be enabled ◦ For a cluster with a non-uniform user name space, between-host account mapping must be enabled ◦ For a MultiCluster environment with a non-uniform user name space, cross-cluster user account mapping must be enabled • Both the original and the new hosts must: <ul style="list-style-type: none"> ◦ Be binary compatible ◦ Run the same dot version of the operating system for predictable results ◦ Have network connectivity and read/execute permissions to the checkpoint and restart executables (in LSF_SERVERDIR by default) ◦ Have network connectivity and read/write permissions to the checkpoint directory and the checkpoint file ◦ Have access to all files open during job execution so that LSF can locate them using an absolute path name

- [Job migration behavior](#)
LSF migrates a job by performing the following actions:
- [Configuration to enable job migration](#)
The job migration feature requires that a job be made checkpoint-able or re-runnable at the job, application, or queue level.
- [Configuration to modify job migration](#)
You can configure LSF to requeue a migrating job rather than restart or rerun the job.
- [Job migration commands](#)

Job migration behavior

LSF migrates a job by performing the following actions:

1. Stops the job if it is running
2. Checkpoints the job if the job is checkpoint-able
3. Kills the job on the current host
4. Restarts or reruns the job on the first available host, bypassing all pending jobs

Configuration to enable job migration

The job migration feature requires that a job be made checkpoint-able or re-runnable at the job, application, or queue level.

An LSF user can make a job

- Checkpoint-able, using **bsub -k** and specifying a checkpoint directory and checkpoint period, and an optional initial checkpoint period
- Re-runnable, using **bsub -r**

Configuration file	Parameter and syntax	Behavior
--------------------	----------------------	----------

Configuration file	Parameter and syntax	Behavior
lsb.queues	CHKPNT= <i>chkpnt_dir</i> [<i>chkpnt_period</i>]	<ul style="list-style-type: none"> All jobs submitted to the queue are checkpointable. <ul style="list-style-type: none"> The specified checkpoint directory must already exist. LSF will not create the checkpoint directory. The user account that submits the job must have read and write permissions for the checkpoint directory. For the job to restart on another execution host, both the original and new hosts must have network connectivity to the checkpoint directory. If the queue administrator specifies a checkpoint period, in minutes, LSF creates a checkpoint file every <i>chkpnt_period</i> during job execution. If a user specifies a checkpoint directory and checkpoint period at the job level with bsub -k, the job-level values override the queue-level values.
	RERUNNABLE=Y	<ul style="list-style-type: none"> If the execution host becomes unavailable, LSF reruns the job from the beginning on a different host.
lsb.applications	CHKPNT_DIR= <i>chkpnt_dir</i>	<ul style="list-style-type: none"> Specifies the checkpoint directory for automatic checkpointing for the application. To enable automatic checkpoint for the application profile, administrators must specify a checkpoint directory in the configuration of the application profile. If CHKPNT_PERIOD, CHKPNT_INITPERIOD or CHKPNT_METHOD was set in an application profile but CHKPNT_DIR was not set, a warning message is issued and those settings are ignored. The checkpoint directory is the directory where the checkpoint files are created. Specify an absolute path or a path relative to the current working directory for the job. Do not use environment variables in the directory path. If checkpoint-related configuration is specified in both the queue and an application profile, the application profile setting overrides queue level configuration.
	CHKPNT_INITPERIOD= <i>init_chkpnt_period</i>	
	CHKPNT_PERIOD= <i>chkpnt_period</i>	
	CHKPNT_METHOD= <i>chkpnt_method</i>	

Configuration to enable automatic job migration

Automatic job migration assumes that if a job is system-suspended (SSUSP) for an extended period of time, the execution host is probably heavily loaded. Configuring a queue-level or host-level migration threshold lets the job to resume on another less loaded host, and reduces the load on the original host. You can use **bmig** at any time to override a configured migration threshold.

Configuration file	Parameter and syntax	Behavior
--------------------	----------------------	----------

Configuration file	Parameter and syntax	Behavior
lsb.queues lsb.applications	MIG= <i>minutes</i>	<ul style="list-style-type: none"> LSF automatically migrates jobs that have been in the SSUSP state for more than the specified number of minutes Specify a value of 0 to migrate jobs immediately upon suspension Applies to all jobs submitted to the queue Job-level command-line migration threshold (bsub -mig) overrides threshold configuration in application profile and queue. Application profile configuration overrides queue level configuration.
lsb.hosts	HOST_NAME MIG <i>host_name</i> <i>minutes</i>	<ul style="list-style-type: none"> LSF automatically migrates jobs that have been in the SSUSP state for more than the specified number of minutes Specify a value of 0 to migrate jobs immediately upon suspension Applies to all jobs running on the host

Note: When a host migration threshold is specified, and is lower than the value for the job, the queue, or the application, the host value is used. You cannot auto-migrate a suspended chunk job member.

Configuration to modify job migration

You can configure LSF to requeue a migrating job rather than restart or rerun the job.

Configuration file	Parameter and syntax	Behavior
lsf.conf	LSB_MIG2PEND=1	<ul style="list-style-type: none"> LSF re-queues a migrating job rather than restarting or rerunning the job LSF re-queues the job as pending in order of the original submission time and priority In a multicluster environment, LSF ignores this parameter
	LSB_REQUEUE_TO_BOTTOM=1	<ul style="list-style-type: none"> When LSB_MIG2PEND=1, LSF re-queues a migrating job to the bottom of the queue, regardless of the original submission time and priority If the queue defines APS scheduling, migrated jobs keep their APS information and compete with other pending jobs based on the APS value

Checkpointing re-sizable jobs

After a checkpoint-able re-sizable job restarts (**brestart**), LSF restores the original job allocation request. LSF also restores job-level autoresizable attribute and notification command if they are specified at job submission.

Example

The following example shows a queue configured for periodic checkpointing in lsb.queues:

```

Begin Queue
...
QUEUE_NAME=checkpoint
CHKPNT=mydir 240
DESCRIPTION=Automatically checkpoints jobs every 4 hours to mydir

```

...
End Queue

Note: The **bqueues** command displays the checkpoint period in seconds; the `lsb.queues CHKPNT` parameter defines the checkpoint period in minutes.

If the command **bchkpnt -k 123** is used to checkpoint and kill job 123, you can restart the job using the **brestart** command as shown in the following example:

```
brestart -q priority mydir 123
```

Job <456> is submitted to queue <priority>

LSF assigns a new job ID of 456, submits the job to the queue named "priority," and restarts the job.

Once job 456 is running, you can change the checkpoint period using the **bchkpnt** command:

```
bchkpnt -p 360 456
```

Job <456> is being checkpointed

Job migration commands

Commands for submission

Job migration applies to checkpoint-able or re-runnable jobs submitted with a migration threshold, or that have already started and are either running or suspended.

Command	Description
bsub -mig <i>migration_thres</i> <i>hold</i>	<ul style="list-style-type: none">Submits the job with the specified migration threshold for checkpoint-able or re-runnable jobs. Enables automatic job migration and specifies the migration threshold, in minutes. A value of 0 (zero) specifies that a suspended job should be migrated immediately.Command-level job migration threshold overrides application profile and queue-level settings.Where a host migration threshold is also specified, and is lower than the job value, the host value is used.

Commands to monitor

Command	Description
bhist -l	<ul style="list-style-type: none">Displays the actions that LSF took on a completed job, including migration to another host
bjobs -l	<ul style="list-style-type: none">Displays information about pending, running, and suspended jobs

Commands to control

Command	Description
bmig	<ul style="list-style-type: none">Migrates one or more running jobs from one host to another. The jobs must be checkpoint-able or re-runnableCheckpoints, kills, and restarts one or more checkpoint-able jobs. bmig combines the functionality of the bchkpnt and brestart commands into a single commandMigrates the job on demand even if you have configured queue-level or host-level migration thresholdsWhen absolute job priority scheduling (APS) is configured in the queue, LSF schedules migrated jobs before pending jobs. For migrated jobs, LSF maintains the existing job priority.

Command	Description
bmod -mig <i>migration_thres</i> <i>hold -mign</i>	<ul style="list-style-type: none"> Modifies or cancels the migration threshold specified at job submission for checkpoint-able or re-runnable jobs. Enables or disables automatic job migration and specifies the migration threshold, in minutes. A value of 0 (zero) specifies that a suspended job should be migrated immediately. Command-level job migration threshold overrides application profile and queue-level settings. Where a host migration threshold is also specified, and is lower than the job value, the host value is used.

Commands to display configuration

Command	Description
bhosts -l	<ul style="list-style-type: none"> Displays information about hosts configured in <code>lsb.hosts</code>, including the values defined for migration thresholds in minutes
bqueues -l	<ul style="list-style-type: none"> Displays information about queues configured in <code>lsb.queues</code>, including the values defined for migration thresholds <p>Note: The bqueues command displays the migration threshold in seconds—the <code>lsb.queues MIG</code> parameter defines the migration threshold in minutes.</p>
badmin showconf	<ul style="list-style-type: none"> Displays all configured parameters and their values set in <code>lsf.conf</code> or <code>ego.conf</code> that affect mbatchd and sbatchd. Use a text editor to view other parameters in the <code>lsf.conf</code> or <code>ego.conf</code> configuration files. In a multicluster environment, displays the parameters of daemons on the local cluster.

Re-sizable jobs

Re-sizable jobs can use the number of tasks that are available at any time and can grow or shrink during the job run time by requesting extra tasks if required or release tasks that are no longer needed.

- [Re-sizable job behavior](#)
To optimize resource utilization, LSF allows the job allocation to shrink or grow during the job run time.
- [Configuration to enable re-sizable jobs](#)
- [Re-sizable job commands](#)
- [Re-sizable job management](#)
Submit and manage re-sizable jobs.
- [Specifying a resize notification command manually](#)
- [Script for resizing](#)
- [How re-sizable jobs work with other LSF features](#)
Re-sizable jobs behave differently when used together with other LSF features.

Re-sizable job behavior

To optimize resource utilization, LSF allows the job allocation to shrink or grow during the job run time.

Use re-sizable jobs for long-tailed jobs, which are jobs that use many resources for a period, but use fewer resources toward the end of the job. Conversely, use re-sizable jobs for jobs in which tasks are easily parallelizable, where each step or task can be made to run on a separate processor to achieve a faster result. The more resources the job gets, the faster the job can run. Session Scheduler jobs are good candidates.

Without re-sizable jobs, a job's task allocation is static from the time the job is dispatched until it finishes. For long-tailed jobs, resources are wasted toward the end of the job, even if you use reservation and backfill because estimated run times can be

inaccurate. Parallel run slower than they could run if there were more assigned tasks. With re-sizable jobs, LSF can remove tasks from long-tailed jobs when the tasks are no longer needed, or add extra tasks to parallel jobs when needed during the job's run time.

Automatic or manual resizing

An automatically re-sizable job is a re-sizable job with a minimum and maximum task request, where LSF automatically schedules and allocates more resources to satisfy the job maximum request as the job runs. Specify an automatically re-sizable job at job submission time by using the **bsub -ar** option.

For automatically re-sizable jobs, LSF automatically recalculates the pending allocation requests. LSF is able to allocate more tasks to the running job. For instance, if a job requests a minimum of 4 and a maximum of 32, and LSF initially allocates 20 tasks to the job initially, its active pending allocation request is for another 12 tasks. After LSF assigns another four tasks, the pending allocation request is now eight tasks.

You can also manually shrink or grow a running job by using the **bresize** command. Shrink a job by releasing tasks from the specified hosts with the **bresize release** subcommand. Grow a job by requesting more tasks with the **bresize request** subcommand.

Pending allocation request

A pending allocation request is an extra resource request that is attached to a re-sizable job. Running jobs are the only jobs that can have pending allocation requests. At any time, a job has only one allocation request.

LSF creates a new pending allocation request and schedules it after a job physically starts on the remote host (after LSF receives the `JOB_EXECUTE` event from the **sbatchd** daemon) or resize notification command successfully completes.

Resize notification command

A resize notification command is an executable that is invoked on the first execution host of a job in response to an allocation (grow or shrink) event. It can be used to inform the running application for allocation change. Due to the variety of implementations of applications, each re-sizable application might have its own notification command that is provided by the application developer.

The notification command runs under the same user ID environment, home, and working directory as the actual job. The standard input, output, and error of the program are redirected to the NULL device. If the notification command is not in the user's normal execution path (the `$PATH` variable), the full path name of the command must be specified.

A notification command exits with one of the following values:

LSB_RESIZE_NOTIFY_OK

LSB_RESIZE_NOTIFY_FAIL

LSF sets these environment variables in the notification command environment. The `LSB_RESIZE_NOTIFY_OK` value indicates that the notification succeeds. For allocation grow and shrink events, LSF updates the job allocation to reflect the new allocation.

The `LSB_RESIZE_NOTIFY_FAIL` value indicates notification failure. For allocation "grow" event, LSF reschedules the pending allocation request. For allocation "shrink" event, LSF fails the allocation release request.

For a list of other environment variables that apply to the resize notification command, see the environment variables reference.

Configuration to enable re-sizable jobs

The re-sizable jobs feature is enabled by defining an application profile using the `RESIZABLE_JOBS` parameter in `lsb.applications`.

Configuration file	Parameter and syntax	Behavior
lsb.applications	RESIZABLE_JOBS=Y N auto	<ul style="list-style-type: none"> When RESIZABLE_JOBS=Y jobs that are submitted to the application profile are re-sizable. When RESIZABLE_JOBS=auto jobs that are submitted to the application profile are automatically re-sizable. To enable cluster-wide re-sizable behavior by default, define RESIZABLE_JOBS=Y in the default application profile. The default value is RESIZABLE_JOBS=N. Jobs that are submitted to the application profile are not resizable.
	RESIZE_NOTIFY_CMD=notify_command	RESIZE_NOTIFY_CMD specifies an application-level resize notification command. The resize notification command is invoked on the first execution host of a running resizable job when a resize event occurs.

Re-sizable job commands

Commands for submission

Command	Description
bsub -app <i>application_profile_name</i>	Submits the job to the specified application profile configured for re-sizable jobs
bsub -app <i>application_profile_name</i> -rnc <i>resize_notification_command</i>	Submits the job to the specified application profile configured for re-sizable jobs, with the specified resize notification command. The job-level resize notification command overrides the application-level RESIZE_NOTIFY_CMD setting.
bsub -ar -app <i>application_profile_name</i>	Submits the job to the specified application profile configured for re-sizable jobs, as an automatically re-sizable job. The job-level -ar option overrides the application-level RESIZABLE_JOBS setting. For example, if the application profile is not automatically re-sizable, job level bsub -ar will make the job automatically re-sizable.

Commands to monitor

Command	Description
bacct -l	<ul style="list-style-type: none"> Displays resize notification command. Displays resize allocation changes.
bhist -l	<ul style="list-style-type: none"> Displays resize notification command. Displays resize allocation changes. Displays the job-level automatically re-sizable attribute.
bjobs -l	<ul style="list-style-type: none"> Displays resize notification command. Displays resize allocation changes. Displays the job-level automatically re-sizable attribute. Displays pending resize allocation requests.

Commands to control

Command	Description
---------	-------------

Command	Description
bmod -ar -arn	Add or remove the job-level autoresizable attribute. bmod only updates the autoresizable attribute for pending jobs.
bmod -rnc <i>resize_notification_cmd -rncn</i>	Modify or remove resize notification command for submitted job.
bresize <i>subcommand</i>	<p>Decrease or increase tasks that are allocated to a running resizable job, or cancel pending job resize allocation requests.</p> <p>Use the bresize release command to explicitly release tasks from a running job. When you release tasks from an allocation, a minimum of one task on the first execution host must be retained. Only hosts (and not host groups or compute units) can be released by using the bresize release command. When you release tasks from compound resource requirements, you can release only tasks that are represented by the last term of the compound resource requirement. To release tasks in earlier terms, run bresize release repeatedly to release tasks in subsequent last terms.</p> <p>Use the bresize request command to trigger a manual request for additional allocated tasks. LSF pends the request if the queue cannot meet the minimum tasks request or if the request is over the TASKLIMIT value for the queue or application profile. Changing the TASKLIMIT value does not affect any requests that are already accepted. For compound resource requirements, the request only applies to the last term. For alternative resource requirements, the request only applies to the term that was used for the initial task allocation. For autoresizable jobs, if there is pending demand, you must first cancel the previous pending demand by running the brequest request -c or bresize cancel commands. After triggering this manual request, the job is no longer autoresizable unless you requeue or rerun the job.</p> <p>Use bresize cancel to cancel a pending allocation request for the specified job ID. The active pending allocation request is generated by LSF automatically for autoresizable jobs. If the job does not have an active pending request, the command fails with an error message.</p> <p>By default, only cluster administrators, queue administrators, root, and the job owner are allowed to run bresize to change job allocations.</p> <p>User group administrators are allowed to run bresize to change the allocation of jobs within their user groups.</p>
bresize <i>subcommand -rnc</i> <i>resize_notification_cmd</i>	<p>Specify or remove a resize notification command. The resize notification is invoked on the job first execution node. The resize notification command only applies to this request and overrides the corresponding resize notification parameters defined in either the application profile (RESIZE_NOTIFY_CMD in lsb.applications) and job level (bsub -rnc notify_cmd), only for this resize request.</p> <p>If the resize notification command completes successfully, LSF considers the allocation resize done and updates the job allocation. If the resize notification command fails, LSF does not update the job allocation.</p> <p>The <i>resize_notification_cmd</i> specifies the name of the executable to be invoked on the first execution host when the job's allocation has been modified.</p> <p>The resize notification command runs under the user account that submitted the job.</p> <p>-rncn overrides the resize notification command in both job-level and application-level for this bresize request.</p> <p>The -rnc and -rncn options do not apply to the bresize cancel subcommand.</p>
bresize <i>subcommand -c</i>	<p>By default, if the job has an active pending allocation request, LSF does not allow users to release or increase resources. Use the -c option to cancel the active pending resource request when releasing or increasing tasks from existing allocation. By default, the command only releases or increases tasks.</p> <p>If a job still has an active pending allocation request, but you do not want to allocate more resources to the job, use the bresize cancel command to cancel allocation request.</p> <p>Only the job owner, cluster administrators, queue administrators, user group administrators, and root are allowed to cancel pending resource allocation requests.</p> <p>The -c option does not apply to the bresize cancel subcommand.</p>

Commands to display configuration

Command	Description
bapp	Displays the value of parameters defined in lsb.applications.

Re-sizable job management

Submit and manage re-sizable jobs.

- [Submitting a re-sizable job](#)
- [Checking pending resize requests](#)
- [Canceling an active pending request](#)

Submitting a re-sizable job

Procedure

1. Run `bsub -n 4,10 -ar -app application_profile_name`
LSF dispatches the job to the specified application profile that is configured for re-sizable jobs (as long as the minimum task request is satisfied).

After the job successfully starts, LSF continues to schedule and allocate additional resources to satisfy the maximum task request for the job.

2. (Optional, as required) Release resources that are no longer needed.

```
bresize release released_host_specification job_ID
```

where *released_host_specification* is the specification (list or range of hosts and number of tasks) of resources to be released.

For example,

```
bresize release "1*hostA 2*hostB hostC" 221
```

LSF releases 1 task on hostA, 2 tasks on hostB, and all tasks on hostC for job 221.

Result: The resize notification command runs on the first execution host.

3. (Optional, as required) Request additional tasks to be allocated to jobs that require more resources.

```
bresize request [min_tasks,] tasks job_ID
```

where

- *tasks* is the total number of tasks to allocate to the jobs.
- *min_tasks* is the minimum number of tasks to allocate to the jobs.

If *min_tasks* is specified, LSF pends the request until the minimum number of tasks can be added to the job, otherwise, LSF pends the request until the full total number of tasks is available. Specifying a minimum number of tasks allows the request to grow the task allocation partially, otherwise the request requires the entire maximum number to be allocated at once.

For example,

```
bresize request 10 221
```

This command requests a total of 10 tasks to be added to job 221. If LSF cannot add 10 tasks to the job, the request pends.

```
bresize request 4,10 221
```

This command requests a minimum of 4 and a total of 10 tasks to be added to job 221. The request pends if LSF cannot add at least 4 tasks to the job. The request can add additional tasks up to a total of 10. Therefore, if LSF initially adds 7 tasks to the job, the remaining 3 tasks requested remain pending.

Result: The resize notification command runs on the first execution host.

Checking pending resize requests

About this task

A resize request pends until the job's maximum task request has been allocated or the job finishes (or the resize request is canceled).

Procedure

Run `bjobs -l job_id`.

Canceling an active pending request

Before you begin

Only the job owner, cluster administrators, queue administrators, user group administrators, and root can cancel pending resource allocation requests.

An allocation request must be pending.

About this task

If a job still has an active pending resize request, but you do not want to allocate more resources to the job, you can cancel it.

By default, if the job has an active pending resize request, you cannot release the resources. You must cancel the request first.

Procedure

Run `bresize cancel`.

Specifying a resize notification command manually

About this task

You can specify a resize notification command on job submission, other than one that is set up for the application profile

Procedure

1. On job submission, run `bsub -rnc resize_notification_cmd`.
The job submission command overrides the application profile setting.
2. Ensure the resize notification command checks any environment variables for resizing.

For example, LSB_RESIZE_EVENT indicates why the notification command was called (grow or shrink) and LSB_RESIZE_HOSTS lists tasks and hosts. Use LSB_JOBID to determine which job is affected.

Results

The command that you specified runs on the first execution host of the resized job.

LSF monitors the exit code from the command and takes appropriate action when the command returns an exit code corresponding to resize failure.

Script for resizing

```
#!/bin/sh
# The purpose of this script is to inform
# an application of a resize event.
#
# You can identify the application by:
#
# 1. LSF job ID ($LSB_JOBID), or
# 2. pid ($LS_JOBPID).
#
# handle the 'grow' event
if [ $LSB_RESIZE_EVENT = "grow" ]; then
    # Inform the application that it can use
    # additional tasks as specified in
    # $LSB_RESIZE_HOSTS.
    #
    # Exit with $LSB_RESIZE_NOTIFY_FAIL if
    # the application fails to resize.
    #
    # If the application cannot use any
    # additional resources, you may want
    # to run 'bresize cancel $LSB_JOBID'
    # before exit.

    exit $LSB_RESIZE_NOTIFY_OK
fi

# handle the 'shrink' event
if [ $LSB_RESIZE_EVENT = "shrink" ]; then
    # Instruct the application to release the
    # tasks specified in $LSB_RESIZE_HOSTS.
    #
    # Exit with $LSB_RESIZE_NOTIFY_FAIL if
    # the resources cannot be released.

    exit $LSB_RESIZE_NOTIFY_OK
fi

# unknown event -- should not happen
exit $LSB_RESIZE_NOTIFY_FAIL
```

How re-sizable jobs work with other LSF features

Re-sizable jobs behave differently when used together with other LSF features.

Resource usage

When a job grows or shrinks, its resource reservation (for example memory or shared resources) changes proportionately.

- Job-based resource usage does not change in grow or shrink operations.
- Host-based resource usage changes only when the job gains tasks on a new host or releases all tasks on a host.
- Task-based resource usage changes whenever the job grows or shrinks.

Limits

Tasks are only added to a job's allocation when resize occurs if the job does not violate any resource limits placed on it.

Job scheduling and dispatch

The `JOB_ACCEPT_INTERVAL` parameter in `lsb.params` or `lsb.queues` controls the number of seconds to wait after dispatching a job to a host before dispatching a second job to the same host. The parameter applies to all allocated hosts of a parallel job. For re-sizable job allocation requests, `JOB_ACCEPT_INTERVAL` applies to newly allocated hosts.

Chunk jobs

Because candidate jobs for the chunk job feature are short-running sequential jobs, the re-sizable job feature does not support job chunking:

- Automatically resizable jobs in a chunk queue or application profile cannot be chunked together.
- **bresize** commands to resize job allocations do not apply to running chunk job members.

Energy aware scheduling

In the case that a job is resizable, **bjobs** can only get the energy cost of the latest re-sizable job's executive hosts.

Requeued jobs

Jobs requeued with **brequeue** start from the beginning. After re-queuing, LSF restores the original allocation request for the job.

Launched jobs

Parallel tasks running through **blaunch** can be re-sizable. Automatic job resizing is a signaling mechanism only. It does not expand the extent of the original job launched with **blaunch**. The resize notification script is required along with a signal listening script. The signal listening script runs additional **blaunch** commands on notification to allocate the re-sized resources to make them available to the job tasks. For help creating signal listening and notification scripts, contact IBM Support.

Switched jobs

bswitch can switch re-sizable jobs between queues regardless of job state (including job's resizing state). Once the job is switched, the parameters in new queue apply, including threshold configuration, run limit, CPU limit, queue-level resource requirements, etc.

User group administrators

User group administrators are allowed to issue **bresize** commands to release a part of resources from job allocation (**bresize release**), request additional tasks to allocate to a job (**bresize request**), or cancel active pending resize request (**bresize cancel**).

Re-queue exit values

If job-level, application-level or queue-level `REQUEUE_EXIT_VALUES` are defined, and as long as job exits with a defined exit code, LSF puts the re-queued job back to PEND status. For re-sizable jobs, LSF schedules the job according to the initial allocation request regardless of any job allocation size change.

Automatic job rerun

A re-runnable job is rescheduled after the first running host becomes unreachable. Once job is rerun, LSF schedules re-sizable jobs that are based on their initial allocation request.

Compute units

Automatically re-sizable jobs can have compute unit requirements.

Alternative resource requirements

Re-sizable jobs can have alternative resource requirements. When using **bresize request** to request additional tasks, the task increase is based on the term used for the initial task allocation.

Compound resource requirements

Re-sizable jobs can have compound resource requirements. Only the portion of the job represented by the last term of the compound resource requirement is eligible for automatic resizing. When using **bresize release** to release tasks or **bresize request** to request additional tasks, you can only release tasks represented by the last term of the compound resource requirement. To release or request tasks in earlier terms, run **bresize release** or **bresize request** repeatedly to release or request tasks in subsequent last terms.

GPU resource requirements

If the value of the `LSB_GPU_NEW_SYNTAX` setting in the `lsf.conf` configuration file is configured with a value of `extend`, when jobs with GPU resource requirement grow or shrink tasks, GPU allocations grow or shrink accordingly based on those resource requirements.

Note: When job slots and GPUs shrink, the whole host shrinks. When releasing all hosts, all execution hosts except the first execution host will be released. Releasing partial slots for the execution host is not supported.

This table outlines how GPU resource requirements impact re-sizeable jobs:

GPU resource requirement	Support	Behavior for resize requirement
<code>num=num_gpus[/task host]</code>	Yes	When a job grows or shrinks slots, its GPU usage changes proportionately.
<code>mode=shared exclusive_process</code>	Yes	When a job grows or shrinks slots, its GPU usage changes proportionately.
<code>mps</code>	No	Automatically resizing jobs will have no pending requests, and manually running bresize will be rejected.
<code>j_exclusive=yes no</code>	Yes	When a job grows or shrinks slots, its GPU usage changes proportionately.
<code>aff=yes no</code>	No	Automatically resizing jobs will have no pending requests, and manually running bresize will be rejected.
<code>block=yes no</code>	Yes	Block distribution will be applied to newly allocated slots.
<code>gpack=yes no</code>	Yes	If a new host will be allocated to a re-sizable job, when selecting GPU and hosts, it will consider gpack policy for pack scheduling.
<code>gvendor=amd nvidia</code>	Yes	LSF allocates GPUs with the specified vendor type.
<code>gmodel=model_name[-mem_size]</code>	Yes	LSF allocates the GPUs with the same model, if available.
<code>gmem=mem_value</code>	Yes	LSF allocates GPU memory on each newly allocated GPU required by the new tasks.
<code>gtile=! tile_num</code>	Yes	gtile requirements only apply to newly allocated GPUs.
<code>glink=yes</code>	Yes	Enables job enforcement for special connections among new allocated GPUs for new tasks.
<code>mig</code>	No	Automatically resizing jobs will have no pending requests, and manually running bresize will be rejected. While LSF re-sizes jobs, hosts with GPU devices that enable mig will not be considered.

IBM Spectrum LSF Security

Learn how to optimize the security of your LSF cluster.

- [LSF security considerations](#)
While the default LSF configuration is adequate for most clusters, you should consider the following issues if you want to increase the security of your LSF cluster.
- [Secure your LSF cluster](#)
Perform the following tasks to secure your LSF cluster.

LSF security considerations

While the default LSF configuration is adequate for most clusters, you should consider the following issues if you want to increase the security of your LSF cluster.

- [Communications between daemons and commands](#)
- [Transmission of IBM Spectrum LSF commands for remote execution](#)
- [Access to jobs belonging to other users](#)
- [Accessing remote hosts](#)
- [False requests](#)
- [Authentication](#)

Communications between daemons and commands

Communications between LSF daemons and between LSF commands and daemons are not encrypted. If your LSF clusters are running in an untrusted or unsecured environment, these communications may be susceptible to interception and spoofing attacks. You can enable strict checking of communications to deal with spoofing attacks.

Transmission of IBM® Spectrum LSF commands for remote execution

By default, the following LSF commands make use of remote shell (**rsh**) and remote login (**rlogin**):

- **bctrld start sbd**
- **bpeek**
- **bctrld start lim**
- **bctrld start res**
- **lsfrestart**
- **lsfshutdown**
- **lsfstartup**
- **lsrnp**

rsh and **rlogin** may not be suitable for transmission over an insecure network because it is not encrypted. You can configure these LSF commands to use secure shell (**ssh**), which provides encryption when transmitting commands for remote execution.

Access to jobs belonging to other users

All LSF jobs are run under the user ID of the user who submitted the job (unless you are using account mapping). LSF enforces restrictions on job access based on the user ID of the user who is running a command and the user ID associated with the submitted job.

All LSF users can view basic information on all jobs, including jobs submitted by other users, but can only view detailed information on or modify jobs submitted by their own user IDs. Only administrators can modify jobs submitted by other users.

User commands providing information on all jobs

Any LSF user can run the following commands to view basic information on any jobs running in the cluster, including jobs submitted by other users:

bjobs

displays information about LSF jobs. By default, **bjobs** displays information about your own pending, running, and suspended jobs. You can view information on jobs submitted by other users by using the **-u** option to specify a specific user, user group, or all users (using the **all** keyword).

bhist

displays historical information about LSF jobs. By default, **bhist** displays historical information about your own pending, running, and suspended jobs. You can view historical information on jobs submitted by other users by using the **-u** option to specify a specific user, user group, or all users (using the **all** keyword).

bhosts

displays information on hosts, including job state statistics and job slot limits. By default, you can view the number of jobs running on each host, including jobs submitted by other users; however, you only see the total number of jobs running on the host, not the specific users who submitted the jobs.

bqueues

displays information on queues, including job slot statistics and job state statistics. By default, the user can view the number of jobs running in each queue, including jobs submitted by other users; however, you only see the total number of jobs running in the queue, not the specific users who submitted the jobs.

User commands that restrict information on jobs submitted by other users

Any LSF user can run the following command to provide detailed information on jobs running in the cluster, but not on jobs submitted by other users:

bpeek

displays standard output and standard error output that have been produced by unfinished jobs. This command displays detailed information on the progress of a job, but you can only view jobs that belong to your own user ID.

Queue and administrator commands that modify all jobs

Queue administrators and LSF administrators can run the following commands to modify jobs submitted by any user. LSF users can also run these commands, but only to modify their own jobs with certain restrictions:

bbot

moves a pending job relative to the last job in the queue.

btop

moves a pending job relative to the first job in the queue.

LSF administrator commands that modify all jobs

LSF administrators can run the following commands to modify jobs submitted by any user. LSF users can also run these commands, but only to modify or control their own jobs with certain restrictions:

bchkpnt

Checkpoints one or more checkpointable jobs. LSF administrators can checkpoint jobs submitted by any user.

bkill

Sends a signal to kill unfinished jobs.

bmod

Modifies job submission options of a job.

brestart

Restarts checkpointed jobs.

brresume

Resumes a suspended job.

bstop

Suspends unfinished jobs.

Job data files

Jobs running in the LSF cluster inherit the environment from the user that submitted the job. Work files and output files are created based on the file permissions environment of the user (such as **umask** in POSIX environments). LSF does not provide additional security to these files. Therefore, to increase the security of work and output data, you need update the security of your hosts and file system according to the operating systems on your hosts.

Related concepts

- [Job information access control](#)
- [Setting job information access control](#)

Accessing remote hosts

By default, LSF provides commands for running tasks on remote hosts using LSF daemons (**lim** and **res**) and LSF ports (LSF_LIM_PORT and LSF_RES_PORT) for communication. Therefore, even if your cluster restricts users from directly logging into or running commands on remote hosts (therefore restricting your users to using LSF batch commands to access remote hosts), users can still run the following commands to run tasks on remote systems under certain circumstances.

lsrun

runs an interactive task on a remote host through LSF. You can run a single task on a single remote host.

lsgrun

runs a task on a set of remote hosts through LSF. You can run a single task on multiple remote hosts.

ch

changes the host on which subsequent commands are to be executed. You can change tasks to run on a selected remote host.

False requests

LSF clusters may be vulnerable to large-scale denial of service (DOS) attacks. If one of the LSF daemons becomes overloaded with false requests, it may not be able to respond to valid requests.

By default, LSF refuses to accept client requests from hosts not listed in `lsf.cluster.cluster_name`. If LSF daemons are started on the unlisted host, the daemons will continue to retry the connection. The LSF management host rejects these requests, but if there are many unlisted hosts doing the same thing, it may become overloaded and be unable to respond to valid requests.

Since LSF can handle large clusters (several thousand hosts in a cluster) and is designed to be resistant to this type of attack, a malicious attack needs to simulate a larger scale of false hosts in order to be successful, but LSF still remains potentially vulnerable to a very large-scale attack.

Authentication

In LSF, authentication can come by means of external authentication using the LSF **eauth** executable, or by means of identification daemons (**identd**). External authentication provides the highest level of security and is the default method of authentication in LSF. It is installed in the directory specified by the LSF_SERVERDIR parameter in the `lsf.conf` file.

By default, **eauth** uses an internal key to encrypt authentication data, but you may use a customized external key to improve security. You can also write your own **eauth** executable to meet the security requirements of your cluster, using the default **eauth** as a demonstration of the **eauth** protocol.

Update the eauth executable file

If you are using LSF, Version 10.1 Fix Packs 2 to 9, you can also replace the default **eauth** executable file with the **eauth.cve** executable file, which automatically generates a site-specific internal key by using 128-bit AES encryption. Rename or move the original **eauth** executable file, then rename the **eauth.cve** executable file to **eauth**.

In Windows hosts, replace the **eauth.exe** executable file with **eauth.cve.exe**

The new **eauth** command rejects LSF requests from hosts with a UTC time offset of more than five minutes compared to the LSF server host.

If you are using LSF, Version 10.1 Fix Pack 10 and later, you do not need to replace the **eauth** executable file because it already includes the features in the **eauth.cve** file. In addition, LSF, Version 10.1 Fix Pack 10 and later no longer allows root execution privileges for jobs from local and remote hosts. Any actions that were performed with root privileges must instead

be performed as the LSF administrator. For more details on temporarily enabling root privileges, refer to [Temporarily enable root privileges](#).

If you are using the new **eaauth** command with the LSF multicluster capability in LSF clusters with the following LSF features, you must configure the same LSF_EAUTH_KEY value in the `lsf.sudoers` file on all related clusters:

- Interactive tasks on remote hosts run by using the **lsrun -m** or **lsgrun -m** commands
- LSF data managers

Note: To use the `lsf.sudoers` file, you must enable the `setuid` bit for the LSF administration commands. Run the **hostsetup --setuid** command option on the LSF management and candidate hosts. Since this allows LSF administration commands to run with root privileges, do not enable the `setuid` bit if you do not want these LSF commands to run with root privileges. The **hostsetup --setuid** command enables the `setuid` bit for the following LSF executable files: **badadmin**, **lsadmin**, **egosh**, **utmpreg**, **swtbl_api**, **ntbl_api**, **lstbl_nid**, and **swtbl_poe**.

If you are using IBM® Spectrum LSF RTM, you must also update to the corresponding new **eaauth** executable file for IBM Spectrum LSF RTM.

Note: You must replace the executable file on all LSF hosts in the LSF cluster to work together. Otherwise, LSF commands that run on the hosts without the new **eaauth** executable file will encounter authentication problems.

Secure your LSF cluster

Perform the following tasks to secure your LSF cluster.

Note: If you run LSF in a mixed cluster, you must make sure that parameters set in the `lsf.conf` file on UNIX and Linux match all corresponding parameters in the local `lsf.conf` files on your Windows hosts. When you need to edit the `lsf.conf` file, make sure to specify the same parameters for UNIX, Linux, and Windows hosts.

- [Secure communications between daemons and commands](#)
 - [Encrypt transmission of LSF commands for remote execution and login](#)
 - [Restrict user access to remote hosts](#)
 - [Secure your cluster against false requests](#)
 - [Customize external authentication](#)
 - [Enable external authentication of LSF daemons](#)
 - [Secure the cluster from root access for batch interactive jobs in pseudoterminals](#)
 - [Restrict user access to administration commands and log files](#)
 - [Job information access control](#)
- LSF allows you to set the job information access control level to jobs by users (including user group, queue, and cluster administrators).
- [Secure the `lsf.conf` file and prevent users from changing the job user at job submission time](#)
 - [Temporarily enable root privileges](#)
- Temporarily enable root privileges if you need to perform tasks as root.
- [View the cluster security settings](#)

Use the **badadmin security view** command to check the current configuration of the LSF security mechanism.

Secure communications between daemons and commands

About this task

Note: These steps are now unnecessary because `LSF_STRICT_CHECKING=ENHANCED` is now the default setting. This topic will be removed in a future version of LSF.

To deal with spoofing attacks in your cluster, enable strict checking of communications between LSF daemons and between LSF commands and daemons.

You need to shut down all hosts in the LSF cluster to enable strict checking.

If you are running a IBM® Spectrum LSF multicluster capability environment, you must enable strict checking in all clusters.

Procedure

1. Shut down all hosts in the LSF cluster.
`lsfshutdown`
2. Edit the `lsf.conf` file.
3. Enable strict checking and checksum-secured authorization requests by specifying the `LSF_STRICT_CHECKING` parameter.
Add the following line to `lsf.conf`:

`LSF_STRICT_CHECKING=ENHANCED`
4. Start up all hosts in the LSF cluster.
`lsfstartup`

Results

Your LSF cluster now requires an LSF-generated checksum for all communications. LSF also adds a checksum for each LSF authentication request, which ensures that any data packet that is sent to LSF is identified if it is modified. This prevents users from changing either the user credential or the content of the LSF authentication requests.

Encrypt transmission of LSF commands for remote execution and login

About this task

By default, certain LSF commands use **rsh** for remote execution and **rlogin** for remote login, both of which are not encrypted. To secure these LSF commands, enable the use of **ssh** for remote execution, because **ssh** provides encryption when transmitting LSF commands.

The following LSF commands are covered by this change:

- **bctrld start sbd**
- **bpeek**
- **bctrld start lim**
- **bctrld start res**
- **lsfrestart**
- **lsfshutdown**
- **lsfstartup**
-
- **lsrnp**

Procedure

1. Edit the `lsf.conf` file.
2. Change the remote execution shell from **rsh** to **ssh** by specifying the `LSF_RSH` parameter.
For example,

`LSF_RSH="ssh -o 'PasswordAuthentication no'
-o 'StrictHostKeyChecking no' "`
3. Change the remote login shell by specifying the `LSF_LSLOGIN_SSH` parameter.
`LSF_LSLOGIN_SSH=yes`
4. Reconfigure LIM and restart **mbatchd** on the management host to activate these changes.

```
lsadmin reconfig
badmin mbdrestart
```

Results

The affected LSF commands now use **ssh** for remote execution and remote login.

Restrict user access to remote hosts

About this task

Even if your cluster restricts users from directly accessing remote hosts, they can still use **lsrun**, **lsgrun**, **lsmake**, and **ch** to run tasks on specific remote hosts.

To prevent users from accessing specific remote hosts and let LSF control which remote hosts are being used, restrict access to the **lsrun**, **lsgrun**, **lsmake**, and **ch** commands.

Procedure

1. Edit the `lsf.conf` file.
2. Restrict user access to the **lsrun**, **lsgrun**, and **lsmake** commands by specifying the `LSF_DISABLE_LSRUN` parameter.
LSF_DISABLE_LSRUN=Y

LSF administrators still have access to **lsrun**, **lsgrun**, and **lsmake**.

3. Reconfigure LIM and restart **mbatchd** on the management host to activate these changes.
`lsadmin reconfig`
`badmin mbdrestart`

4. Restrict access to the **ch** commands by restricting the execution permissions of the **ch** binary in the LSF binary directories to the LSF administrators.

Results

Only LSF administrators can run **lsrun**, **lsgrun**, and **lsmake** to launch tasks in remote hosts, and only LSF administrators can run **ch** to change the remote hosts on which a task runs.

Secure your cluster against false requests

About this task

To secure your cluster against false requests sent from unlisted hosts, restrict access to the LSF management host and management candidates.

The parameters you set to restrict access depend on whether your cluster allows dynamic hosts.

Procedure

1. Edit the `lsf.conf` file.
2. Limit the number of management candidates in your cluster that are specified by the `LSF_MASTER_LIST` parameter.
3. If your cluster does not allow dynamic hosts, prevent unlisted hosts from sending requests by specifying the `LSF_REJECT_NONLSFHOST` parameter.

```
LSF_REJECT_NONLSFHOST=yes
```

4. Edit the `lsf.cluster.cluster_name` file.
5. Limit or remove the range of IP addresses that are allowed to be dynamic LSF hosts by editing or deleting the `LSF_HOST_ADDR_RANGE` parameter.
 - If your cluster allows dynamic hosts, limit the range of IP addresses that are specified by the `LSF_HOST_ADDR_RANGE` parameter.
 - If your cluster does not allow dynamic hosts, ensure that the `LSF_HOST_ADDR_RANGE` parameter is not specified.
6. Reconfigure LIM and restart **mbatchd** on the management host to activate these changes.

```
lsadmin reconfig
```

```
badmin mbdrestart
```

Customize external authentication

Before you begin

To use the `lsf.sudoers` file, you must enable the `setuid` bit for the LSF administration commands. Run the **hostsetup --setuid** command option on the LSF management and candidate hosts. Since this allows LSF administration commands to run with root privileges, do not enable the `setuid` bit if you do not want these LSF commands to run with root privileges. The **hostsetup --setuid** command enables the `setuid` bit for the following LSF executable files: **badmin**, **lsadmin**, **egosh**, **utmpreg**, **swtbl_api**, **ntbl_api**, **lstbl_nid**, and **swtbl_poe**.

About this task

By default, **eauth** uses an internal key to encrypt authentication data, but you may wish to use your own external key to further improve security.

If you are using LSF, Version 10.1 Fix Pack 2, or later, you can also replace the default **eauth** executable file with the **eauth.cve** executable file, which automatically generates a site-specific internal key by using 128-bit AES encryption. For more detailed information on how to replace the **eauth** executable file, see [lsf_security_authentication.html#lsf_security_authentication_v1726199](https://www.ibm.com/support/ctodoc/p1726199).

You can also write your own external authentication application to meet the security requirements of your cluster.

Procedure

1. Edit the `lsf.sudoers` file.
2. Use a custom external key by specifying the `LSF_EAUTH_KEY` parameter.

```
LSF_EAUTH_KEY=key
```

You can generate a key by using public key generator tools such as GnuPG or **ssh-keygen**. The following is an example of generating authorization keys with GnuPG:

- a. Download the GnuPG package from [the GnuPG website](https://www.gnupg.org/) and install the package on the LSF management host.
- b. Create GPG keys by running the **gpg --gen-key** command.
- c. Edit the `lsf.sudoers` file and comment out the original `LSF_EAUTH_KEY` parameter definition.
- d. Export the ASCII-armored format output of the GPG key and append this format output to the `lsf.sudoers` file.

```
LSF_EAUTH_KEY=<ASCII-armored format of the GPG key>
```

Note: In Windows hosts, you must edit the shared `lsf.sudoers` file.

- e. In UNIX hosts, copy the `lsf.sudoers` file to the `/etc` directory on each LSF host.
- f. Reconfigure the cluster to apply this change.

```
badmin reconfig
```

3. Restart the cluster to activate this change.

Enable external authentication of LSF daemons

Before you begin

To use the `lsf.sudoers` file, you must enable the `setuid` bit for the LSF administration commands. Run the **hostsetup --setuid** command option on the LSF management and candidate hosts. Since this allows LSF administration commands to run with root privileges, do not enable the `setuid` bit if you do not want these LSF commands to run with root privileges.

The **hostsetup --setuid** command enables the `setuid` bit for the following LSF executable files: **badmin**, **lsadmin**, **egosh**, **utmpreg**, **swtbl_api**, **ntbl_api**, **lstbl_nid**, and **swtbl_poe**.

About this task

You can increase LSF daemon security in your cluster by enabling LSF daemon authentication.

Procedure

1. Edit the `lsf.sudoers` file.
2. Enable LSF daemon authentication by specifying the `LSF_AUTH_DAEMONS` parameter.
LSF_AUTH_DAEMONS=Y
3. Reconfigure the management host to activate this change.
`badmin reconfig`

Secure the cluster from root access for batch interactive jobs in pseudoterminals

About this task

Batch interactive jobs in pseudoterminals (that is, jobs submitted using **bsub -Is** and **bsub -Ip** commands) could obtain root privileges to your cluster due to environment variables (`LD_PRELOAD` and `LD_LIBRARY_PATH`) contained in the jobs.

To enhance security against users obtaining root privileges using batch interactive jobs in pseudoterminals, enable the cluster remove these environment variables from batch interactive jobs during job initialization. These environment variables are put back before the job runs.

Procedure

1. Edit the `lsf.conf` file.
2. Enable the cluster to remove the `LD_PRELOAD` and `LD_LIBRARY_PATH` environment variables from jobs submitted using **bsub -Is** and **bsub -Ip** commands during job initialization by specifying the `LSF_LD_SECURITY` parameter.
LSF_LD_SECURITY=y
3. Reconfigure LIM and restart **mbatchd** on the management host to activate these changes.
`lsadmin reconfig`

`badmin mbdrestart`

Results

In jobs submitted using **bsub -Is** and **bsub -Ip** commands, the LD_PRELOAD and LD_LIBRARY_PATH environment variables are moved to the LSF_LD_PRELOAD and LSF_LD_LIBRARY_PATH environment variables and are moved back before the job runs.

Restrict user access to administration commands and log files

About this task

Log files may contain sensitive cluster information that need to be restricted to LSF administrators only. To restrict access to the LSF cluster log files, restrict the read/write permissions to all files in the log directory.

Cluster administrative tools (**badmin** and **lsadmin**) can only be used by LSF administrators. To provide an additional layer of security to prevent unauthorized administrator access to your LSF cluster, restrict the execution permissions for these commands.

Procedure

1. Restrict access to the LSF cluster log files by restricting the read/write permissions of the **log** directory to the LSF administrators.
2. Restrict access to the administrative tools by restricting the execution permissions of the **badmin** and **lsadmin** binaries in the LSF binary directories to the LSF administrators.

Tip:

You can also restrict access to other LSF commands by restricting the execution permissions of their respective binary files.

Results

Only LSF administrators can read the contents of the log directory or run cluster administration commands (**badmin** and **lsadmin**).

Job information access control

LSF allows you to set the job information access control level to jobs by users (including user group, queue, and cluster administrators).

This control is useful for large environments where many groups may share the same cluster and it may be a security threat to allow some users to view job details and summary information. With job information access control levels configured, you may prevent users (including administrator users) from viewing other user's job information through LSF commands including **bjobs**, **bjdepinfo**, **bread**, **bstatus**, **bhist**, and **bacct**.

Note:

- There are no rights restrictions for the primary administrator. They may always see all job detail information.
- On UNIX platforms, there is no rights restriction for root. On Windows platforms, the Windows administrator is treated as a regular user.
- Job information access control is not supported on LSF Express Edition.
- Some batch commands that use the job query API (that is, **bkill**, **bstop**, **bresume**, **bchkpnt**, **bmig**, **bqueue**, and **bswitch**) are affected by enabling job information access control. If these commands are issued without specifying the jobId, the behavior will follow the job information access control settings, when enabled. If these commands are issued with the jobId specified, the behavior will not follow the job information access control settings.

Job information types

There are two kinds of job information which will be viewed by users:

- **Summary Information:**
Obtained from **bjobs** with options other than -l, such as -aps, -fwd, -p, -ss, -sum, -W, -WF, -WP, -WL, etc.

If SECURE_INFODIR_USER_ACCESS is set to G in the lsb.params file, this is also obtained from **bacct** with options that are not -l and -UF and from **bhist** with that are not -b, -l, and -UF. This includes the **bacct -d, -e, -q**, and **-w** options; and the **bhist -a, -b**, and **-d** options.

- **Detail Information:**
Obtained from **bjobs -l, bjobs -UF, bjobs -N, bjobinfo, bread**, and **bstatus**.

If SECURE_INFODIR_USER_ACCESS is set to G in the lsb.params file, this is also obtained from **bacct -l, bacct -UF, bhist -b, bhist -l**, and **bhist -UF**.

There are two kinds of user rights which will determine what kind of information a user can view for a job:

- **Basic rights:** User can see all summary information.
- **Detail rights:** User can see all detail information.
- **[Setting job information access control](#)**
There are three parameters available in lsb.params that allow you to control access to job information: SECURE_JOB_INFO_LEVEL, ENABLE_JOB_INFO_BY_ADMIN_ROLE, and SECURE_INFODIR_USER_ACCESS.

Setting job information access control

There are three parameters available in lsb.params that allow you to control access to job information: SECURE_JOB_INFO_LEVEL, ENABLE_JOB_INFO_BY_ADMIN_ROLE, and SECURE_INFODIR_USER_ACCESS.

Controlling jobs a user can see

The parameter SECURE_JOB_INFO_LEVEL in lsb.params allows you to control which jobs any user (including administrators other than the primary administrator) can see information for. A value between 0 and 4 is defined, with 0 being no security and 4 being the highest security.

When a user or administrator enters one of the commands to see job information (**bjobs, bjobinfo, bread**, or **bstatus**; also **bacct** and **bhist** if SECURE_INFODIR_USER_ACCESS=G), the SECURE_JOB_INFO_LEVEL parameter controls what they see. The following table describes the type of job information that can be viewed by a user with each security level.

Security Level	User's Own Job	Same User Group Job Summary Info	Same User Group Job Detail Info	All Other Jobs' Summary Info	All Other Jobs' Detail Info
0	Y	Y	Y	Y	Y
1	Y	Y	Y	Y	
2	Y	Y	Y		
3	Y	Y			
4	Y				
5	Y	Y		Y	

Note:

- If SECURE_JOB_INFO_LEVEL is set to a level greater than 0, LSF checks if SECURE_INFODIR_USER_ACCESS is enabled (set to Y or G). If it is not enabled, access to **bjobs** functions will be restricted, but access to **bhist** / **bacct** will be available.
- When using the LSF multicluster capability, the SECURE_JOB_INFO_LEVEL definition still applies when a user attempts to view job information from a remote cluster through the **bjobs -m remotecoluster** command. The security level configuration of a specified cluster will take effect.

Enabling administrator rights to job information

By default, an administrator's access to job details is determined by the setting of `SECURE_JOB_INFO_LEVEL`, the same as a regular user. The parameter `ENABLE_JOB_INFO_BY_ADMIN_ROLE` in `lsb.params` allows you to enable user group, queue, and cluster administrators the right to access job detail information for jobs in the user group, queue, and clusters they manage, even when the administrator has no right based on the configuration of `SECURE_JOB_INFO_LEVEL`.

When an administrator enters one of the commands to see job information (**bjobs**, **bjdepinfo**, **bread**, or **bstatus**; also **bacct** and **bhist** if `SECURE_INFODIR_USER_ACCESS=G`), the `ENABLE_JOB_INFO_BY_ADMIN_ROLE` definition controls whether they see job detail information about jobs in their user group, queue or cluster that they manage.

The parameter may be set with any combination of the values `usergroup`, `queue`, or `cluster`.

Note: This does not apply to the primary administrator who will always see job information.

Preventing users from viewing jobs that belong to other users

The parameter `SECURE_INFODIR_USER_ACCESS` in `lsb.params` allows you to control whether regular and administrator users (except the primary admin) can see other user's jobs when using the **bhist** or **bacct** command.

If enabled (defined as `Y`), regular users and administrators can view only their own job information when using the **bhist** or **bacct** command, but you can control the granularity of the **bjobs** command to specify the information that other users can see by specifying a value for the `SECURE_JOB_INFO_LEVEL` parameter in the `lsb.params` file. `LSB_SHAREDIR/cluster/logdir` will be readable only by the primary administrator.

If enabled with increased granularity (defined as `G`), regular users and administrators can normally view only their own job information when using the **bhist** or **bacct** commands, but you can control the granularity of these commands to specify the information that other users can see by specifying a value for the `SECURE_JOB_INFO_LEVEL` parameter in the `lsb.params` file. `LSB_SHAREDIR/cluster/logdir` will be readable only by the primary administrator.

When disabled (defined as `N`), access to read `LSB_SHAREDIR/cluster/logdir` returns to default after an **mbatchd restart** or **reconfig**.

Note: An LSF cluster should have only one primary administrator. For example, server and management hosts should have the same primary administrator to ensure **bhist** and **bacct** commands have rights to access the events file.

Note: This feature is only supported when LSF is installed on a file system that supports `setuid` bit for file. Therefore, this feature does not work on Windows platforms.

Secure the `lsf.conf` file and prevent users from changing the job user at job submission time

About this task

LSF uses an external authentication framework to secure user credentials for the data stream between LSF clients and servers. However, there is a potential security issue where, when submitting a job, users can preload the **getuid** and change the job user.

To prevent the preloading of **getuid** to prevent users from changing the job user at job submission time, run the **hostsetup --ext-serverdir** and **--eauth** command options with root privileges.

Procedure

On each host, run the **hostsetup --ext-serverdir** and **--eauth** command options with root privileges.

```
hostsetup --ext-serverdir="file_path" --eauth-key="my_eauth_key"
```

```
--ext-serverdir="file_path"
```

Specify the location of the **eauth** executable file. This file path must be accessible to the local host from which you are running the **hostsetup** command.

```
--eauthkey="my_eauth_key"
```

Specify the key string. Running this command option writes the following line to the `/etc/lsf.sudoers` file:

```
LSF_EAUTH_KEY="my_eauth_key"
```

If you are using special characters, you must use an escape character before the special character (for example, `--eauth-key="\&abdef"`), which is the same as other shell terminal input.

Note: If EGO is enabled, define the `EGO_SERVERDIR` parameter in the `$EGO_CONFDIR/ego.conf` file.

The **hostsetup --ext-serverdir** command option performs the following actions:

- Creates a soft link from the cluster's `lsf.conf` file to `/etc/lsf.conf`.
- Writes values for the `LSF_EXT_SERVERDIR`, `LSF_SERVERDIR`, and `LSF_ENV_OVERRIDE=N` parameters to the `/etc/lsf.conf` file.
- Copies `eauth` and `esub*` to the `LSF_EXT_SERVERDIR` directory, gives it root privileges, and sets the setuid bit to **eauth**.

Setting `LSF_ENV_OVERRIDE=N` means that LSF only uses parameters in the `/etc/lsf.conf` file. The `LSF_SERVERDIR` and `LSF_BINDIR` parameters must be defined in this file. If you defined the `LSF_EXT_SERVERDIR` parameter, LSF uses the **eauth** executable file in this directory. Do not remove the **eauth** file from the `LSF_SERVERDIR` directory for compatibility reasons.

Temporarily enable root privileges

Temporarily enable root privileges if you need to perform tasks as root.

LSF does not allow root execution privileges for jobs from local and remote hosts. Any actions that you would perform in LSF with root privileges must instead be performed as the LSF administrator.

If you need to temporarily run LSF commands with root privileges, specify `LSF_ROOT_USER=Y` in the `lsf.conf` file. When you are done, you must disable this parameter to ensure that your cluster remains secure.

If you need to temporarily run LSF License Scheduler commands with root privileges, specify `LS_ROOT_USER=Y` in the `lsf.licensescheduler` file. This parameter allows the root user to run the **bladmin**, **blkill**, **globauth**, and **taskman** commands. When you are done, you must disable this parameter to ensure that your cluster remains secure.

If you need to enable root privileges on hosts for LSF Application Center, LSF RTM, or LSF Explorer, specify a space-separated lists of hosts in the `LSF_ADDON_HOSTS` in the `lsf.conf` file. The root users on these specified hosts can remotely execute commands. You must also set `LSF_DISABLE_LSRUN=N` in the `lsf.conf` file to enable hosts that are running LSF Application Center to use the **lsrun** and **lsgrun** commands.

View the cluster security settings

Use the **badmin security view** command to check the current configuration of the LSF security mechanism.

- The **badmin security view** command displays a summary of the current configuration.
- The **badmin security view -v** command option provides a detailed description of the current configuration and displays any changes that you need to make to the configuration to secure your cluster.

IBM Spectrum LSF advanced configuration

Learn about LSF error and event logging and how LSF handles job exceptions. Configure advanced LSF features.

- Error and event logging**
Learn how LSF uses system directories, log files, temporary work files, log files, and transaction files and job spooling files. Manage LSF error logs, system event logs. Configure duplicate logging of event logs and set daemon message log levels. Set daemon timing levels and configure LSF job termination reason logging. Learn about LSF job exit codes.

- [Event generation](#)
Learn how LSF detects events occurring during daemon operations. LSF provides a program which translates LSF events into SNMP traps. Certain daemon operations cause **mbatchd** or the parent LIM to call the event program to generate an event. Each LSF event is identified by a predefined number, which is passed as an argument to the event program.
- [Customize batch command messages](#)
- [How LIM determines host models and types](#)
- [Shared file access](#)
- [Shared configuration file content](#)
Use the `#INCLUDE` directive to insert the contents of the specified file into a configuration file to share common configurations between clusters or hosts.
- [Authentication and authorization](#)
- [Handle job exceptions](#)
- [Tune CPU factors](#)
- [Set clean period for DONE jobs](#)
- [Enable host-based resources](#)
Learn how Portable Hardware Locality (hwloc) is integrated into LSF to detect hardware information. Enable LSF so applications can use NVIDIA Graphic Processing Units (GPUs) and Intel Xeon Phi (MIC) co-processors in a Linux environment.
- [Global fair share scheduling](#)
The global fair share scheduling policy divides the processing power of the IBM Spectrum LSF multicluster capability and the LSF/XL feature of IBM Spectrum LSF Advanced Edition among users to provide fair access to all resources, so that every user can use the resources of multiple clusters according to their configured shares.
- [Manage LSF on EGO](#)
The enterprise grid orchestrator capability (EGO) enables enterprise applications to benefit from sharing of resources across the enterprise grid. When LSF on EGO is configured, EGO serves as the central resource broker for LSF.
- [Load sharing X applications](#)
- [Using LSF with the Etnus TotalView Debugger](#)
- [Register LSF host names and IP addresses to LSF servers](#)
Register the IP address and host name of your local LSF host with LSF servers so that LSF servers do not need to use the DNS server to resolve your local host.

Error and event logging

Learn how LSF uses system directories, log files, temporary work files, log files, and transaction files and job spooling files. Manage LSF error logs, system event logs. Configure duplicate logging of event logs and set daemon message log levels. Set daemon timing levels and configure LSF job termination reason logging. Learn about LSF job exit codes.

Tip: The LSF log and event files (including those for License Scheduler and Data Manager) can contain personal information (for example, UNIX usernames and email addresses). OS-level access control is one level of protecting this persona data; for further protection, you can use encryption software tools to encrypt the disk partition or volume used to store the log and events files. LSF log and event files are located at `LSF_TOP/log` and `LSF_TOP/work/cluster_name/logdir`.

- [System directories and log files](#)
LSF uses directories for temporary work files, log files, and transaction files and spooling.
- [Manage error logs](#)
- [System event log](#)
- [Duplicate logging of event logs](#)
- [Set daemon message log to debug level](#)
- [Set daemon timing levels](#)
- [LSF job termination reason logging](#)
- [LSF job exit codes](#)

System directories and log files

LSF uses directories for temporary work files, log files, and transaction files and spooling.

- [About LSF log files and directories](#)

Learn how LSF logs daemon errors, and keeps track of job events and accounting information in the system. The LSF log files are found in the director `LSB_SHAREDIR/cluster_name/logdir`.

- [Log levels and descriptions](#)

About LSF log files and directories

Learn how LSF logs daemon errors, and keeps track of job events and accounting information in the system. The LSF log files are found in the director `LSB_SHAREDIR/cluster_name/logdir`.

LSF daemon error logs

LSF log files are reopened each time a message is logged, so if you rename or remove a daemon log file, the daemons will automatically create a new log file.

The LSF daemons log messages when they detect problems or unusual situations.

The daemons can be configured to put these messages into files.

The following error log files are maintained for the LSF system daemons:

- `res.log.host_name`
- `sbatchd.log.host_name`
- `mbatchd.log.host_name`
- `mbschd.log.host_name`

LSF daemons log error messages in different levels so that you can choose to log all messages, or only log messages that are deemed critical. Message logging for LSF daemons is controlled by the parameter `LSF_LOG_MASK` in the `lsf.conf` file. Possible values for this parameter can be any log priority symbol that is defined in `/usr/include/sys/syslog.h`. The default value for `LSF_LOG_MASK` is `LOG_WARNING`.

LSF error log location

If the optional `LSF_LOGDIR` parameter is defined in `lsf.conf`, error messages from LSF servers are logged to files in this directory.

If the `LSF_LOGDIR` parameter is defined, but the daemons cannot write to files there, the error log files are created in `/tmp`.

If the `LSF_LOGDIR` parameter is not defined, errors are logged to the system error logs (**syslog**) using the `LOG_DAEMON` facility. **syslog** messages are highly configurable, and the default configuration varies widely from system to system. Start by looking for the file `/etc/syslog.conf`, and read the man pages for **syslog(3)** and **syslogd(1)**.

If the error log is managed by **syslog**, it is probably already being automatically cleared.

If LSF daemons cannot find `lsf.conf` when they start, they will not find the definition of the `LSF_LOGDIR` parameter. In this case, error messages go to **syslog**. If you cannot find any error messages in the log files, they are likely in the **syslog**.

Log directory permissions and ownership

Make sure that the permissions on the `LSF_LOGDIR` directory to be writable by `root`. The LSF administrator must own `LSF_LOGDIR`.

LSF event and accounting logs

The following files keep track of the state of the LSF system:

`lsb.events`

LSF uses the `lsb.events` file to keep track of the state of all jobs. Each job is a transaction from job submission to job completion. The system keeps track of everything associated with the job in the `lsb.events` file.

`lsb.events.n`

The events file is automatically trimmed and old job events are stored in files that are named `lsb.event.n`. When the **mbatchd** daemon starts, it refers only to the `lsb.events` file, not the `lsb.events.n` files. The **bhist** command does refer to the `lsb.events.n` files.

Job script files in the info directory

When you use the **bsub** command from a shell prompt, LSF collects all of the commands on the **bsub** line and spools the data to the **mbatchd** daemon, which saves the **bsub** command script in the info directory (or in one of its subdirectories if the `MAX_INFO_DIRS` parameter is defined in the `lsb.params` file) for use at dispatch time or if the job is rerun. The info directory is managed by LSF. No one should modify it.

Note: Job script files for jobs that are stored in the jobinfo cache are not stored in the info directory, but they are stored in the `lsb.jobinfo.events` file.

Log levels and descriptions

Number	Level	Description
0	LOG_EMERG	Log only those messages in which the system is unusable.
1	LOG_ALERT	Log only those messages for which action must be taken immediately.
2	LOG_CRIT	Log only those messages that are critical.
3	LOG_ERR	Log only those messages that indicate error conditions.
4	LOG_WARNING	Log only those messages that are warnings or more serious messages. This is the default level of debug information.
5	LOG_NOTICE	Log those messages that indicate normal but significant conditions or warnings and more serious messages.
6	LOG_INFO	Log all informational messages and more serious messages.
7	LOG_DEBUG	Log all debug-level messages.
8	LOG_TRACE	Log all available messages.

Manage error logs

Error logs maintain important information about LSF operations. When you see any abnormal behavior in LSF, you should first check the appropriate error logs to find out the cause of the problem.

LSF log files grow over time. These files should occasionally be cleared, either by hand or using automatic scripts.

Daemon error logs

LSF log files are reopened each time a message is logged, so if you rename or remove a daemon log file, the daemons will automatically create a new log file.

The LSF daemons log messages when they detect problems or unusual situations.

The daemons can be configured to put these messages into files.

The error log file names for the LSF system daemons are:

- `res.log.host_name`
- `sbatchd.log.host_name`

- `mbatchd.log.host_name`
- `mbschd.log.host_name`

LSF daemons log error messages in different levels so that you can choose to log all messages, or only log messages that are deemed critical. Message logging for LSF daemons (except LIM) is controlled by the parameter `LSF_LOG_MASK` in `lsf.conf`. Possible values for this parameter can be any log priority symbol that is defined in `/usr/include/sys/syslog.h`. The default value for `LSF_LOG_MASK` is `LOG_WARNING`.

Important:

`LSF_LOG_MASK` in `lsf.conf` no longer specifies LIM logging level in LSF 10. For LIM, you must use `EGO_LOG_MASK` in `ego.conf` to control message logging for LIM. The default value for `EGO_LOG_MASK` is `LOG_WARNING`.

- [Set the log files owner](#)
- [View the number of file descriptors remaining](#)
- [Locate error logs](#)

Set the log files owner

Before you begin

You must be the cluster administrator. The performance monitoring (**perfmon**) metrics must be enabled or you must set `LC_PERFM` to debug.

About this task

You can set the log files owner for the LSF daemons (not including **mbschd**). The default owner log files owner is the owner of the `lsf.conf` file, who is normally the LSF administrator with root privileges.

Restriction:

1. Applies to UNIX hosts only.
2. This change only takes effect for daemons that are running as `root`.

Procedure

1. Edit `lsf.conf` and add the parameter `LSF_LOGFILE_OWNER`.
2. Specify a user account name to set the owner of the log files.
3. Shut down the LSF daemon or daemons you want to set the log file owner for.
Run **lsfshutdown** on the host.
4. Delete or move any existing log files.
Important: If you do not clear out the existing log files, the file ownership does not change.
5. Restart the LSF daemons that you shut down.
Run **lsfstartup** on the host.

View the number of file descriptors remaining

Before you begin

The performance monitoring (perfmon) metrics must be enabled or you must set `LC_PERFM` to debug.

About this task

The `mbatchd` daemon can log a large number of files in a short period when you submit a large number of jobs to LSF. You can view the remaining file descriptors at any time.

Restriction:

Applies to UNIX hosts only.

Procedure

Run **`badadmin perfmon view`**.

The free, used, and total amount of file descriptors display.

On AIX5, 64-bit hosts, if the file descriptor limit has never been changed, the maximum value displays:
9223372036854775797.

Locate error logs

Procedure

- Optionally, set the `LSF_LOGDIR` parameter in `lsf.conf`.
Error messages from LSF servers are logged to files in this directory.
- If `LSF_LOGDIR` is defined, but the daemons cannot write to files there, the error log files are created in `/tmp`.
- If `LSF_LOGDIR` is not defined, errors are logged to the system error logs (**`syslog`**) using the `LOG_DAEMON` facility.
`syslog` messages are highly configurable, and the default configuration varies from system to system. Start by looking for the file `/etc/syslog.conf`, and read the man pages for `syslog(3)` and `syslogd(1)`. If the error log is managed by **`syslog`**, it is probably being automatically cleared.
- If LSF daemons cannot find `lsf.conf` when they start, they will not find the definition of `LSF_LOGDIR`. In this case, error messages go to **`syslog`**. If you cannot find any error messages in the log files, they are likely in the **`syslog`**.

System event log

The LSF daemons keep an event log in the `lsb.events` file. The `mbatchd` daemon uses this information to recover from server failures, host reboots, and `mbatchd` restarts. The `lsb.events` file is also used by the **`bhist`** command to display detailed information about the execution history of batch jobs, and by the **`badadmin`** command to display the operational history of hosts, queues, and daemons.

By default, `mbatchd` automatically backs up and rewrites the `lsb.events` file after every 1000 batch job completions. This value is controlled by the `MAX_JOB_NUM` parameter in the `lsb.params` file. The old `lsb.events` file is moved to `lsb.events.1`, and each old `lsb.events.n` file is moved to `lsb.events.n+1`. LSF never deletes these files. If disk storage is a concern, the LSF administrator should arrange to archive or remove old `lsb.events.n` files periodically.

CAUTION:

Do not remove or modify the current `lsb.events` file. Removing or modifying the `lsb.events` file could cause batch jobs to be lost.

Duplicate logging of event logs

To recover from server failures, host reboots, or `mbatchd` restarts, LSF uses information that is stored in `lsb.events`. To improve the reliability of LSF, you can configure LSF to maintain a copy of `lsb.events` to use as a backup.

If the host that contains the primary copy of the logs fails, LSF will continue to operate using the duplicate logs. When the host recovers, LSF uses the duplicate logs to update the primary copies.

- [Configure duplicate logging](#)

How duplicate logging works

By default, the event log is located in LSB_SHAREDIR. Typically, LSB_SHAREDIR resides on a reliable file server that also contains other critical applications necessary for running jobs, so if that host becomes unavailable, the subsequent failure of LSF is a secondary issue. LSB_SHAREDIR must be accessible from all potential LSF management hosts.

When you configure duplicate logging, the duplicates are kept on the file server, and the primary event logs are stored on the first management host. In other words, LSB_LOCALDIR is used to store the primary copy of the batch state information, and the contents of LSB_LOCALDIR are copied to a replica in LSB_SHAREDIR, which resides on a central file server. This has the following effects:

- Creates backup copies of lsb.events
- Reduces the load on the central file server
- Increases the load on the LSF management host

Failure of file server

If the file server containing LSB_SHAREDIR goes down, LSF continues to process jobs. Client commands such as **bhist**, which directly read LSB_SHAREDIR will not work.

When the file server recovers, the current log files are replicated to LSB_SHAREDIR.

Failure of first management host

If the first management host fails, the primary copies of the files (in LSB_LOCALDIR) become unavailable. Then, a new management host is selected. The new management host uses the duplicate files (in LSB_SHAREDIR) to restore its state and to log future events. There is no duplication by the second or any subsequent LSF management hosts.

When the first management host becomes available after a failure, it will update the primary copies of the files (in LSB_LOCALDIR) from the duplicates (in LSB_SHAREDIR) and continue operations as before.

If the first management host does not recover, LSF will continue to use the files in LSB_SHAREDIR, but there is no more duplication of the log files.

Simultaneous failure of both hosts

If the management host containing LSB_LOCALDIR and the file server containing LSB_SHAREDIR both fail simultaneously, LSF will be unavailable.

Network partitioning

We assume that Network partitioning does not cause a cluster to split into two independent clusters, each simultaneously running mbatchd.

This may happen given certain network topologies and failure modes. For example, connectivity is lost between the first management host, M1, and both the file server and the secondary management host, M2. Both M1 and M2 will run mbatchd service with M1 logging events to LSB_LOCALDIR and M2 logging to LSB_SHAREDIR. When connectivity is restored, the changes made by M2 to LSB_SHAREDIR will be lost when M1 updates LSB_SHAREDIR from its copy in LSB_LOCALDIR.

The archived event files are only available on LSB_LOCALDIR, so in the case of network partitioning, commands such as **bhist** cannot access these files. As a precaution, you should periodically copy the archived files from LSB_LOCALDIR to LSB_SHAREDIR.

Automatic archives

Archived event logs, lsb.events.n, are not replicated to LSB_SHAREDIR. If LSF starts a new event log while the file server containing LSB_SHAREDIR is down, you might notice a gap in the historical data in LSB_SHAREDIR.

Configure duplicate logging

Procedure

1. Edit `lsf.conf` and set `LSB_LOCALDIR` to a local directory that exists only on the first management host. This directory is used to store the primary copies of `lsb.events`.
2. Use the commands **lsadmin reconfig** and **badmin mbdrestart** to make the changes take effect.

Set an event update interval

About this task

If NFS traffic is high you can reduce network traffic by changing the update interval.

Procedure

Use `EVENT_UPDATE_INTERVAL` in `lsb.params` to specify how often to back up the data and synchronize the `LSB_SHAREDIR` and `LSB_LOCALDIR` directories.

The directories are always synchronized when data is logged to the files, or when `mbatchd` is started on the first LSF management host.

Set daemon message log to debug level

The message log level for LSF daemons is set in `lsf.conf` with the parameter `LSF_LOG_MASK`. To include debugging messages, set `LSF_LOG_MASK` to one of:

- `LOG_DEBUG`
- `LOG_DEBUG1`
- `LOG_DEBUG2`
- `LOG_DEBUG3`

By default, `LSF_LOG_MASK=LOG_WARNING` and these debugging messages are not displayed.

The debugging log classes for LSF daemons are set in `lsf.conf` with the parameters `LSB_DEBUG_CMD`, `LSB_DEBUG_MBD`, `LSB_DEBUG_SBD`, `LSB_DEBUG_SCH`, `LSF_DEBUG_LIM`, `LSF_DEBUG_RES`.

There are also parameters to set the logmask for each of the following daemons separately: `mbatchd`, `sbatchd`, `mbschd`, `lim`, and `res`. For more details, see [configuration reference](#).

The location of log files is specified with the parameter `LSF_LOGDIR` in `lsf.conf`.

You can use the **lsadmin** and **badmin** commands to temporarily change the class, log file, or message log level for specific daemons such as `LIM`, `RES`, `mbatchd`, `sbatchd`, and `mbschd` without changing `lsf.conf`.

How the message log level takes effect

The message log level you set will only be in effect from the time you set it until you turn it off or the daemon stops running, whichever is sooner. If the daemon is restarted, its message log level is reset back to the value of `LSF_LOG_MASK` and the log file is stored in the directory specified by `LSF_LOGDIR`.

Limitations

When debug or timing level is set for `RES` with **lsadmin resdebug**, or **lsadmin restime**, the debug level only affects root `RES`. The root `RES` is the `RES` that runs under the root user ID.

Application `RESs` always use `lsf.conf` to set the debug environment. Application `RESs` are the `RESs` that have been created by `sbatchd` to service jobs and run under the ID of the user who submitted the job.

This means that any RES that has been launched automatically by the LSF system will not be affected by temporary debug or timing settings. The application RES will retain settings specified in `lsf.conf`.

Debug commands for daemons

The following commands set temporary message log level options for LIM, RES, `mbatchd`, `sbatchd`, and `mbschd`.

```
lsadmin limdebug [-c class_name] [-l debug_level] [-f logfile_name] [-o] [host_name]
lsadmin resdebug [-c class_name] [-l debug_level] [-f logfile_name] [-o] [host_name]
badadmin mbddebug [-c class_name] [-l debug_level] [-f logfile_name] [-o] [-s log_queue_size]
badadmin sbddebug [-c class_name] [-l debug_level] [-f logfile_name] [-o] [host_name]
badadmin schddebug [-c class_name] [-l debug_level] [-f logfile_name] [-o] [-s log_queue_size]
```

For a detailed description of **lsadmin** and **badadmin**, see the *IBM® Spectrum LSF Command Reference*.

Examples

```
lsadmin limdebug -c "LC_MULTI LC_PIM" -f myfile hostA hostB
```

Log additional messages for the LIM daemon running on `hostA` and `hostB`, related to MultiCluster and PIM. Create log files in the `LSF_LOGDIR` directory with the name `myfile.lim.log.hostA`, and `myfile.lim.log.hostB`. The debug level is the default value, `LOG_DEBUG` level in parameter `LSF_LOG_MASK`.

```
lsadmin limdebug -o hostA hostB
```

Turn off temporary debug settings for LIM on `hostA` and `hostB` and reset them to the daemon starting state. The message log level is reset back to the value of `LSF_LOG_MASK` and classes are reset to the value of `LSF_DEBUG_RES`, `LSF_DEBUG_LIM`, `LSB_DEBUG_MBD`, `LSB_DEBUG_SBD`, and `LSB_DEBUG_SCH`. The log file is reset to the LSF system log file in the directory specified by `LSF_LOGDIR` in the format `daemon_name.log.host_name`.

```
badadmin sbddebug -o
```

Turn off temporary debug settings for `sbatchd` on the local host (host from which the command was submitted) and reset them to the daemon starting state. The message log level is reset back to the value of `LSF_LOG_MASK` and classes are reset to the value of `LSF_DEBUG_RES`, `LSF_DEBUG_LIM`, `LSB_DEBUG_MBD`, `LSB_DEBUG_SBD`, and `LSB_DEBUG_SCH`. The log file is reset to the LSF system log file in the directory specified by `LSF_LOGDIR` in the format `daemon_name.log.host_name`.

```
badadmin mbddebug -l 1
```

Log messages for `mbatchd` running on the local host and set the log message level to `LOG_DEBUG1`. This command must be submitted from the host on which `mbatchd` is running because `host_name` cannot be specified with **mbddebug**.

```
badadmin mbddebug -s 20000
```

Changes the maximum number of entries in the logging queue that the **mbatchd** logging thread uses to 20000 entries. The logging queue is full when the number of entries in the log queue is 20000. This value temporarily overrides the value of `LSF_LOG_QUEUE_SIZE` in `lsf.conf`, but this value is ignored if `LSF_LOG_THREAD=N` is defined in `lsf.conf`.

```
badadmin sbddebug -f hostB/myfolder/myfile hostA
```

Log messages for `sbatchd` running on `hostA`, to the directory `myfile` on the server `hostB`, with the file name `myfile.sbatchd.log.hostA`. The debug level is the default value, `LOG_DEBUG` level in parameter `LSF_LOG_MASK`.

```
badadmin schddebug -l 2
```

Log messages for `mbatchd` running on the local host and set the log message level to `LOG_DEBUG2`. This command must be submitted from the host on which `mbatchd` is running because `host_name` cannot be specified with **schddebug**.

```
badadmin schddebug -s 20000
```

Changes the maximum number of entries in the logging queue that the **mbschd** logging thread uses to 20000 entries. The logging queue is full when the number of entries in the log queue is 20000. This value temporarily overrides the value of `LSF_LOG_QUEUE_SIZE` in `lsf.conf`, but this value is ignored if `LSF_LOG_THREAD=N` is defined in `lsf.conf`.

```
badadmin schddebug -l 1 -c "LC_PERFM"
badadmin schdtime -l 2
```

Activate the LSF scheduling debug feature.

Log performance messages for mbatchd running on the local host and set the log message level to LOG_DEBUG. Set the timing level for mbschd to include two levels of timing information.

```
lsadmin resdebug -o hostA
```

Turn off temporary debug settings for RES on `hostA` and reset them to the daemon starting state. The message log level is reset back to the value of LSF_LOG_MASK and classes are reset to the value of LSF_DEBUG_RES, LSF_DEBUG_LIM, LSB_DEBUG_MBD, LSB_DEBUG_SBD, and LSB_DEBUG_SCH. The log file is reset to the LSF system log file in the directory specified by LSF_LOGDIR in the format `daemon_name.log.host_name`.

Set daemon timing levels

The timing log level for LSF daemons is set in `lsf.conf` with the parameters LSB_TIME_CMD, LSB_TIME_MBD, LSB_TIME_SBD, LSB_TIME_SCH, LSF_TIME_LIM, LSF_TIME_RES.

The location of log files is specified with the parameter LSF_LOGDIR in `lsf.conf`. Timing is included in the same log files as messages.

To change the timing log level, you need to stop any running daemons, change `lsf.conf`, and then restart the daemons.

It is useful to track timing to evaluate the performance of the LSF system. You can use the **lsadmin** and **badmin** commands to temporarily change the timing log level for specific daemons such as LIM, RES, mbatchd, sbatchd, and mbschd without changing `lsf.conf`.

LSF_TIME_RES is not supported on Windows.

How the timing log level takes effect

The timing log level you set will only be in effect from the time you set it until you turn off the timing log level or the daemon stops running, whichever is sooner. If the daemon is restarted, its timing log level is reset back to the value of the corresponding parameter for the daemon (LSB_TIME_MBD, LSB_TIME_SBD, LSF_TIME_LIM, LSF_TIME_RES). Timing log messages are stored in the same file as other log messages in the directory specified with the parameter LSF_LOGDIR in `lsf.conf`.

Limitations

When debug or timing level is set for RES with **lsadmin resdebug**, or **lsadmin restime**, the debug level only affects root RES. The root RES is the RES that runs under the root user ID.

An application RES always uses `lsf.conf` to set the debug environment. An application RES is the RES that has been created by sbatchd to service jobs and run under the ID of the user who submitted the job.

This means that any RES that has been launched automatically by the LSF system will not be affected by temporary debug or timing settings. The application RES will retain settings that are specified in `lsf.conf`.

Timing level commands for daemons

The total execution time of a function in the LSF system is recorded to evaluate response time of jobs submitted locally or remotely.

The following commands set temporary timing options for LIM, RES, mbatchd, sbatchd, and mbschd.

```
lsadmin limtime [-l timing_level] [-f logfile_name] [-o] [host_name]
lsadmin restime [-l timing_level] [-f logfile_name] [-o] [host_name]
badmin mbdtime [-l timing_level] [-f logfile_name] [-o]
badmin sbdtime [-l timing_level] [-f logfile_name] [-o] [host_name]
badmin schdtime [-l timing_level] [-f logfile_name] [-o]
```

LSF job termination reason logging

When a job finishes, LSF reports the last job termination action it took against the job and logs it into `lsb.acct`.

If a running job exits because of node failure, LSF sets the correct exit information in `lsb.acct`, `lsb.events`, and the job output file. Jobs terminated by a signal from LSF, the operating system, or an application have the signal logged as the LSF exit code. Exit codes are not the same as the termination actions.

- [View logged job exit information \(bacct -l\)](#)
- [View recent job exit information \(bjobs -l\)](#)
- [Termination reasons displayed by bacct, bhist, and bjobs](#)

When LSF detects that a job is terminated, **bacct -l**, **bhist -l**, and **bjobs -l** display a termination reason.

View logged job exit information (bacct -l)

Procedure

Use **bacct -l** to view job exit information logged to `lsb.acct`:

```
bacct -l 328

Accounting information about jobs that are:
- submitted by all users.
- accounted on all projects.
- completed normally or exited
- executed on all hosts.
- submitted to all queues.
- accounted on all service classes.
-----

Job <328>, User <lsfadmin>, Project <default>, Status <EXIT>, Queue <normal>,
      Command <sleep 3600>
Thu Sep 16 15:22:09 2009: Submitted from host <hostA>, CWD <$HOME>;
Thu Sep 16 15:22:20 2009: Dispatched to 1 Task(s) on Hosts <hostA>;
      Allocated 1 Slot(s) on Host(s) <hostA>, Effective RES_REQ
      <select[type== local] order[r15s:pg] >;
Thu Sep 16 15:23:21 2009: Completed <exit>; TERM_RUNLIMIT: job killed after
      reaching LSF run time limit

Accounting information about this job:
      Share group charged </lsfadmin>
      CPU_T      WAIT      TURNAROUND      STATUS      HOG_FACTOR      MEM      SWAP
      0.04       11       72       exit       0.0006       OK       OK
-----

SUMMARY:      ( time unit: second )
Total number of done jobs:      0      Total number of exited jobs:      1
Total CPU time consumed:      0.0      Average CPU time consumed:      0.0
Maximum CPU time of a job:      0.0      Minimum CPU time of a job:      0.0
Total wait time in queues:      11.0
Average wait time in queue:      11.0
Maximum wait time in queue:      11.0      Minimum wait time in queue:      11.0
Average turnaround time:      72 (seconds/job)
Maximum turnaround time:      72      Minimum turnaround time:      72
Average hog factor of a job:      0.00 ( cpu time / turnaround time )
Maximum hog factor of a job:      0.00      Minimum hog factor of a job:      0.00
Total Run time consumed:      64      Average Run time consumed:      64
Maximum Run time of a job:      64      Minimum Run time of a job:      64
...
```

View recent job exit information (bjobs -l)

Procedure

Use **bjobs -l** to view job exit information for recent jobs:

```
bjobs -l 7265

Job <642>, User <user12>, Project <default>, Status <EXIT>, Queue <normal>, Command <perl -e
"while(1){}">
Fri Nov 27 15:06:35 2012: Submitted from host <hostabc>,
CWD <${HOME}/home/lsf/lsf10.1.0.slt/10.1.0/linux2.4-glibc2.3-x86/bin>;
CPULIMIT
1.0 min of hostabc
Fri Nov 27 15:07:59 2012: Started on <hostabc>, Execution Home </home/user12>, Execution CWD
</home/user12/home/lsf/
lsf10.1.0.slt/10.1.0/linux2.4-glibc2.3-x86/bin>;
Fri Nov 27 15:09:30 2012: Exited by signal 24. The CPU time used is 84.0 seconds.
Fri Nov 27 15:09:30 2012: Completed <exit>; TERM_CPULIMIT: job killed after reaching LSF CPU
usage limit.
...
```

Termination reasons displayed by bacct, bhist, and bjobs

When LSF detects that a job is terminated, **bacct -l**, **bhist -l**, and **bjobs -l** display a termination reason.

Table 1. Termination reasons

Keyword displayed by bacct	Termination reason	Integer value logged to JOB_FINISH in lsb.acct
TERM_ADMIN	Job killed by root or LSF administrator	15
TERM_BUCKET_KILL	Job killed with kill-b	23
TERM_CHKPNT	Job killed after checkpointing	13
TERM_CPULIMIT	Job killed after reaching LSF CPU usage limit	12
TERM_CSM_ALLOC	Job killed by LSF due to CSM allocation API error	32
TERM_CWD_NOTEXIST	Current working directory is not accessible or does not exist on the execution host	25
TERM_DATA	Job killed by LSF due to failed data staging	29
TERM_DEADLINE	Job killed after deadline expires	6
TERM_EXTERNAL_SIGNAL	Job killed by a signal external to LSF	17
TERM_FORCE_ADMIN	Job killed by root or LSF administrator without time for cleanup	9
TERM_FORCE_OWNER	Job killed by owner without time for cleanup	8
TERM_KUBE	Job killed by LSF due to Kubernetes integration	33
TERM_LOAD	Job killed after load exceeds threshold	3
TERM_MC_RECALL	Job killed by LSF due to multicluster job recall	30
TERM_MEMLIMIT	Job killed after reaching LSF memory usage limit	16
TERM_OTHER	Member of a chunk job in WAIT state killed and requeued after being switched to another queue.	4
TERM_OWNER	Job killed by owner	14
TERM_PREEMPT	Job killed after preemption	1

Keyword displayed by bacct	Termination reason	Integer value logged to JOB_FINISH in lsb.acct
TERM_PRE_EXEC_FAIL	Job killed after reaching pre-execution retry limit	28
TERM_PROCESSLIMIT	Job killed after reaching LSF process limit	7
TERM_RC	Job killed by LSF when an LSF resource connector execution host is reclaimed by cloud	34
TERM_REMOVE_HUNG_JOB	Job removed from LSF	26
TERM_REQUEUE_ADMIN	Job killed and requeued by root or LSF administrator	11
TERM_REQUEUE_OWNER	Job killed and requeued by owner	10
TERM_REQUEUE_RC	Job killed and requeued when an LSF resource connector execution host is reclaimed by cloud	31
TERM_RMS	Job exited from an RMS system error	18
TERM_RUNLIMIT	Job killed after reaching LSF run time limit	5
TERM_SWAP	Job killed after reaching LSF swap usage limit	20
TERM_THREADLIMIT	Job killed after reaching LSF thread limit	21
TERM_UNKNOWN	LSF cannot determine a termination reason; 0 is logged but TERM_UNKNOWN is not displayed	0
TERM_ORPHAN_SYSTEM	The orphan job was automatically terminated by LSF	27
TERM_WINDOW	Job killed after queue run window closed	2
TERM_ZOMBIE	Job exited while LSF is not available	19

Tip: The integer values logged to the **JOB_FINISH** event in the lsb.acct file and termination reason keywords are mapped in the lsbatch.h file.

Restrictions

- If a queue-level **JOB_CONTROL** is configured, LSF cannot determine the result of the action. The termination reason only reflects what the termination reason could be in LSF.
- LSF cannot be guaranteed to catch any external signals sent directly to the job.
- In IBM® Spectrum LSF multicluster capability, a **brequeue** request sent from the submission cluster is translated to **TERM_OWNER** or **TERM_ADMIN** in the remote execution cluster. The termination reason in the email notification sent from the execution cluster as well as that in the lsb.acct file is set to TERM_OWNER or TERM_ADMIN.

LSF job exit codes

Exit codes are generated by LSF when jobs end due to signals received instead of exiting normally. LSF collects exit codes via the wait3() system call on UNIX platforms. The LSF exit code is a result of the system exit values. Exit codes less than 128 relate to application exit values, while exit codes greater than 128 relate to system signal exit values (LSF adds 128 to system values). Use **bhist** to see the exit code for your job.

How or why the job may have been signaled, or exited with a certain exit code, can be application and/or system specific. The application or system logs might be able to give a better description of the problem.

Note:

Termination signals are operating system dependent, so signal 5 may not be SIGTRAP and 11 may not be SIGSEGV on all UNIX and Linux systems. You need to pay attention to the execution host type in order to correct translate the exit value if the job has been signaled.

Application exit values

The most common cause of abnormal LSF job termination is due to application system exit values. If your application had an explicit exit value less than 128, **bjobs** and **bhist** display the actual exit code of the application; for example, **Exited with exit code 3**. You would have to refer to the application code for the meaning of exit code 3.

It is possible for a job to explicitly exit with an exit code greater than 128, which can be confused with the corresponding system signal. Make sure that applications you write do not use exit codes greater than 128.

System signal exit values

Jobs terminated with a system signal are returned by LSF as exit codes greater than 128 such that `exit_code - 128 = signal_value`. For example, exit code 133 means that the job was terminated with signal 5 (SIGTRAP on most systems, $133 - 128 = 5$). A job with exit code 130 was terminated with signal 2 (SIGINT on most systems, $130 - 128 = 2$).

Some operating systems define exit values as 0-255. As a result, negative exit values or values > 255 may have a wrap-around effect on that range. The most common example of this is a program that exits -1 will be seen with "exit code 255" in LSF.

bhist and bjobs output

In most cases, **bjobs** and **bhist** show the application exit value ($128 + \text{signal}$). In some cases, **bjobs** and **bhist** show the actual signal value.

If LSF sends catchable signals to the job, it displays the exit value. For example, if you run **bkill jobID** to kill the job, LSF passes SIGINT, which causes the job to exit with exit code 130 (SIGINT is 2 on most systems, $128 + 2 = 130$).

If LSF sends uncatchable signals to the job, then the entire process group for the job exits with the corresponding signal. For example, if you run **bkill -s SEGV jobID** to kill the job, **bjobs** and **bhist** show:

```
Exited by signal 7
```

In addition, **bjobs** displays the termination reason immediately following the exit code or signal value. For example:

```
Exited by signal 24. The CPU time used is 84.0 seconds.  
Completed <exit>; TERM_CPULIMIT: job killed after reaching LSF CPU usage limit.
```

Unknown termination reasons appear without a detailed description in the **bjobs** output as follows:

```
Completed <exit>;
```

Example

The following example shows a job that exited with exit code 130, which means that the job was terminated by the owner.

```
bkill 248  
Job <248> is being terminated  
bjobs -l 248  
Job <248>, User <user1>, Project <default>, Status <EXIT>, Queue <normal>, Command  
Sun May 31 13:10:51 2009: Submitted from host <host1>, CWD <${HOME}>;  
Sun May 31 13:10:54 2009: Started on <host5>, Execution Home </home/user1>,  
Execution CWD <${HOME}>;  
Sun May 31 13:11:03 2009: Exited with exit code 130. The CPU time used is 0.9 seconds.  
Sun May 31 13:11:03 2009: Completed <exit>; TERM_OWNER: job killed by owner.  
...
```

Event generation

Learn how LSF detects events occurring during daemon operations. LSF provides a program which translates LSF events into SNMP traps. Certain daemon operations cause **mbatchd** or the parent LIM to call the event program to generate an event. Each LSF event is identified by a predefined number, which is passed as an argument to the event program.

- [Event generation](#)
- [Events list](#)
- [Arguments passed to the LSF event program](#)

Event generation

LSF detects events occurring during the operation of LSF daemons. LSF provides a program which translates LSF events into SNMP traps. You can also write your own program that runs on the management host to interpret and respond to LSF events in other ways. For example, your program could:

- Page the system administrator
- Send email to all users
- Integrate with your existing network management software to validate and correct the problem

On Windows, use the Windows Event Viewer to view LSF events.

SNMP trap program

If you use the LSF SNMP trap program as the event handler, see the SNMP documentation for instructions on how to enable event generation.

- [Enable event generation for custom programs](#)

Enable event generation for custom programs

About this task

If you use a custom program to handle the LSF events, take the following steps to enable event generation.

Procedure

1. Write a custom program to interpret the arguments passed by LSF.
2. To enable event generation, define `LSF_EVENT_RECEIVER` in `lsf.conf`. You must specify an event receiver even if your program ignores it.
The event receiver maintains cluster-specific or changeable information that you do not want to hard-code into the event program. For example, the event receiver could be the path to a current log file, the email address of the cluster administrator, or the host to send SNMP traps to.
3. Set `LSF_EVENT_PROGRAM` in `lsf.conf` and specify the name of your custom event program. If you name your event program `genevent` (`genevent.exe` on Windows) and place it in `LSF_SERVERDIR`, you can skip this step.
4. Reconfigure the cluster with the commands **lsadmin reconfig** and **badmin reconfig**.

Events list

The following daemon operations cause `mbatchd` or the management host LIM to call the event program to generate an event. Each LSF event is identified by a predefined number, which is passed as an argument to the event program. Events 1-9 also return the name of the host on which an event occurred.

1. LIM goes down (detected by the management host LIM). This event may also occur if LIM temporarily stops communicating to the management host LIM.
2. RES goes down (detected by the management host LIM).
3. `sbatchd` goes down (detected by `mbatchd`).

4. A host becomes the new management host (detected by the management host LIM).
5. The management host stops being the management host (detected by the management host LIM).
6. mbatchd comes up and is ready to schedule jobs (detected by mbatchd).
7. mbatchd goes down (detected by mbatchd).
8. mbatchd receives a reconfiguration request and is being reconfigured (detected by mbatchd).
9. LSB_SHAREDIR becomes full (detected by mbatchd).
10. The administrator opens a host.
11. The administrator closes a host.
12. The administrator opens a queue.
13. The administrator closes a queue.
14. mbschd goes down.

Arguments passed to the LSF event program

If LSF_EVENT_RECEIVER is defined, a function called ls_postevent() allows specific daemon operations to generate LSF events. This function then calls the LSF event program and passes the following arguments:

- The event receiver (LSF_EVENT_RECEIVER in lsf.conf)
- The cluster name
- The LSF event number (LSF events list or LSF_EVENT_XXXX macros in lsf.h)
- The event argument (for events that take an argument)

Example

For example, if the event receiver is the string xxx and LIM goes down on HostA in Cluster1, the function returns:

```
xxx Cluster1 1 HostA
```

The custom LSF event program can interpret or ignore these arguments.

Customize batch command messages

About this task

LSF displays error messages when a batch command cannot communicate with mbatchd. Users see these messages when the batch command retries the connection to mbatchd.

You can customize three of these messages to provide LSF users with more detailed information and instructions.

Procedure

1. In the file lsf.conf, identify the parameter for the message that you want to customize.
The following lists the parameters that you can use to customize messages when a batch command does not receive a response from mbatchd.

Reason for no response from mbatchd	Default message	Parameter used to customize the message
mbatchd is too busy to accept new connections or respond to client requests	LSF is processing your request. Please wait...	LSB_MBD_BUSY_MSG
internal system connections to mbatchd fail	Cannot connect to LSF. Please wait...	LSB_MBD_CONNECT_FAIL_MSG

Reason for no response from mbatchd	Default message	Parameter used to customize the message
mbatchd is down or there is no process listening at either the LSB_MBD_PORT or the LSB_QUERY_PORT	LSF is down. Please wait...	LSB_MBD_DOWN_MSG

2. Specify a message string, or specify an empty string:

- To specify a message string, enclose the message text in quotation marks (") as shown in the following example:

```
LSB_MBD_BUSY_MSG="The mbatchd daemon is busy. Your command will retry every 5 minutes.  
No action required."
```

- To specify an empty string, type quotation marks (") as shown in the following example:

```
LSB_MBD_BUSY_MSG=""
```

Whether you specify a message string or an empty string, or leave the parameter undefined, the batch command retries the connection to mbatchd at the intervals specified by the parameters LSB_API_CONNTIMEOUT and LSB_API_RECVTIMEOUT.

Note:

Before Version 7.0, LSF displayed the following message for all three message types: "batch daemon not responding... still trying." To display the previous default message, you must define each of the three message parameters and specify "batch daemon not responding...still trying" as the message string.

3. Save and close the lsf.conf file.

How LIM determines host models and types

The LIM (load information manager) daemon/service automatically collects information about hosts in an LSF cluster, and accurately determines running host models and types. At most, 1024 model types can be manually defined in lsf.shared.

If lsf.shared is not fully defined with all known host models and types found in the cluster, LIM attempts to match an unrecognized running host to one of the models and types that is defined.

LIM supports both exact matching of host models and types, and "fuzzy" matching, where an entered host model name or type is slightly different from what is defined in lsf.shared (or in ego.shared if EGO is enabled in the LSF cluster).

How does "fuzzy" matching work?

LIM reads host models and types that are manually configured in lsf.shared. The format for entering host models and types is *model_bogomips_architecture* (for example, x15_4604_OpteronProcessor142, IA64_2793, or SUNWUltra510_360_sparc). Names can be up to 64 characters long.

When LIM attempts to match running host model with what is entered in lsf.shared, it first attempts an exact match, then proceeds to make a fuzzy match.

How LIM attempts to make matches

Architecture name of running host	What the lim reports	Additional information about the lim process
Same as definition in lsf.shared (exact match)	Reports the reference index of exact match	LIM detects an exact match between model and input architecture string

Architecture name of running host	What the lim reports	Additional information about the lim process
Similar to what is defined in lsf.shared (fuzzy match)	Reports fuzzy match that is based on detection of 1 or 2 fields in the input architecture string	<ul style="list-style-type: none"> For input architecture strings with only one field, if LIM cannot detect an exact match for the input string, then it reports the <i>best match</i>. A best match is a model field with the most characters shared by the input string.
		<ul style="list-style-type: none"> For input architecture strings with two fields: <ol style="list-style-type: none"> If LIM cannot detect an exact match, it attempts to find a best match by identifying the <i>model</i> field with the most characters that match the input string LIM then attempts to find the best match on the <i>bogomips</i> field
		<ul style="list-style-type: none"> For architecture strings with three fields: <ol style="list-style-type: none"> If LIM cannot detect an exact match, it attempts to find a best match by identifying the <i>model</i> field with the most characters that match the input string After finding the best match for the model field, LIM attempts to find the best match on the <i>architecture</i> field LIM then attempts to find the closest match on the <i>bogomips</i> field, with wildcards supported (where the <i>bogomips</i> field is a wildcard)
Has an illegal name	Reports default host model	An illegal name is one that does not follow the permitted format for entering an architecture string where the first character of the string is not an English-language character.

- [Automatically detect operating system types and versions](#)
- [Add a custom host type or model](#)
- [Automatic detection of hardware reconfiguration](#)

Automatically detect operating system types and versions

About this task

LSF can automatically detect most operating system types and versions so that you do not need to add them to the lsf.shared file manually. The list of automatically detected operating systems is updated regularly.

Procedure

1. Edit lsf.shared.
2. In the Resource section, define any of the following lines:

```

os          String  ()      ()      ()      (Operating System)
ostype      String  ()      ()      ()      (Operating System and version)
osmajor     String  ()      ()      ()      (Operating System and major
version)
kernel      String  ()      ()      ()      (Operating System kernel level)
```

The **eslim.ostype** executable file detects four resources in the system. You can define any of these resources in the lsf.shared file.

3. In `$LSF_SERVERDIR`, rename `tmp.eslim.ostype` to `eslim.ostype`.
4. Run the following commands to restart the LIM and management batch daemon:
 - a. **`lsadmin reconfig`**
 - b. **`badmin mbdrestart`**
5. To view operating system types and versions, run **`lshosts -l`** or **`lshosts -s`**.
LSF displays the operating system types and versions in your cluster, including any that LSF automatically detects as well as those you have defined manually in the HostType section of `lsf.shared`.

Results

You can specify `ostype` in your resource requirement strings. For example, when submitting a job you can specify the following resource requirement: `-R "select[ostype=RHEL2.6]"`.

Modify how long LSF waits for new operating system types and versions

Before you begin

You must enable LSF to automatically detect operating system types and versions.

About this task

You can configure how long LSF waits for OS type and version detection.

Procedure

In `lsf.conf`, modify the value for `EGO_ESLIM_TIMEOUT`.
The value is time in seconds.

Add a custom host type or model

About this task

The `lsf.shared` file contains a list of host type and host model names for most operating systems. You can add to this list or customize the host type and host model names. A host type and host model name can be any alphanumeric string up to 39 characters long.

Procedure

1. Log on as the LSF administrator on any host in the cluster.
2. Edit `lsf.shared`:
 - a. For a new host type, modify the HostType section:

```
Begin HostType
TYPENAME                                # Keyword
DEFAULT
IBMAIX564
LINUX86
LINUX64
NTX64
NTIA64
SUNSOL
SOL732
SOL64
SGI658
SOLX86
HPPA11
HPUXIA64
```

```
MACOSX
End HostType
```

- b. For a new host model, modify the HostModel section:
Add the new model and its CPU speed factor relative to other models.

```
Begin HostModel
MODELNAME  CPUFACTOR  ARCHITECTURE # keyword
# x86 (Solaris, Windows, Linux): approximate values, based on SpecBench results
# for Intel processors (Sparc/Win) and BogoMIPS results (Linux).
PC75      1.5      (i86pc_75 i586_75 x586_30)
PC90      1.7      (i86pc_90 i586_90 x586_34 x586_35 x586_36)
HP9K715   4.2      (HP9000715 100)
SunSparc   12.0      ()
CRAYJ90    18.0      ()
IBM350     18.0      ()
End HostModel
```

3. Save the changes to `lsf.shared`.
4. Run **lsadmin reconfig** to reconfigure LIM.
5. Run **badmin reconfig** to reconfigure `mbatchd`.

Automatic detection of hardware reconfiguration

Some UNIX operating systems support dynamic hardware reconfiguration; that is, the attaching or detaching of system boards in a live system without having to reboot the host.

Supported platforms

LSF is able to recognize changes in `ncpus`, `maxmem`, `maxswp`, `maxtmp` in the following platforms:

- Sun Solaris 10 and 11+
- HP UX 11
- IBM AIX 5, 6 and 7 on IBM POWER

Dynamic changes in `ncpus`

LSF is able to automatically detect a change in the number of processors in systems that support dynamic hardware reconfiguration.

The local LIM checks if there is a change in the number of processors at an internal interval of 2 minutes. If it detects a change in the number of processors, the local LIM also checks `maxmem`, `maxswp`, `maxtmp`. The local LIM then sends this new information to the management host LIM.

Dynamic changes in `maxmem`, `maxswp`, `maxtmp`

If you dynamically change `maxmem`, `maxswp`, or `maxtmp` without changing the number of processors, you need to restart the local LIM with the command `lsadmin limrestart` so that it can recognize the changes.

If you dynamically change the number of processors and any of `maxmem`, `maxswp`, or `maxtmp`, the change is automatically recognized by LSF. When it detects a change in the number of processors, the local LIM also checks `maxmem`, `maxswp`, `maxtmp`.

View dynamic hardware changes

lsxxx Commands

There may be a 2-minute delay before the changes are recognized by **lsxxx** commands (for example, before **lshosts** displays the changes).

bxxx Commands

There may be at most a 2 + 10 minute delay before the changes are recognized by **bx** commands (for example, before **bhosts -l** displays the changes).

This is because **mbatchd** contacts the management host LIM at an internal interval of 10 minutes.

Platform MultiCluster

Configuration changes from a local cluster are communicated from the management host LIM to the remote cluster at an interval of $2 * \text{CACHE_INTERVAL}$. The parameter `CACHE_INTERVAL` is configured in `lsf.cluster.cluster_name` and is by default 60 seconds.

This means that for changes to be recognized in a remote cluster there is a maximum delay of 2 minutes + $2 * \text{CACHE_INTERVAL}$.

How dynamic hardware changes affect LSF

LSF uses `ncpus`, `maxmem`, `maxswp`, `maxtmp` to make scheduling and load decisions.

When processors are added or removed, LSF licensing is affected because LSF licenses are based on the number of processors.

If you put a processor offline, dynamic hardware changes have the following effects:

- Per host or per-queue load thresholds may be exceeded sooner. This is because LSF uses the number of CPUs and relative CPU speeds to calculate effective run queue length.
- The value of CPU run queue lengths (`r15s`, `r1m`, and `r15m`) increases.
- Jobs may also be suspended or not dispatched because of load thresholds.
- Per-processor job slot limit (`PJOB_LIMIT` in `lsb.queues`) may be exceeded sooner.

If you put a new processor online, dynamic hardware changes have the following effects:

- Load thresholds may be reached later.
- The value of CPU run queue lengths (`r15s`, `r1m`, and `r15m`) is decreased.
- Jobs suspended due to load thresholds may be resumed.
- Per-processor job slot limit (`PJOB_LIMIT` in `lsb.queues`) may be reached later.
- [Set the external static LIM](#)

Set the external static LIM

About this task

Use the external static LIM to automatically detect the operating system type and version of hosts.

Procedure

1. In `lsf.shared`, uncomment the indices that you want detected.
2. In `$LSF_SERVERDIR`, rename `tmp.eslim.<extension>` to `eslim.<extension>`.
3. Set `EGO_ESLIM_TIMEOUT` in `lsf.conf` or `ego.conf`.
4. Restart the `lim` on all hosts.

Shared file access

A frequent problem with LSF is non-accessible files due to a non-uniform file space. If a task is run on a remote host where a file it requires cannot be accessed using the same name, an error results. Almost all interactive LSF commands fail if the user's current working directory cannot be found on the remote host.

Shared files on UNIX

If you are running NFS, rearranging the NFS mount table may solve the problem. If your system is running the **automount** server, LSF tries to map the filenames, and in most cases it succeeds. If shared mounts are used, the mapping may break for those files. In such cases, specific measures need to be taken to get around it.

The automount maps must be managed through NIS. When LSF tries to map filenames, it assumes that automounted file systems are mounted under the /tmp_mnt directory.

Shared files across UNIX and Windows

For file sharing across UNIX and Windows, you require a third-party NFS product on Windows to export directories from Windows to UNIX.

- [Shared files on Windows](#)
- [Use LSF with non-shared file systems](#)

Shared files on Windows

Procedure

To share files among Windows machines, set up a share on the server and access it from the client. You can access files on the share either by specifying a UNC path (\\server\share\path) or connecting the share to a local drive name and using a drive:\path syntax. Using UNC is recommended because drive mappings may be different across machines, while UNC allows you to unambiguously refer to a file on the network.

Use LSF with non-shared file systems

Procedure

1. Follow the complete installation procedure on every host to install all the binaries, man pages, and configuration files.
2. Update the configuration files on all hosts so that they contain the complete cluster configuration.
Configuration files must be the same on all hosts.
3. Choose one host to act as the LSF management host.
 - a. Install LSF configuration files and working directories on this host
 - b. Edit `lsf.cluster.cluster_name` and list this host first.
 - c. Use the parameter `LSF_MASTER_LIST` in `lsf.conf` to set management host candidates.
For Windows password authentication in a non-shared file system environment, you must define the parameter `LSF_MASTER_LIST` in `lsf.conf` so that jobs will run with correct permissions. If you do not define this parameter, LSF assumes that the cluster uses a shared file system environment.

Note:

Fault tolerance can be introduced by choosing more than one host as a possible management host, and using NFS to mount the LSF working directory on only these hosts. All the possible management hosts must be listed first in `lsf.cluster.cluster_name`. As long as one of these hosts is available, LSF continues to operate.

Shared configuration file content

Use the `#INCLUDE` directive to insert the contents of the specified file into a configuration file to share common configurations between clusters or hosts.

About this task

LSF allows common configurations to be shared by all clusters (when you use the LSF multicluster capability) and by all hosts. In addition, configuration for the LSF multicluster capability requires that you keep the `lsf.shared` configuration files consistent between clusters for definitions and resources that are used by the LSF multicluster capability. To facilitate this requirement, there is an `#INCLUDE` directive to clearly partition information that is common to all clusters or hosts. The `#INCLUDE` directive specifies a file whose contents are inserted into the current configuration file. There is no need to create a primary copy for distribution, which can then have local customizations with definition changes that could cause unpredictable behavior for jobs that use the LSF multicluster capability. The `#INCLUDE` directive simplifies application maintenance by allowing a relevant power user to own and modify each application-specific include file. The `#INCLUDE` directive also separates portions of the LSF configuration files to allow specific users or user groups to own and modify their portion of the configuration.

You can use the `#INCLUDE` directive in any place in the following configuration files:

- `lsb.applications`
- `lsb.hosts`
- `lsb.queues`
- `lsb.reasons`
- `lsb.resources`
- `lsb.users`

You can only use the `#INCLUDE` directive at the beginning of the `lsf.shared` file.

Procedure

1. Create the common configuration files and send them to the local administrators. The format for the common configuration files is the same as the existing LSF configuration files. Reserved words and structures (such as `#if`, `#elif`, `#else`, and `#endif`; or the structures that are specific to LSF configuration files) must be entirely contained in a single `#include` file and cannot cross multiple files. That is, you cannot have `#if` and `#elif` in one file with `#else` and `#endif` in an `#include` file. If you have an `#if` statement in one file, it must have a corresponding `#endif` statement in the same file.

For example, the following configuration for the `lsb.users` file results in an error because the `#if` and `#endif` statements are in different files:

- Contents of the `lsb.users` file:

```
Begin User
USER_NAME    MAX_JOBS    JL/P    MAX_PEND_JOBS    PRIORITY
user1        10         -        1000         200
user2         4         -         -         100
user3         -         -         -         300
groupC        -         -         500         100
default       6         1         10          0
#if time(07:00-09:00)
groupA        10         1       100000         400
groupA@       -         1         100          200
#else
#include "/home/work/lsb.users_groupA"
End User
```

- Contents of the `lsb.users_groupA` file:

```
#USER_NAME    MAX_JOBS    JL/P    MAX_PEND_JOBS    PRIORITY
groupA         5         1         1000         300
groupA@        -         1         100          200
#endif
```

The following configuration for the `lsb.users` file results in an error because the `Begin User` and `End User` lines are in different files:

- Contents of the `lsb.users` file:

```
Begin User
USER_NAME    MAX_JOBS    JL/P    MAX_PEND_JOBS    PRIORITY
```

```

user1      10      -      1000      200
user2      4       -       -       100
user3      -       -       -       300
groupC     -       -       500      100
default    6       1       10       0
#include "/home/work/lsb.users_groupA"

```

- Contents of the lsb.users_groupA file:

```

#USER_NAME  MAX_JOBS  JL/P  MAX_PEND_JOBS  PRIORITY
groupA      5         1      1000           300
groupA@     -         1      100           200
End User

```

The following configuration for the lsb.users file is correct:

- Contents of the lsb.users file:

```

Begin User
USER_NAME  MAX_JOBS  JL/P  MAX_PEND_JOBS  PRIORITY
user1      10      -      1000           200
user2      4       -       -           100
user3      -       -       -           300
#include "/home/work/lsb.users_groupA"
groupC     -       -       500           100
default    6       1       10            0
End User

```

- Contents of the lsb.users_groupA file:

```

#USER_NAME  MAX_JOBS  JL/P  MAX_PEND_JOBS  PRIORITY
#if time (07:00-09:00)
groupA      10         1      100000         400
groupA@     -         1      100            200
#else
groupA      5         1      1000           300
groupA@     -         1      100            200
#endif

```

2. Ensure that the local administrators for each cluster open their local configuration files and add the `#include "path_to_file"` syntax to them.

Grant write access to the intended owners of the shared common files that are specified in the `#include` directive so that they can maintain their own files. These owners do not necessarily have write access to the entire configuration file, which limits the changes that they can make to the entire cluster and ensures that only the local administrator can affect the system.

All `#include` lines must be inserted at the beginning of the local `lsf.shared` file. If `#include` lines are placed within or after any other sections in `lsf.shared`, LSF reports an error.

You can insert the `#include` lines anywhere in the other configuration files.

For example, `lsf.shared` file has four included common files that represent the `HostType`, `HostModel`, and `Resource` sections. Different users can own the different common files, so one user can be responsible for maintaining the list of hardware types and operating systems, one can be responsible for the list of machine model names and CPU factors, and one can be responsible for the list of LSF resources. The `Cluster` section is different for each cluster when using the LSF multicluster capability, so this section is not used in a shared common file.

```

#include "/Shared/lsf.shared.HostType" # Comments
#include "/Shared/lsf.shared.HostModel"
#include "/Shared/lsf.shared.Resource"
Begin Cluster
ClusterName Servers
cluster1     hostA
cluster2     hostB
End Cluster

```

The following `lsb.queues` file has three included common files that represent three different queues. This allows different users (such as queue administrators) to own each file and be responsible for their own queues. The file also

has one included common file that specifies the users who can submit jobs to the `normal` queue. This allows a user, for example, to control the users or user groups that have access to the `normal` queue.

```
#INCLUDE "/Shared/lsb.queues.interactive"
#INCLUDE "/Shared/lsb.queues.priority"
#INCLUDE "/Shared/lsb.queues.night"
Begin Queue
QUEUE_NAME = normal
PRIORITY = 30
STACKLIMIT= 2048
DESCRIPTION = For normal low priority jobs, running only if hosts are lightly loaded.
QJOB_LIMIT = 60      # job limit of the queue
PJOB_LIMIT = 2       # job limit per processor
ut = 0.2
io = 50/240
#INCLUDE "/Shared/lsb.queues.normal.users" #USERS parameter
HOSTS = all
NICE = 20
End Queue
```

The following `lsb.applications` file has four included common files that represent four different application profiles. This allows different users to own each file and be responsible for their own application profiles. The file also has one included common file that specifies the pre-execution and post-execution scripts for the `4proc` application. This allows a user, for example, to change the scripts for this particular application.

```
#INCLUDE "/Shared/lsb.applications.dockerapp"
#INCLUDE "/Shared/lsb.applications.shifterapp"
#INCLUDE "/Shared/lsb.applications.singapp"
Begin Application
NAME          = 4proc
RUNTIME       = 120      # scheduling hint of 15 minutes
TASKLIMIT     = 4 4 4
RES_REQ       = span[hosts=1]
#INCLUDE "/Shared/lsb.applications.4proc.scripts" #PRE_EXEC/POST_EXEC parameters
End Application
#INCLUDE "/Shared/lsb.applications.catia"
```

If there is any duplicate configuration between the common configuration file and the local ones, the common one takes effect and LSF reports an error. If there are fewer columns in the common files than the local configurations, LSF provides default values for the undefined columns. If there are more columns in the common files than the local configurations, LSF ignores the lines with more columns and reports an error.

3. Make the configuration active.

For more information about how to apply the changes and make the configuration active, see the configuration files that you modified.

Authentication and authorization

LSF uses authentication and authorization to ensure the security of your cluster. The authentication process verifies the identity of users, hosts, and daemons, depending on the security requirements of your site. The authorization process enforces user account permissions.

- [Change authentication method](#)
- [Authentication options](#)
- [Operating system authorization](#)
- [LSF authorization](#)
- [Authorization failure](#)
- [External authentication](#)

The external authentication feature provides a framework that enables you to integrate LSF with any third-party authentication product—such as DCE Security Services—to authenticate users, hosts, and daemons. This feature provides a secure transfer of data within the authentication data stream between LSF clients and servers. Using external authentication, you can customize LSF to meet the security requirements of your site.

- [Kerberos authentication](#)

Kerberos authentication is an extension of external daemon authentication, providing authentication of LSF users and

Change authentication method

About this task

During LSF installation, the authentication method is set to external authentication (eauth), which offers the highest level of security.

Procedure

Set LSF_AUTH in lsf.conf.

- For external authentication (the default), set LSF_AUTH=eauth
- For authentication using the identd daemon, set LSF_AUTH=ident
- For privileged port authentication, leave LSF_AUTH undefined

Note:

If you change the authentication method while LSF daemons are running, you must shut down and restart the daemons on all hosts in order to apply the changes.

When the external authentication (eauth) feature is enabled, you can also configure LSF to authenticate daemons by defining the parameter LSF_AUTH_DAEMONS in lsf.conf.

All authentication methods supported by LSF depend on the security of the root account on all hosts in the cluster.

Authentication options

Authentication method	Description	Configuration	Behavior
External authentication	<ul style="list-style-type: none">• A framework that enables you to integrate LSF with any third-party authentication product—such as Kerberos or DCE Security Services—to authenticate users, hosts, and daemons. This feature provides a secure transfer of data within the authentication data stream between LSF clients and servers. Using external authentication, you can customize LSF to meet the security requirements of your site.	LSF_AUTH=eauth	<ul style="list-style-type: none">• LSF uses the default eauth executable located in LSF_SERVERDIR. The default executable provides an example of how the eauth protocol works. You should write your own eauth executable to meet the security requirements of your cluster. For a detailed description of the external authentication feature and how to configure it, see External authentication.

Authentication method	Description	Configuration	Behavior
Identification daemon (identd)	<ul style="list-style-type: none"> Authentication using the identd daemon available in the public domain. 	LSF_AUTH=ident	<ul style="list-style-type: none"> LSF uses the identd daemon available in the public domain. LSF supports both RFC 931 and RFC 1413 protocols.
Privileged ports (setuid)	<ul style="list-style-type: none"> User authentication between LSF clients and servers on UNIX hosts only. An LSF command or other executable configured as setuid uses a reserved (privileged) port number (1-1024) to contact an LSF server. The LSF server accepts requests received on a privileged port as coming from the root user and then runs the LSF command or other executable using the real user account of the user who issued the command. 	LSF_AUTH not defined	<ul style="list-style-type: none"> For UNIX hosts only, LSF clients (API functions) use reserved ports 1-1024 to communicate with LSF servers. The number of user accounts that can connect concurrently to remote hosts is limited by the number of available privileged ports. LSF_AUTH must be deleted or commented out and LSF commands must be installed as setuid programs owned by root. <p>Note: To enable the setuid bit on the LSF commands, run the hostsetup --setuid command option on the LSF parent and candidate hosts. Since this allows LSF administration commands to run with root privileges, do not enable the setuid bit if you do not want these LSF commands to run with root privileges.</p> <p>The hostsetup --setuid command enables the setuid bit for the following LSF executable files: badmin, lsadmin, egosh, utmpreg, swtbl_api, ntbl_api, lstbl_nid, and swtbl_poe.</p>

UNIX user and host authentication

The primary LSF administrator can configure additional authentication for UNIX users and hosts by defining the parameter LSF_USE_HOSTEQUIV in the lsf.conf file. With LSF_USE_HOSTEQUIV defined, mbatchd on the parent host and RES on the remote host call the **ruserok(3)** function to verify that the originating host is listed in the /etc/hosts.equiv file and that the host and user account are listed in the \$HOME/.rhosts file. Include the name of the local host in both files. This additional level of authentication works in conjunction with eauth, privileged ports (setuid), or identd authentication.

CAUTION:

Using the /etc/hosts.equiv and \$HOME/.rhosts files grants permission to use the **rlogin** and **rsh** commands without requiring a password.

SSH

SSH is a network protocol that provides confidentiality and integrity of data using a secure channel between two networked devices. Use SSH to secure communication between submission, execution, and display hosts.

A frequently used option is to submit jobs with SSH X11 forwarding (**bsub -XF**), which allows a user to log into an X-Server client, access the submission host through the client, and run an interactive X-Window job, all through SSH.

Strict checking protocol in an untrusted environment

Note: `LSF_STRICT_CHECKING=ENHANCED` is now the default setting and this section might be removed in a future version of LSF.

To improve security in an untrusted environment, the primary LSF administrator can enable the use of a strict checking communications protocol. When you define `LSF_STRICT_CHECKING` in `lsf.conf`, LSF authenticates messages passed between LSF daemons and between LSF commands and daemons. This type of authentication is *not* required in a secure environment, such as when your cluster is protected by a firewall.

Important: You must shut down the cluster before adding or deleting the `LSF_STRICT_CHECKING` parameter.

Authentication failure

If authentication fails (the user's identity cannot be verified), LSF displays the following error message after a user issues an LSF command:

User permission denied

This error has several possible causes depending on the authentication method used.

Authentication method	Possible cause of failure
eauth	<ul style="list-style-type: none">External authentication failed
identd	<ul style="list-style-type: none">The identification daemon is not available on the local or submitting host
setuid	<ul style="list-style-type: none">The LSF applications are not installed setuidThe NFS directory is mounted with the nosuid option
ruserok	<ul style="list-style-type: none">The client (local) host is not found in either the <code>/etc/hosts.equiv</code> or the <code>\$HOME/.rhosts</code> file on the parent or remote host

Operating system authorization

By default, an LSF job or command runs on the execution host under the user account that submits the job or command, with the permissions associated with that user account. Any UNIX or Windows user account with read and execute permissions for LSF commands can use LSF to run jobs—the LSF administrator does not need to define a list of LSF users. User accounts must have the operating system permissions required to execute commands on remote hosts. When users have valid accounts on all hosts in the cluster, they can run jobs using their own account permissions on any execution host.

Windows passwords

Windows users must register their Windows user account passwords with LSF by running the command **lspasswd**. If users change their passwords, they must use this command to update LSF. A Windows job does not run if the password is not registered in LSF. Passwords must be 31 characters or less.

For Windows password authorization in a non-shared file system environment, you must define the parameter `LSF_MASTER_LIST` in `lsf.conf` so that jobs run with correct permissions. If you do not define this parameter, LSF assumes that

the cluster uses a shared file system environment.

LSF authorization

As an LSF administrator, you have the following authorization options:

- Enable one or more types of user account mapping
- Specify the user account that is used to run `eauth` and `eexec` executables or queue level commands for pre- and post-execution processing
- Control user access to LSF resources and functionality

Enable user account mapping

You can configure different types of user account mapping so that a job or command submitted by one user account runs on the remote host under a different user account.

Type of account mapping	Description
Between-host	Enables job submission and execution within a cluster that has different user accounts assigned to different hosts. Using this feature, you can map a local user account to a different user account on a remote host.
Cross-cluster	Enables cross-cluster job submission and execution for a MultiCluster environment that has different user accounts assigned to different hosts. Using this feature, you can map user accounts in a local cluster to user accounts in one or more remote clusters.
UNIX/Windows	Enables cross-platform job submission and execution in a mixed UNIX/Windows environment. Using this feature, you can map Windows user accounts, which include a domain name, to UNIX user accounts, which do not include a domain name, for user accounts with the same user name on both operating systems.

For a detailed description of the user account mapping features and how to configure them, see [UNIX and Windows user account mapping](#).

Specify a user account

To change the user account for ...	Define the parameter ...	In the file ...
<code>eauth</code>	<code>LSF_EAUTH_USER</code>	<code>lsf.sudoers</code>
<code>eexec</code>	<code>LSF_EEXEC_USER</code>	
Pre- and post execution commands	<code>LSB_PRE_POST_EXEC_USER</code>	

Control user access to LSF resources and functionality

If you want to ...	Define ...	In the file ...	Section ...
Specify the user accounts with cluster administrator privileges	<code>ADMINISTRATORS</code>	<code>lsf.cluster.cluster_name</code>	ClusterAdmins
Allow the root user to run jobs on a remote host	<code>LSF_ROOT_REX</code>	<code>lsf.conf</code>	N/A
Allow user accounts other than root to start LSF daemons Note: For information about how to configure the LSF daemon startup control feature, see LSF daemon startup control .	<code>LSF_STARTUP_USERS</code> <code>LSF_STARTUP_PATH</code>	<code>lsf.sudoers</code>	N/A

Authorization failure

Symptom	Probable cause	Solution
User receives an email notification that LSF has placed a job in the USUSP state.	The job cannot run because the Windows password for the job is not registered with LSF.	The user should <ul style="list-style-type: none"> Register the Windows password with LSF using the command lspasswd. Use the bresume command to resume the suspended job.
LSF displays one of the following error messages: <ul style="list-style-type: none"> <code>findHostbyAddr/<proc>: Host <host>/<port> is unknown by <myhostname></code> <code>function: Gethostbyaddr(<host>/<port>) failed: error</code> <code>main: Request from unknown host <host>/<port>: error</code> <code>function: Received request from non-LSF host <host>/<port></code> 	The LSF daemon does not recognize <i>host</i> as part of the cluster. These messages can occur if you add <i>host</i> to the configuration files without reconfiguring all LSF daemons.	Run the following commands after adding a host to the cluster: <ul style="list-style-type: none"> lsadmin reconfig badmin mbdrestart <p>If the problem still occurs, the host might have multiple addresses. Match all of the host addresses to the host name by either:</p> <ul style="list-style-type: none"> Modifying the system hosts file (/etc/hosts). The changes affect all software programs on your system. Creating an LSF hosts file (EGO_CONFDIR/hosts). Only LSF resolves the addresses to the specified host.
<ul style="list-style-type: none"> <code>doacceptconn: getpwnam(<username> @<host>/<port>) failed: error</code> <code>doacceptconn: User <username> has uid <uid1> on client host <host>/<port>, uid <uid2> on RES host; assume bad user</code> <code>authRequest: username/uid <userName>/<uid>@<host>/<port> does not exist</code> <code>authRequest: Submitter's name <clname>@<clhost> is different from name <lname> on this host</code> 	RES assumes that a user has the same UNIX user name and user ID on all LSF hosts. These messages occur if this assumption is violated.	If the user is allowed to use LSF for interactive remote execution, make sure the user's account has the same user ID and user name on all LSF hosts.
<ul style="list-style-type: none"> <code>doacceptconn: root remote execution permission denied</code> <code>authRequest: root job submission rejected</code> 	The root user tried to execute or submit a job but LSF_ROOT_REX is not defined in lsf.conf.	To allow the root user to run jobs on a remote host, define LSF_ROOT_REX in lsf.conf.

Symptom	Probable cause	Solution
<ul style="list-style-type: none"> resControl: operation permission denied, uid = <uid> 	<p>The user with user ID <i>uid</i> is not allowed to make RES control requests. By default, only the LSF administrator can make RES control requests.</p>	<p>To allow the root user to make RES control requests, define LSF_ROOT_REX in lsf.conf.</p>
<ul style="list-style-type: none"> do_restartReq: Failed to get hData of host <host_name>/<host_addr> 	<p>mbatchd received a request from sbatchd on host <i>host_name</i>, but that host is not known to mbatchd. Either</p> <ul style="list-style-type: none"> The configuration file has been changed but mbatchd has not been reconfigured. <i>host_name</i> is a client host but sbatchd is running on that host. 	<p>To reconfigure mbatchd, run the command admin reconfig</p> <p>To shut down sbatchd on <i>host_name</i>, run the command bctrl stop sbd host_name</p>

External authentication

The external authentication feature provides a framework that enables you to integrate LSF with any third-party authentication product—such as DCE Security Services—to authenticate users, hosts, and daemons. This feature provides a secure transfer of data within the authentication data stream between LSF clients and servers. Using external authentication, you can customize LSF to meet the security requirements of your site.

- [External authentication with IBM Spectrum LSF \(eauth\)](#)
 The external authentication feature uses an executable file called eauth. You can write an eauth executable that authenticates users, hosts, and daemons that use a site-specific authentication method such as Kerberos or DCE Security Services client authentication. You can also specify an external encryption key (recommended) and the user account under which eauth runs.
- [Configuration to enable external authentication](#)
 During LSF installation, the parameter LSF_AUTH in the lsf.conf file is set to eauth, which enables external authentication. A default **eauth** executable file is installed in the directory that is specified by the parameter LSF_SERVERDIR in the lsf.conf file.
- [External authentication behavior](#)
- [Configuration to modify external authentication](#)
 You can modify external authentication behavior by writing your own eauth executable and by modifying configuration parameters.
- [External authentication commands](#)

External authentication with IBM® Spectrum LSF (eauth)

The external authentication feature uses an executable file called eauth. You can write an eauth executable that authenticates users, hosts, and daemons that use a site-specific authentication method such as Kerberos or DCE Security Services client authentication. You can also specify an external encryption key (recommended) and the user account under which eauth runs.

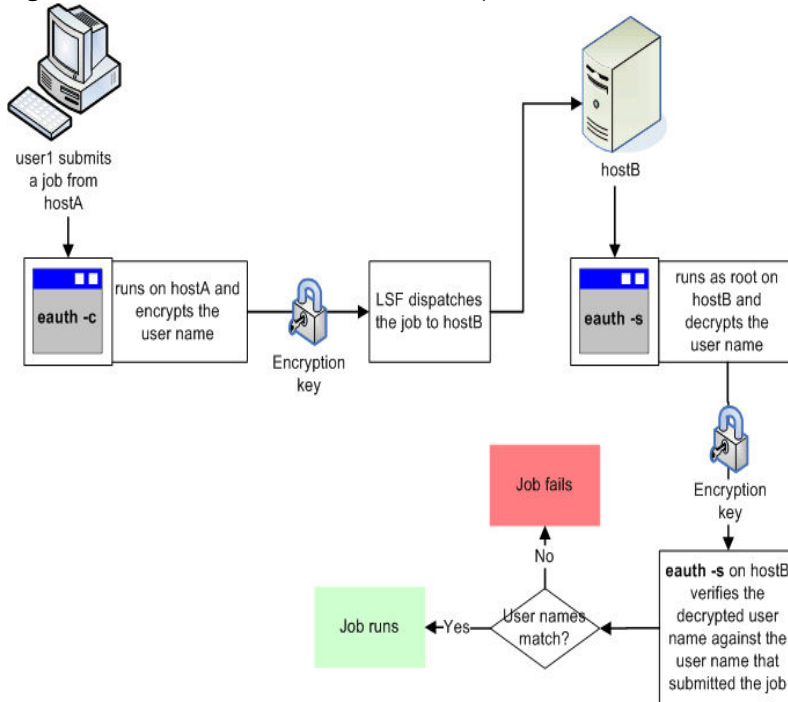
Important: LSF uses an internal encryption key by default. To increase security, configure an external encryption key by defining the parameter LSF_EAUTH_KEY in lsf.sudoers. To use the lsf.sudoers file, you must enable the setuid bit for the LSF administration commands. Run the **hostsetup --setuid** command option on the LSF management and candidate hosts. Since this allows LSF administration commands to run with root privileges, do not enable the setuid bit if you do not want these LSF commands to run with root privileges.

The **hostsetup --setuid** command enables the setuid bit for the following LSF executable files: **admin**, **lsadmin**, **egosh**, **utmpreg**, **swtbl_api**, **ntbl_api**, **lstbl_nid**, and **swtbl_poe**.

During LSF installation, a default eauth executable is installed in the directory that is specified by the parameter LSF_SERVERDIR (set by cshrc.lsf and profile.lsf). The default executable provides an example of how the eauth protocol works.

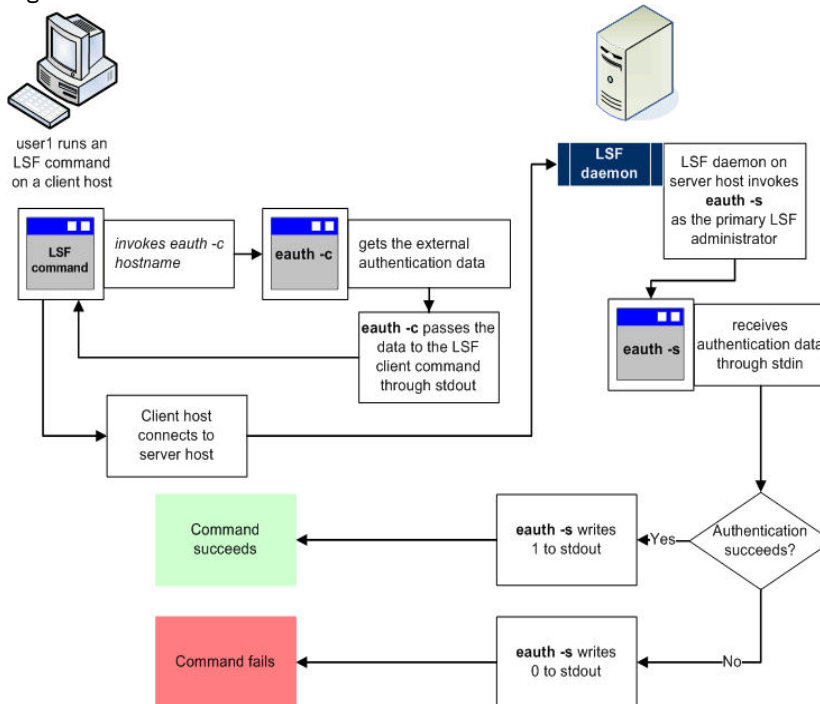
Write your own eauth executable based on this example to meet the security requirements of your cluster.

Figure 1. Default behavior (**eauth** executable provided with LSF)



The eauth executable uses corresponding processes `eauth -c host_name` (client) and `eauth -s` (server) to provide a secure data exchange between LSF daemons on client and server hosts. The variable `host_name` refers to the host on which `eauth -s` runs; that is, the host called by the command. For **bsub**, for example, the `host_name` is NULL, which means the authentication data works for any host in the cluster.

Figure 2. How **eauth** works



One `eauth -s` process can handle multiple authentication requests. If `eauth -s` terminates, the LSF daemon invokes another instance of `eauth -s` to handle new authentication requests.

The standard input stream to `eauth -s` is a text string with the following format:

```
uid gid user_name client_addr client_port user_auth_data_len eauth_client
eauth_server aux_data_file aux_data_status user_auth_data
```

The following table explains the format of the text stream:

Variable	Represents
<i>uid</i>	User ID of the client user
<i>gid</i>	Group ID of the client user
<i>user_name</i>	User name of the client user
<i>client_addr</i>	IP address of the client host
<i>client_port</i>	Port number from which the client request originates.
<i>user_auth_data_len</i>	Length of the external authentication data that is passed from the client host.
<i>eauth_client</i>	Daemon or user that invokes the eauth -cc command.
<i>eauth_server</i>	Daemon that invokes the eauth -s command.
<i>aux_data_file</i>	Location of the temporary file that stores encrypted authentication data.
<i>aux_data_status</i>	File in which eauth -s stores authentication status. When used with Kerberos authentication, eauth -s writes the source of authentication to this file if authentication fails. For example, if <i>mbatchd</i> to <i>mbatchd</i> authentication fails, eauth -s writes "mbatchd" to the file defined by <i>aux_data_status</i> . If user to <i>mbatchd</i> authentication fails, eauth -s writes "user" to the <i>aux_data_status</i> file.
<i>user_auth_data</i>	External authentication data that is passed from the client host.

The variables that are required for the **eauth** executable depend on how you implement external authentication at your site. For **eauth** parsing, unused variables are marked by empty quotation marks (").

User credentials

When an LSF user submits a job or issues a command, the LSF daemon that receives the request verifies the identity of the user by checking the user credentials. External authentication provides the greatest security of all LSF authentication methods because the user credentials are obtained from an external source, such as a database, and then encrypted prior to transmission. For Windows hosts, external authentication is the only truly secure type of LSF authentication.

Host credentials

LSF first authenticates users and then checks host credentials. LSF accepts requests that are sent from all hosts that are configured as part of the LSF cluster, including floating clients and any hosts that are dynamically added to the cluster. LSF rejects requests sent from a host that is not running LSF. If your cluster requires extra host authentication, you can write an **eauth** executable that verifies both user and host credentials.

Daemon credentials

Daemon authentication provides a secure channel for passing credentials between hosts, mediated by the management host. The management host mediates authentication by using the **eauth** executable, which ensures secure passing of credentials between submission hosts and execution hosts, even though the submission host does not know which execution host is selected to run a job.

Daemon authentication applies to the following communications between LSF daemons:

- **mbatchd** requests to **sbatchd**
- **sbatchd** updates to **mbatchd**
- PAM interactions with **res**
- **mbatchd** to **mbatchd** (for the LSF multicluster capability)

Scope

Applicability	Details
Operating system	UNIX

Applicability	Details
Allows	<ul style="list-style-type: none"> Authentication of LSF users, hosts, and daemons Authentication of any number of LSF users
Not required for	Authorization of users based on account permissions
Dependencies	<ul style="list-style-type: none"> UNIX user accounts must be valid on all hosts in the cluster, or the correct type of account mapping must be enabled: <ul style="list-style-type: none"> For a cluster with a non-uniform user name space, between-host account mapping must be enabled. You must enable cross-cluster user account mapping if you use the LSF multicluster capability with a non-uniform user name space. User accounts must have the correct permissions to successfully run jobs. The owner of <code>lsf.sudoers</code> on Windows must be <code>Administrators</code>.

Configuration to enable external authentication

During LSF installation, the parameter `LSF_AUTH` in the `lsf.conf` file is set to `eauth`, which enables external authentication. A default **eauth** executable file is installed in the directory that is specified by the parameter `LSF_SERVERDIR` in the `lsf.conf` file.

The default executable provides an example of how the `eauth` protocol works. You can write your own `eauth` executable to meet the security requirements of your cluster.

Configuration file	Parameter and syntax	Default behavior
lsf.conf	<code>LSF_AUTH=eauth</code>	<ul style="list-style-type: none"> Enables external authentication
	<code>LSF_AUTH_DAEMONS=1</code>	<ul style="list-style-type: none"> Enables daemon authentication when external authentication is enabled. Note: By default, daemon authentication is not enabled. If you enable daemon authentication and want to turn it off later, you must comment out or delete the parameter <code>LSF_AUTH_DAEMONS</code>.
lsf.sudoers	<code>LSF_EAUTH_USER=root</code>	<ul style="list-style-type: none"> Specifies that the eauth binary file is run as the root account.

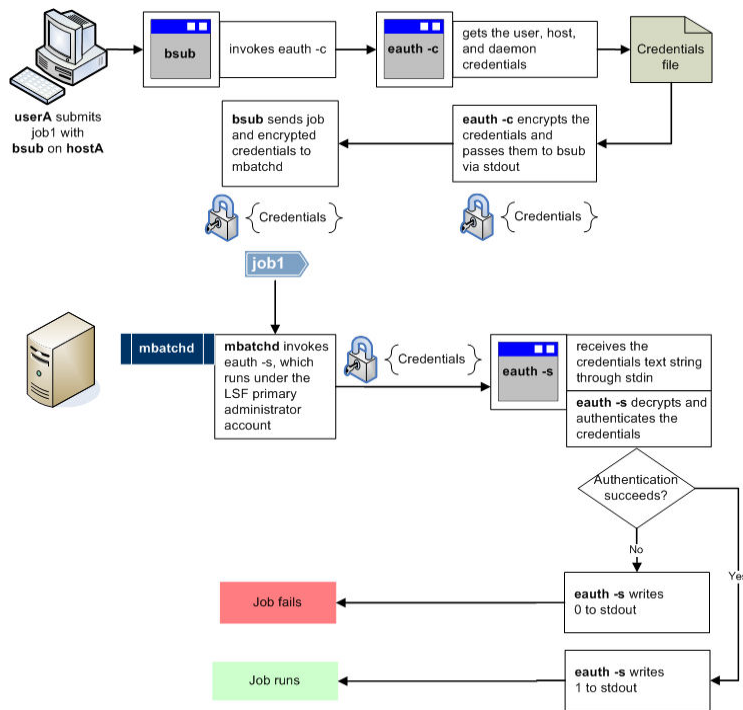
Note:

- Before enabling query command authentication in the LSF cluster, you must ensure that all hosts in the cluster (including management, server, and client hosts), including all LSF command executable files, are updated to LSF, Version 10.1 Fix Pack 11, or newer. If you have any commands that are built using the APIs from previous versions of LSF, you must also rebuild these commands with the APIs from this latest version of LSF.
- If you are enabling query command authentication for the LSF multicluster capability, ensure that all LSF clusters use the same `LSF_EAUTH_KEY` value in the `lsf.sudoers` file.

External authentication behavior

The following example illustrates how a customized `eauth` executable can provide external authentication of users, hosts, and daemons. In this example, the `eauth` executable has been customized so that corresponding instances of `eauth -c` and `eauth -s` obtain user, host, and daemon credentials from a file that serves as the external security system. The `eauth` executable can also be customized to obtain credentials from an operating system or from an authentication protocol such as Kerberos.

Figure 1. Example of external authentication



Authentication failure

When external authentication is enabled, the message

```
User permission denied
```

indicates that the eauth executable failed to authenticate the user's credentials.

Security

External authentication—and any other LSF authentication method—depends on the security of the root account on all hosts within the cluster. Limit access to the root account to prevent unauthorized use of your cluster.

Configuration to modify external authentication

You can modify external authentication behavior by writing your own eauth executable and by modifying configuration parameters.

The configuration parameters modify various aspects of external authentication behavior by:

- Increasing security by using an external encryption key (recommended)
- Specifying a trusted user account under which the eauth executable runs (UNIX and Linux only)

Note: To use the `lsf.sudoers` file, you must enable the setuid bit for the LSF administration commands. Run the **hostsetup --setuid** command option on the LSF management and candidate hosts. Since this allows LSF administration commands to run with root privileges, do not enable the setuid bit if you do not want these LSF commands to run with root privileges.

The **hostsetup --setuid** command enables the setuid bit for the following LSF executable files: **badadmin**, **lsadmin**, **egosh**, **utmpreg**, **swtbl_api**, **ntbl_api**, **lstbl_nid**, and **swtbl_poe**.

Configuration to modify security

File	Parameter and syntax	Descriptions
------	----------------------	--------------

File	Parameter and syntax	Descriptions
lsf.sudoers	LSF_EAUTH_KEY= <i>key</i>	<ul style="list-style-type: none"> The eauth executable file uses the external encryption key that you define to encrypt and decrypt the credentials. The key must contain at least 6 characters and must use only printable characters. For UNIX, you must edit the lsf.sudoers file on all hosts within the cluster and specify the same encryption key. You must also configure eauth as setuid to root so that eauth can read the lsf.sudoers file and obtain the value of LSF_EAUTH_KEY. For Windows, you must edit the shared lsf.sudoers file.
	LSF_EAUTH_OLDKEY= <i>key</i>	<ul style="list-style-type: none"> Specifies the previous key that eauth used to encrypt and decrypt user authentication data after you specify a new eauth key. Defining this parameter gives LSF administrators time to update the eauth key on each host in the cluster without disrupting authentication operations. All rules that apply to LSF_EAUTH_KEY also apply to LSF_EAUTH_OLDKEY. To use this parameter, you must also define LSF_EAUTH_OLDKEY_EXPIRY.
	LSF_EAUTH_OLDKEY_EXPIRY=[[<i>year</i> :] [<i>month</i> :] <i>day</i> :] <i>hour</i> : <i>minute</i>	<ul style="list-style-type: none"> Specifies the expiry date and time for the previous eauth key (LSF_EAUTH_OLDKEY), after which the previous key no longer works and only LSF_EAUTH_KEY works.

Configuration to specify the eauth user account

On UNIX hosts, the **eauth** executable runs under the account of the primary LSF administrator. You can modify this behavior by specifying a different trusted user account. For Windows hosts, you do not need to modify the default behavior because **eauth** runs under the service account, which is always a trusted, secure account.

File	Parameter and syntax	Description
lsf.sudoers	LSF_EAUTH_USER= <i>user_name</i>	<ul style="list-style-type: none"> UNIX only The eauth executable runs under the account of the specified user rather than the account of the LSF primary administrator. You must edit the lsf.sudoers file on all hosts within the cluster and specify the same user name.

External authentication commands

Commands for submission

Command	Description
All LSF commands	<ul style="list-style-type: none"> If the parameter LSF_AUTH=eauth in the file lsf.conf, LSF daemons authenticate users and hosts—as configured in the eauth executable—before executing an LSF command If external authentication is enabled and the parameter LSF_AUTH_DAEMONS=1 in the file lsf.conf, LSF daemons authenticate each other as configured in the eauth executable

Commands to monitor

Not applicable: There are no commands to monitor the behavior of this feature.

Commands to control

Not applicable: There are no commands to control the behavior of this feature.

Commands to display configuration

Command	Description
badmin showconf	<ul style="list-style-type: none">Displays all configured parameters and their values set in <code>lsf.conf</code> or <code>ego.conf</code> that affect mbatchd and sbatchd.Use a text editor to view other parameters in the <code>lsf.conf</code> or <code>ego.conf</code> configuration files.In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Use a text editor to view the `lsf.sudoers` configuration file.

Kerberos authentication

Kerberos authentication is an extension of external daemon authentication, providing authentication of LSF users and daemons during client-server interactions.

The **eauth.krb** executable is provided in the installation package under `$LSF_SERVERDIR` and uses Kerberos Version 5 APIs for interactions between the **mbatchd** and **sbatchd** daemons, and between the **pam** and **res** commands. When you use Kerberos authentication for a cluster or for the LSF multicluster capability, authentication data is encrypted along the entire path from job submission through to job completion.

You can also use Kerberos authentication for delegation of rights (forwarding credentials) when a job requires a Kerberos ticket during job execution. LSF ensures that a ticket-granting ticket (TGT) can be forwarded securely to the execution host. LSF also automatically renews Kerberos credentials.

- [Kerberos authentication with IBM Spectrum LSF](#)
The Kerberos integration for LSF allows you to use Kerberos authentication for LSF clusters and jobs.
- [Configuration to enable Kerberos authentication](#)
To enable Kerberos authentication, replace the default **eauth** executable file with the `eauth.krb5` file, then configure LSF and the Kerberos server to enable Kerberos authentication
- [Configuration to modify Kerberos authentication](#)
Modify Kerberos authentication to provide a secure data exchange during LSF user and daemon authentication and to forward credentials to a remote host for use during job execution.
- [Kerberos authentication commands](#)

Kerberos authentication with IBM® Spectrum LSF

The Kerberos integration for LSF allows you to use Kerberos authentication for LSF clusters and jobs.

Kerberos integration for LSF includes the following features:

- The dedicated binary **krbrenewd** renews TGTs for pending jobs and running jobs. It is enhanced to handle several jobs without creating too much overhead for **mbatchd** and KDC.
- Separate user TGT forwarding from daemon and user authentication with a parameter, `LSB_KRB_TGT_FWD`, to control TGT forwarding.
- Kerberos solution package is preinstalled in the LSF installation directory, relieving users from compiling from source code. `krb5` function calls are dynamically linked.
- Preliminary TGT forwarding support for parallel jobs, including shared directory support for parallel jobs. If all hosts at a customer site have a shared directory, you can configure this directory in `lsf.conf` via parameter `LSB_KRB_TGT_DIR`, and

the TGT for each individual job is stored here.

- LSF Kerberos integration works in an NFSv4 environment.

Install LSF in a location that does not require a credential to access.

You must provide the following krb5 libraries since they do not ship with LSF:

- libkrb5.so
- libkrb5support.so
- libk5crypto.so
- libcom_err.so

Set `LSB_KRB_LIB_PATH` in `lsf.conf` to the path that contains these four libraries.

Note the following issues when using the Kerberos integration:

- If you turn on the account mapping feature of LSF, you must ensure that the execution user has read/write permission for the directory that is defined by the `LSB_KRB_TGT_DIR` parameter, which holds the runtime TGT.
- krb5 libraries are required for TGT manipulation.
- Configure the TGT renew limit so it is long enough for jobs to finish running. Long jobs that last several hours or even several days need their TGTs renewed in time to keep the job running. Ensure that the job execution time does not exceed the TGT renew limit.
- With the **blaunch** command, only one task **res** is invoked per host.
- **blaunch** krb5 does not support auto-resizable jobs.
- **blaunch** krb5 does not support remote execution servers that are running LSF, Versions 9.1.2, or older, and therefore the renew script does not work with these versions of RES. Similarly, **blaunch** krb5 does not support sbatchd daemons from LSF, Versions 9.1.2, or older. Therefore, child sbatchd daemons cannot be kerberized and the renew script does not work in root sbatchd daemons from LSF, Versions 9.1.2, or older.
- The **brequeue** command does not transfer new TGTs to the **mbatchd** daemon. If a job is requeued by the **brequeue** command, the TGT job that is used is the one that is cached by the **mbatchd** daemon.
- LSF does not check the contents or exit code of the `erenew` script. If `erenew` contains the wrong command, AFS tokens cannot be renewed and LSF does not report any errors in the log file. Therefore, users must ensure that the commands in the `erenew` script can renew AFS tokens successfully.
- Some **bsub** options, such as **bsub -Zs** or **bsub -is** require the **bsub** command to do file manipulation. In this case, if the file involved resides in the AFS volume, users must ensure that they acquire a proper AFS token before they run the **bsub** command.

Kerberos Support for NFSv4 and AFS

When LSF is used on NFSv4 or Andrew File System (AFS), each process in a sequential job or a distributed parallel job needs to periodically renew its credentials. For this reauthentication to take place in a secure, user friendly environment, a TGT file is distributed to each execution host and the root sbatchd daemon in each execution host renews the TGT.

If you use the AFS feature, you must provide the `libkopenafs.so` or `libkopenafs.so.1` libraries, which do not ship with LSF. You can use them from the `openafs-authlibs-*` package or build them directly from the AFS source.

To support AFS, LSF provides an external renew hook mechanism, which is called after TGT is renewed. Users can write their own renew logic through this renew hook. More specifically, users can use the demo script `erenew.krb5` in the `$LSF_SERVERDIR` directory and rename it to `erenew`. Users can also create the `erenew` executable file in the `$LSF_SERVERDIR` directory. This `erenew` script is called immediately at job startup time to make sure the user's job has a valid AFS token. LSF also automatically calls the binary file after TGT is renewed. For example, AFS users can use this hook to run the **aklog** command for renewing AFS tokens.

Configuration to enable Kerberos authentication

To enable Kerberos authentication, replace the default **eauth** executable file with the `eauth.krb5` file, then configure LSF and the Kerberos server to enable Kerberos authentication

The following are the independent features you can configure with Kerberos:

- TGT forwarding
- User **eauth** with krb5
- Inter-daemon authentication with krb5

TGT forwarding is the most commonly used. All of these features need to dynamically load krb5 libs, which can be set by the `LSB_KRB_LIB_PATH` parameter. This parameter is optional. It tells LSF where krb5 is installed. If not set, it defaults to `/usr/local/lib`.

To enable TGT forwarding:

1. Register the user principal in the KDC server (if not already done). Set **LSB_KRB_TGT_FWD=Y|y** in the `lsf.conf` file. This parameter serves as an overall switch that turns TGT forwarding on or off.
2. Set the `LSB_KRB_CHECK_INTERVAL` parameter in the `lsf.conf` file. This parameter is optional. The parameter controls the time interval for TGT checking. If it is not set, the default value of 15 minutes is used.
3. Set the `LSB_KRB_RENEW_MARGIN` parameter in the `lsf.conf` file. This parameter is optional. The parameter controls how much elapses before TGT is renewed. If it is not set, the default value of 1 hour is used.
4. Set the `LSB_KRB_TGT_DIR` parameter in the `lsf.conf` file. This parameter is optional. It specifies where to store TGT on the execution host. If not set, it defaults to `/tmp` on the execution host.
5. Restart LSF.
6. Run **kinit -r [sometime] -f** to obtain a user TGT for forwarding.
7. Submit jobs as normal.

To enable user eauth with krb5:

1. Replace the `eauth` binary file in the `$LSF_SERVERDIR` directory with the `eauth.krb5` file, which resides in the same directory.
2. Set `LSF_AUTH=eauth` in the `lsf.conf` file. The value `eauth` is the default setting.
3. Set `LSB_EAUTH_DATA_REUSE=N` in the `lsf.conf` file. This is required for the **blaunch** command to work.
4. Set `LSF_EAUTH_USER=root` in the `/etc/lsf.sudoers` file for all management and management candidate hosts.
5. Optional. Enable the Kerberos user impersonation feature.
 - a. Set `LSF_KRB_IMPERSONATE=Y` in the `lsf.conf` file.
 - b. Restart LSF.
 - c. Run `kinit -r user_name` to obtain a user TGT for the submission user to impersonate.

Note: For security purposes, using root is not supported for Kerberos. Do not use root and do not add the root user to Kerberos.

If Kerberos user impersonation is enabled, the following LSF commands work differently:

- If the token user is not the OS submission user, commands that depend on OS file permissions (such as **bpeek** and **brestart**) do not work properly.
- The ACL feature for the **bacct** and **bhist** commands is disabled to prevent other users from getting the LSF administrator token. To ensure that the commands remain secure, do not enable the `setuid` bit for the **bacct** and **bhist** executable files, and disable them if they are already set.
- The **lsrun** command might behave inconsistently between running on local and remote hosts, because when an **lsrun** task is run on the local host, it does not go through **eauth** authorization.

To enable inter-daemon authentication with krb5:

1. Replace the `eauth` binary file in the `$LSF_SERVERDIR` directory with the `eauth.krb5` file, which resides in the same directory.
2. Set `LSF_AUTH=eauth` in the `lsf.conf` file. The value `eauth` is the default setting.
3. Set `LSF_AUTH_DAEMONS=1` in the `lsf.conf` file.

Adding Kerberos principals for LSF for user and daemon authentication

The first step is to configure the Kerberos server. The following procedure sets up a Kerberos principal and key table entry items that are used by the LSF **mbatchd** daemon to communicate with user commands and other daemons:

1. Create a Kerberos "management LSF principal" using the **kadmin** command's add principal subcommand (**addprinc**). The principal's name is `lsf/cluster_name@realm_name`. In this example, you add a management LSF principal to `cluster1`:
 - a. Run **kadmin: addprinc lsf/cluster1**
 - b. Enter a password for the principal `lsf/cluster1@COMPANY.COM`: `<enter password here>`

- c. Reenter the password for the principal `lsf/cluster1@COMPANY.COM:<re-type password>`The principal `lsf/cluster1@COMPANY.COM` is created.
 - d. Run the **ktadd** subcommand of **kadmin** on all management hosts to add a key for the **mbatchd** daemon to the local host keytab file:
kadmin: ktadd -k /etc/krb5.keytab lsf/cluster_name
2. After you create the management LSF principal, you must set up a principal for each LSF server host. Create a host principal for LSF with the **kadmin** command's add principal subcommand (**addprinc**). The principal's name is `lsf/host_name@realm_name`. In this example, you add a host principal for HostA:
- a. Run **kadmin: addprinc lsf/hostA.company.com**
 - b. Enter a password for the principal `lsf/hostA.company.com@COMPANY.COM:<enter password here>`
 - c. Reenter the password for the principal `lsf/hostA.company.com@COMPANY.COM:<re-type password>`
 - d. Run **kadmin** and use **ktadd** to add this key to the local keytab on each host. You must run **kadmin** as root. In this example, you create a local key table entry for HostA:
kadmin: ktadd -k /etc/krb5.keytab lsf/hostA.company.com

Configuring LSF to work in an AFS or NFSv4 environment

To configure LSF to work in an AFS or NFSv4 environment (for example, to give LSF and the user's job access to an AFS filesystem):

1. Set `LSB_KRB_TGT_FWD=Y` in `lsf.conf`.
2. Set `LSB_AFS_JOB_SUPPORT=Y` in `lsf.conf`.
3. Set `LSB_AFS_BIN_DIR= path to aklog command`. If not set, the system searches in `/bin`, `/usr/bin`, `/usr/local/bin`, `/usr/afs/bin`.
4. Rename the `$LSF_SERVERDIR/erenew.krb5` file to `$LSF_SERVERDIR/erenew` or write an executable file that is named **erenew** in the `$LSF_SERVERDIR` directory with at least the following content:

```
#!/bin/sh
/path/to/aklog/command/aklog
```

5. Submit the job. For example, a user might submit a parallel job to run on two hosts:
`bsub -m "host1 host2" -n 2 -R "span[ptile=1]" blaunch <user job commands...>`

End users can use the system normally as long as they have a Kerberos credential before they submit a job.

Generally, the `erenew` interface functions as follows: If `LSB_KRB_TGT_FWD=Y` in the `lsf.conf` file and there is an executable file that is named `erenew` in the `$LSF_SERVERDIR` directory, then LSF runs this executable:

- Once per host per job on dispatch.
- Once per host per job immediately after the Kerberos TGT is renewed.

If the system is configured for AFS, the user's tasks run in the same Process Authentication Group (PAG) in which this executable is run on each host. Users ensure their `renew` script does not create new PAG because every task process is automatically put into an individual PAG. PAG is the group with which AFS associates security tokens.

When the parameter `LSB_AFS_JOB_SUPPORT` in `lsf.conf` is set to `Y|y`:

1. LSF assumes the user's job is running in an AFS environment, and calls **aklog -setpag** to create a new PAG for the user's job if it is a sequential job, or to create a separate PAG for each task **res** if the job is a **blaunch** job.
2. LSF runs the `erenew` script after the TGT is renewed. This script is primarily used to run **aklog**.
3. LSF assumes that `JOB_SPOOL_DIR` resides in the AFS volume. It kerberizes the child **sbatchd** to get the AFS token so the child **sbatchd** can access `JOB_SPOOL_DIR`.

A typical use case for an end user is to set `LSB_AFS_JOB_SUPPORT=Y` in `lsf.conf` and call the **aklog** command in the `erenew` script. The user must not initiate a new PAG in the `erenew` script (such as by calling **aklog -setpag**) in this case. If this parameter is changed, you must restart root **res** to make the change take effect.

If `LSB_AFS_JOB_SUPPORT=Y`, then LSF needs **aklog** in AFS to create a new PAG. You can then use the `LSB_AFS_BIN_DIR` parameter in `lsf.conf` to tell LSF the file path and directory where **aklog** resides.

If `LSB_AFS_BIN_DIR` is not defined, LSF searches in the following order: `/bin`, `/usr/bin`, `/usr/local/bin`, `/usr/afs/bin`. The search stops as soon as an executable **aklog** is found.

To turn off this TGT renewal process where the TGT file is distributed to each execution host, and instead have the TGT reside on a shared file system where each process can read it, define a directory for `LSB_KRB_TGT_DIR` in the `lsf.conf` file.

Configuration to modify Kerberos authentication

Modify Kerberos authentication to provide a secure data exchange during LSF user and daemon authentication and to forward credentials to a remote host for use during job execution.

Kerberos authentication is supported only for UNIX and Linux hosts, and only on the following operating systems:

- IRIX 6.5
- Linux 2.x
- Solaris (all versions)

Note: To use the `lsf.sudoers` file, you must enable the `setuid` bit for the LSF administration commands. Run the **hostsetup --setuid** command option on the LSF management and candidate hosts. Since this allows LSF administration commands to run with root privileges, do not enable the `setuid` bit if you do not want these LSF commands to run with root privileges.

The **hostsetup --setuid** command enables the `setuid` bit for the following LSF executable files: **badadmin**, **lsadmin**, **egosh**, **utmpreg**, **swtbl_api**, **ntbl_api**, **lstbl_nid**, and **swtbl_poe**.

Configuration file	Parameter and syntax	Behavior
lsf.conf	LSF_AUTH=eauth	<ul style="list-style-type: none"> • Enables external authentication
	LSF_AUTH_DAE MONS=1	<ul style="list-style-type: none"> • Enables daemon authentication when external authentication is enabled.
	LSB_KRB_TGT_FWD=Y y N n	<ul style="list-style-type: none"> • Controls the user Ticket Granting Ticket (TGT) forwarding feature
	LSB_KRB_TGT_DIR=dir	<ul style="list-style-type: none"> • Specifies a directory in which Ticket Granting Ticket (TGT) for a running job is stored.
	LSB_KRB_CHECK_INTERVAL=minutes	<ul style="list-style-type: none"> • Sets a time interval for how long the krbrenewd and root sbatchd daemons can wait before the next check.
	LSB_KRB_RENEW_MARGIN=minutes	<ul style="list-style-type: none"> • Specifies how long krbrenewd and root sbatchd daemons have to renew Ticket Granting Ticket (TGT) before it expires.
	LSB_KRB_LIB_PATH=path to krb5 lib	<ul style="list-style-type: none"> • Specifies the file path that contains krb5 libraries.
	LSB_EAUTH_EACH_SUBPACK=Y y N n	<ul style="list-style-type: none"> • Makes the bsub command call eauth for each subpack.
	LSB_KRB_IMPERSONATE=Y y N n	<ul style="list-style-type: none"> • Enables user impersonation with krb5 • All commands that call the eauth.krb5 external authentication are run as the TGT user
	LSB_EAUTH_DATA_REUSE=Y y N n	<ul style="list-style-type: none"> • Y enables the blaunch command to cache authentication data when connecting to the first remote execution server in memory. blaunch uses this cached data to authenticate subsequent first remote execution servers. • N means that the blaunch command does not cache authentication. Every time blaunch connects to a different server, it fetches new authentication data. • You must set this to N if you are enabling user eauth with krb5.

Configuration file	Parameter and syntax	Behavior
lsf.sudoers	LSF_EAUTH_US ER=root	<ul style="list-style-type: none"> For Kerberos authentication, the eauth executable file must run under the root account. You must edit the lsf.sudoers file on all hosts within the cluster and specify the same user name. The Kerberos specific eauth is only used for user authentication or daemon authentication.

Kerberos authentication commands

Commands for submission

Command	Description
All LSF commands	<ul style="list-style-type: none"> If the parameter LSF_AUTH=eauth in the file lsf.conf, LSF daemons authenticate users and hosts, as configured in the eauth executable, before they execute an LSF command. If external authentication is enabled and the parameter LSF_AUTH_DAEMONS=1 in the file lsf.conf, LSF daemons authenticate each other as configured in the eauth executable file.

Commands to monitor

Not applicable: There are no commands to monitor the behavior of this feature.

Commands to control

Not applicable: There are no commands to control the behavior of this feature.

Commands to display configuration

Command	Description
badmin showconf	<ul style="list-style-type: none"> Displays all configured parameters and their values set in lsf.conf or ego.conf that affect mbatchd and sbatchd. <p>Use a text editor to view other parameters in the lsf.conf or ego.conf configuration files.</p> <ul style="list-style-type: none"> In a MultiCluster environment, displays the parameters of daemons on the local cluster.

Use a text editor to view the lsf.sudoers configuration file.

Handle job exceptions

You can configure hosts and queues so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected and their corresponding actions. By default, LSF does not detect any exceptions.

Run **bjobs -d -m host_name** to see exited jobs for a particular host.

Job exceptions LSF can detect

If you configure job exception handling in your queues, LSF detects the following job exceptions:

- Job underrun - jobs end too soon (run time is less than expected). Underrun jobs are detected when a job exits abnormally
- Job overrun - job runs too long (run time is longer than expected). By default, LSF checks for overrun jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for job overrun.
- Job estimated run time exceeded—the job’s actual run time has exceeded the estimated run time.
- Idle job - running job consumes less CPU time than expected (in terms of CPU time/runtime). By default, LSF checks for idle jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for idle jobs.

Host exceptions LSF can detect

If you configure host exception handling, LSF can detect jobs that exit repeatedly on a host. The host can still be available to accept jobs, but some other problem prevents the jobs from running. Typically jobs dispatched to such “black hole”, or “job-eating” hosts exit abnormally. By default, LSF monitors the job exit rate for hosts, and closes the host if the rate exceeds a threshold you configure (`EXIT_RATE` in `lsb.hosts`).

If `EXIT_RATE` is not specified for the host, LSF invokes **eadmin** if the job exit rate for a host remains above the configured threshold for longer than 5 minutes. Use `JOB_EXIT_RATE_DURATION` in `lsb.params` to change how frequently LSF checks the job exit rate.

Use `GLOBAL_EXIT_RATE` in `lsb.params` to set a cluster-wide threshold in minutes for exited jobs. If `EXIT_RATE` is not specified for the host in `lsb.hosts`, `GLOBAL_EXIT_RATE` defines a default exit rate for all hosts in the cluster. Host-level `EXIT_RATE` overrides the `GLOBAL_EXIT_RATE` value.

Customize job exception actions with the eadmin script

When an exception is detected, LSF takes appropriate action by running the script `LSF_SERVERDIR/eadmin` on the management host.

You can customize **eadmin** to suit the requirements of your site. For example, **eadmin** could find out the owner of the problem jobs and use **bstop -u** to stop all jobs that belong to the user.

In some environments, a job running 1 hour would be an overrun job, while this may be a normal job in other environments. If your configuration considers jobs running longer than 1 hour to be overrun jobs, you may want to close the queue when LSF detects a job that has run longer than 1 hour and invokes **eadmin**.

- [Email job exception details](#)
- [Default eadmin actions](#)
- [Handle job initialization failures](#)
- [Handle host-level job exceptions](#)
- [Handle job exceptions in queues](#)
- [Understand successful application exit values](#)

Email job exception details

About this task

Set LSF to send you an email about job exceptions that includes details including `JOB_ID`, `RUN_TIME`, `IDLE_FACTOR` (if job has been idle), `USER`, `QUEUE`, `EXEC_HOST`, and `JOB_NAME`.

Procedure

1. In `lsb.params`, set `EXTEND_JOB_EXCEPTION_NOTIFY=Y`.
2. Set the format option in the **eadmin** script (`LSF_SERVERDIR/eadmin` on the management host).
 - a. Uncomment the `JOB_EXCEPTION_EMAIL_FORMAT` line and add a value for the format:
 - `JOB_EXCEPTION_EMAIL_FORMAT=fixed`: The `eadmin` shell generates an exception email with a fixed length for the job exception information. For any given field, the characters truncate when the maximum is reached (between 10-19).
 - `JOB_EXCEPTION_EMAIL_FORMAT=full`: The `eadmin` shell generates an exception email without a fixed length for the job exception information.

Default eadmin actions

For host-level exceptions, LSF closes the host and sends email to the LSF administrator. The email contains the host name, job exit rate for the host, and other host information. The message `eadmin: JOB EXIT THRESHOLD EXCEEDED` is attached to the closed host event in `lsb.events`, and displayed by **badmin hist** and **badmin hhist**.

For job exceptions, LSF sends email to the LSF administrator. The email contains the job ID, exception type (overrun, underrun, idle job), and other job information.

An email is sent for all detected job exceptions according to the frequency configured by `EADMIN_TRIGGER_DURATION` in `lsb.params`. For example, if `EADMIN_TRIGGER_DURATION` is set to 5 minutes, and 1 overrun job and 2 idle jobs are detected, after 5 minutes, **eadmin** is invoked and only one email is sent. If another overrun job is detected in the next 5 minutes, another email is sent.

Handle job initialization failures

By default, LSF handles job exceptions for jobs that exit after they have started running. You can also configure LSF to handle jobs that exit during initialization because of an execution environment problem, or because of a user action or LSF policy.

LSF detects that the jobs are exiting before they actually start running, and takes appropriate action when the job exit rate exceeds the threshold for specific hosts (`EXIT_RATE` in `lsb.hosts`) or for all hosts (`GLOBAL_EXIT_RATE` in `lsb.params`).

Use `EXIT_RATE_TYPE` in `lsb.params` to include job initialization failures in the exit rate calculation. The following table summarizes the exit rate types that you can configure:

Table 1. Exit rate types you can configure

Exit rate type ...	Includes ...
JOBEXIT	Local exited jobs Remote job initialization failures Parallel job initialization failures on hosts other than the first execution host Jobs exited by user action (e.g., <code>bkill</code> , <code>bstop</code> , etc.) or LSF policy (e.g., load threshold exceeded, job control action, advance reservation expired, etc.)
JOBEXIT_NONLSF This is the default when <code>EXIT_RATE_TYPE</code> is not set	Local exited jobs Remote job initialization failures Parallel job initialization failures on hosts other than the first execution host
JOBINIT	Local job initialization failures Parallel job initialization failures on the first execution host
HPCINIT	Job initialization failures for HPC jobs

Job exits excluded from exit rate calculation

By default, jobs that are exited for non-host related reasons (user actions and LSF policies) are not counted in the exit rate calculation. Only jobs that are exited for what LSF considers host-related problems and are used to calculate a host exit rate.

The following cases are *not included* in the exit rate calculations:

- **bkill, bkill -r**
- **bqueue**
- RERUNNABLE jobs killed when a host is unavailable
- Resource usage limit exceeded (for example, PROCESSLIMIT, CPULIMIT, etc.)
- Queue-level job control action TERMINATE and TERMINATE_WHEN
- Checkpointing a job with the kill option (**bchkpnt -k**)
- Rerunnable job migration
- Job killed when an advance reservation has expired
- Remote lease job start fails
- Any jobs with an exit code found in SUCCESS_EXIT_VALUES, where a particular exit value is deemed as successful.

Exclude LSF and user-related job exits

To explicitly *exclude* jobs exited because of user actions or LSF-related policies from the job exit calculation, set `EXIT_RATE_TYPE = JOBEXIT_NONLSF` in `lsb.params`. `JOBEXIT_NONLSF` tells LSF to include all job exits *except* those that are related to user action or LSF policy. This is the default value for `EXIT_RATE_TYPE`.

To *include* all job exit cases in the exit rate count, you must set `EXIT_RATE_TYPE = JOBEXIT` in `lsb.params`. `JOBEXIT` considers all job exits.

Jobs killed by signal external to LSF will still be counted towards exit rate

Jobs killed because of job control `SUSPEND` action and `RESUME` action are still counted towards the exit rate. This because LSF cannot distinguish between jobs killed from `SUSPEND` action and jobs killed by external signals.

If both `JOBEXIT` and `JOBEXIT_NONLSF` are defined, `JOBEXIT_NONLSF` is used.

Local jobs

When `EXIT_RATE_TYPE=JOBINIT`, various job initialization failures are included in the exit rate calculation, including:

- Host-related failures; for example, incorrect user account, user permissions, incorrect directories for checkpointable jobs, host name resolution failed, or other execution environment problems
- Job-related failures; for example, pre-execution or setup problem, job file not created, etc.

Parallel jobs

By default, or when `EXIT_RATE_TYPE=JOBEXIT_NONLSF`, job initialization failure on the first execution host does not count in the job exit rate calculation. Job initialization failure for hosts other than the first execution host are counted in the exit rate calculation.

When `EXIT_RATE_TYPE=JOBINIT`, job initialization failure happens on the first execution host are counted in the job exit rate calculation. Job initialization failures for hosts other than the first execution host are *not* counted in the exit rate calculation.

Tip:

For parallel job exit exceptions to be counted for *all* hosts, specify `EXIT_RATE_TYPE=HPCINIT` or `EXIT_RATE_TYPE=JOBEXIT_NONLSF JOBINIT`.

Remote jobs

By default, or when `EXIT_RATE_TYPE=JOBEXIT_NONLSF`, job initialization failures are counted as exited jobs on the remote execution host and are included in the exit rate calculation for that host. To include only *local* job initialization failures on the execution cluster from the exit rate calculation, set `EXIT_RATE_TYPE` to include only `JOBINIT` or `HPCINIT`.

Scale and tune job exit rate by number of slots

On large, multiprocessor hosts, use `ENABLE_EXIT_RATE_PER_SLOT=Y` in `lsb.params` to scale the job exit rate so that the host is only closed when the job exit rate is high enough in proportion to the number of processors on the host. This avoids having a relatively low exit rate close a host inappropriately.

Use a float value for `GLOBAL_EXIT_RATE` in `lsb.params` to tune the exit rate on multislot hosts. The actual calculated exit rate value is never less than 1.

Example: exit rate of 5 on single processor and multiprocessor hosts

On a single-processor host, a job exit rate of 5 is much more severe than on a 20-processor host. If a stream of jobs to a single-processor host is consistently failing, it is reasonable to close the host or take some other action after five failures.

On the other hand, for the same stream of jobs on a 20-processor host, it is possible that 19 of the processors are busy doing other work that is running fine. To close this host after only 5 failures would be wrong because effectively less than 5% of the jobs on that host are actually failing.

Example: float value for GLOBAL_EXIT_RATE on multislot hosts

Using a float value for `GLOBAL_EXIT_RATE` allows the exit rate to be less than the number of slots on the host. For example, on a host with four slots, `GLOBAL_EXIT_RATE=0.25` gives an exit rate of 1. The same value on an eight slot machine would be two, and so on. On a single-slot host, the value is never less than 1.

Handle host-level job exceptions

You can configure hosts so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected, and the corresponding actions. By default, LSF does not detect any exceptions.

Host exceptions LSF can detect

If you configure host exception handling, LSF can detect jobs that exit repeatedly on a host. The host can still be available to accept jobs, but some other problem prevents the jobs from running. Typically jobs that are dispatched to such “black hole”, or “job-eating” hosts exit abnormally. LSF monitors the job exit rate for hosts, and closes the host if the rate exceeds a threshold you configure (`EXIT_RATE` in `lsb.hosts`).

If `EXIT_RATE` is specified for the host, LSF invokes **eadmin** if the job exit rate for a host remains above the configured threshold for longer than 5 minutes. Use `JOB_EXIT_RATE_DURATION` in `lsb.params` to change how frequently LSF checks the job exit rate.

Use `GLOBAL_EXIT_RATE` in `lsb.params` to set a cluster-wide threshold in minutes for exited jobs. If `EXIT_RATE` is not specified for the host in `lsb.hosts`, `GLOBAL_EXIT_RATE` defines a default exit rate for all hosts in the cluster. Host-level `EXIT_RATE` overrides the `GLOBAL_EXIT_RATE` value.

Configure host exception handling (lsb.hosts)

EXIT_RATE

Specify a threshold for exited jobs. If the job exit rate is exceeded for 5 minutes or the period specified by `JOB_EXIT_RATE_DURATION` in `lsb.params`, LSF invokes **eadmin** to trigger a host exception.

Example

The following Host section defines a job exit rate of 20 jobs for all hosts, and an exit rate of 10 jobs on `hostA`.

```
Begin Host
HOST_NAME    MXJ      EXIT_RATE # Keywords
Default      !        20
hostA        !        10
End Host
```

Configure thresholds for host exception handling

By default, LSF checks the number of exited jobs every 5 minutes. Use `JOB_EXIT_RATE_DURATION` in `lsb.params` to change this default.

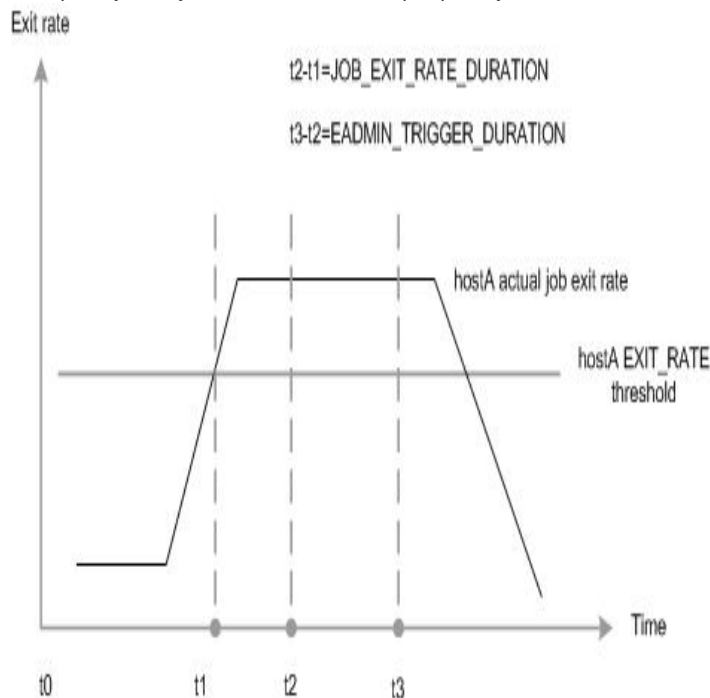
Tuning

Tip:

Tune `JOB_EXIT_RATE_DURATION` carefully. Shorter values may raise false alarms, longer values may not trigger exceptions frequently enough.

Example

In the following diagram, the job exit rate of `hostA` exceeds the configured threshold (`EXIT_RATE` for `hostA` in `lsb.hosts`). LSF monitors `hostA` from time `t1` to time `t2` ($t2 = t1 + \text{JOB_EXIT_RATE_DURATION}$ in `lsb.params`). At `t2`, the exit rate is still high, and a host exception is detected. At `t3` (`EADMIN_TRIGGER_DURATION` in `lsb.params`), LSF invokes **eadmin** and the host exception is handled. By default, LSF closes `hostA` and sends email to the LSF administrator. Since `hostA` is closed and cannot accept any new jobs, the exit rate drops quickly.



Handle job exceptions in queues

You can configure queues so that LSF detects exceptional conditions while jobs are running, and take appropriate action automatically. You can customize what exceptions are detected, and the corresponding actions. By default, LSF does not detect any exceptions.

Job exceptions LSF can detect

If you configure job exception handling in your queues, LSF detects the following job exceptions:

- Job underrun - jobs end too soon (run time is less than expected). Underrun jobs are detected when a job exits abnormally
- Job overrun - job runs too long (run time is longer than expected). By default, LSF checks for overrun jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for job overrun.
- Idle job - running job consumes less CPU time than expected (in terms of CPU time/runtime). By default, LSF checks for idle jobs every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for idle jobs.

Configure job exception handling (lsb.queues)

You can configure your queues to detect job exceptions. Use the following parameters:

JOB_IDLE

Specify a threshold for idle jobs. The value should be a number between 0.0 and 1.0 representing CPU time/runtime. If the job idle factor is less than the specified threshold, LSF invokes **eadmin** to trigger the action for a job idle exception.

JOB_OVERRUN

Specify a threshold for job overrun. If a job runs longer than the specified run time, LSF invokes **eadmin** to trigger the action for a job overrun exception.

JOB_UNDERRUN

Specify a threshold for job underrun. If a job exits before the specified number of minutes, LSF invokes **eadmin** to trigger the action for a job underrun exception.

Example

The following queue defines thresholds for all types job exceptions:

```
Begin Queue
...
JOB_UNDERRUN = 2
JOB_OVERRUN  = 5
JOB_IDLE     = 0.10
...
End Queue
```

For this queue:

- A job underrun exception is triggered for jobs running less than 2 minutes
- A job overrun exception is triggered for jobs running longer than 5 minutes
- A job idle exception is triggered for jobs with an idle factor (CPU time/runtime) less than 0.10

Configure thresholds for job exception handling

By default, LSF checks for job exceptions every 1 minute. Use `EADMIN_TRIGGER_DURATION` in `lsb.params` to change how frequently LSF checks for overrun, underrun, and idle jobs.

Tuning

Tip:

Tune `EADMIN_TRIGGER_DURATION` carefully. Shorter values may raise false alarms, longer values may not trigger exceptions frequently enough.

Understand successful application exit values

Jobs that exit with one of the exit codes specified by `SUCCESS_EXIT_VALUES` in an application profile are marked as DONE. These exit values are not counted in the `EXIT_RATE` calculation.

0 always indicates application success regardless of `SUCCESS_EXIT_VALUES`.

If both `SUCCESS_EXIT_VALUES` and `REQUEUE_EXIT_VALUES` are defined with the same exit code, `REQUEUE_EXIT_VALUES` will take precedence and the job will be set to PENDING state and requeued. For example:

bapp -l test

```
APPLICATION NAME: test
-- Turns on absolute runlimit for this application

STATISTICS:
      NJOBS      PEND      RUN      SSUSP      USUSP      RSV
        0         0         0         0         0         0
```

Both parameters `REQUEUE_EXIT_VALUES` and `SUCCESS_EXIT_VALUE` are set to 17.

bsub -app test ./non_zero.sh

Job <5583> is submitted to default queue <normal>

bhist -l 5583

```
Job <5583>, user <name>, Project <default>, Application <test>, Command <./non_zero.sh>
Fri Feb  1 10:52:20: Submitted from host <HostA>, to Queue <normal>, CWD <${HOME}>;
Fri Feb  1 10:52:22: Dispatched to <intel4>, Effective RES_REQ <select[type == local]
order[slots] >;
Fri Feb  1 10:52:22: Starting (Pid 31390);
Fri Feb  1 10:52:23: Running with execution home </home/dir>, Execution CWD </home/dir>,
Execution Pid <31390>;
Fri Feb  1 10:52:23: Pending: Requeued job is waiting for rescheduling;(exit code 17)
Fri Feb  1 10:52:23: Dispatched to <intel4>, Effective RES_REQ <select[type == local]
order[slots] >;
Fri Feb  1 10:52:23: Starting (Pid 31464);
Fri Feb  1 10:52:26: Running with execution home </home/dir>, Execution CWD </home/dir>,
Execution Pid <31464>;
Fri Feb  1 10:52:27: Pending: Requeued job is waiting for rescheduling;(exit code 17)
Fri Feb  1 10:52:27: Dispatched to <intel4>, Effective RES_REQ <select[type == local]
order[slots] >;
Fri Feb  1 10:52:27: Starting (Pid 31857);
Fri Feb  1 10:52:30: Running with execution home </home/dir>, Execution CWD </home/dir>,
Execution Pid <31857>;
Fri Feb  1 10:52:30: Pending: Requeued job is waiting for rescheduling;(exit code 17)
Fri Feb  1 10:52:31: Dispatched to <intel4>, Effective RES_REQ <select[type == local]
order[slots] >;
Fri Feb  1 10:52:31: Starting (Pid 32149);
Fri Feb  1 10:52:34: Running with execution home </home/dir>, Execution CWD </home/dir>,
Execution Pid <32149>;
Fri Feb  1 10:52:34: Pending: Requeued job is waiting for rescheduling;(exit code 17)
Fri Feb  1 10:52:34: Dispatched to <intel4>, Effective RES_REQ <select[type == local]
order[slots] >;
Fri Feb  1 10:52:34: Starting (Pid 32312);
Fri Feb  1 10:52:38: Running with exit code 17
```

`SUCCESS_EXIT_VALUES` has no effect on pre-exec and post-exec commands. The value is only used for user jobs.

If the job exit value falls into `SUCCESS_EXIT_VALUES`, the job will be marked as DONE. Job dependencies on done jobs behave normally.

For parallel jobs, the exit status refers to the job exit status and not the exit status of individual tasks.

Exit codes for jobs terminated by LSF are excluded from success exit value even if they are specified in `SUCCESS_EXIT_VALUES`.

For example, if `SUCCESS_EXIT_VALUES=2` is defined, jobs exiting with 2 are marked as DONE. However, if LSF cannot find the current working directory, LSF terminates the job with exit code 2, and the job is marked as EXIT. The appropriate termination reason is displayed by **bacct**.

MultiCluster jobs

In the job forwarding model, for jobs sent to a remote cluster, jobs exiting with success exit codes defined in the remote cluster are considered done successfully.

In the lease model, the parameters of `lsb.applications` apply to jobs running on remote leased hosts as if they are running on local hosts.

- [Specify successful application exit values](#)

Specify successful application exit values

About this task

Use `SUCCESS_EXIT_VALUES` to specify a list of exit codes that will be considered as successful execution for the application.

Procedure

1. Log in as the LSF administrator on any host in the cluster.
2. Edit the `lsb.applications` file.
3. Set `SUCCESS_EXIT_VALUES` to specify a list of job success exit codes for the application.

```
SUCCESS_EXIT_VALUES=230 222 12
```

4. Save the changes to `lsb.applications`.
5. Run **`admin reconfig`** to reconfigure `mbatchd`.

Tune CPU factors

CPU factors are used to differentiate the relative speed of different machines. LSF runs jobs on the best possible machines so that response time is minimized.

To achieve this, it is important that you define correct CPU factors for each machine model in your cluster.

How CPU factors affect performance

Incorrect CPU factors can reduce performance the following ways.

- If the CPU factor for a host is too low, that host might not be selected for job placement when a slower host is available. This means that jobs would not always run on the fastest available host.
- If the CPU factor is too high, jobs are run on the fast host even when they would finish sooner on a slower but lightly loaded host. This causes the faster host to be overused while the slower hosts are underused.

Both of these conditions are somewhat self-correcting. If the CPU factor for a host is too high, jobs are sent to that host until the CPU load threshold is reached. LSF then marks that host as busy, and no further jobs are sent there. If the CPU factor is too low, jobs might be sent to slower hosts. This increases the load on the slower hosts, making LSF more likely to schedule future jobs on the faster host.

Guidelines for setting CPU factors

CPU factors should be set based on a benchmark that reflects your workload. If there is no such benchmark, CPU factors can be set based on raw CPU power.

The CPU factor of the slowest hosts should be set to 1, and faster hosts should be proportional to the slowest.

Example

Consider a cluster with two hosts: `hostA` and `hostB`. In this cluster, `hostA` takes 30 seconds to run a benchmark and `hostB` takes 15 seconds to run the same test. The CPU factor for `hostA` should be 1, and the CPU factor of `hostB` should be 2 because it is twice as fast as `hostA`.

- [View normalized ratings](#)
- [Tune CPU factors](#)

View normalized ratings

Procedure

Run **lsload -N** to display normalized ratings.

LSF uses a normalized CPU performance rating to decide which host has the most available CPU power. Hosts in your cluster are displayed in order from best to worst. Normalized CPU run queue length values are based on an estimate of the time it would take each host to run one additional unit of work, given that an unloaded host with CPU factor 1 runs one unit of work in one unit of time.

Tune CPU factors

Procedure

1. Log in as the LSF administrator on any host in the cluster.
2. Edit `lsf.shared`, and change the HostModel section:

```
Begin HostModel
MODELNAME  CPUFACTOR  ARCHITECTURE # keyword
#HPUX (HPPA)
HP9K712S      2.5    (HP9000712_60)
HP9K712M      2.5    (HP9000712_80)
HP9K712F      4.0    (HP9000712_100)
```

See the *LSF Configuration Reference* for information about the `lsf.shared` file.

3. Save the changes to `lsf.shared`.
4. Run **lsadmin reconfig** to reconfigure LIM.
5. Run `badmin reconfig` to reconfigure **mbatchd**.

Set clean period for DONE jobs

You can control the amount of time during which successfully finished jobs are kept in `mbatchd` memory. This is useful if you ran thousands of jobs which finished successfully and you do not want to keep them stored in memory, which results in receiving a huge list of jobs every time you query with **bjobs -a**.

You can use the `CLEAN_PERIOD_DONE` parameter in `lsb.params` to set the amount of time (in seconds) to keep `DONE` and `PDONE` (post job execution processing) jobs in `mbatchd` memory after they have finished.

For example, to clean `DONE` and `PDONE` jobs from memory after one day, set `CLEAN_PERIOD_DONE= 86400`.

To set the amount of time:

1. Configure `CLEAN_PERIOD_DONE` in `lsb.params`.
2. Run **`badmin reconfig`** to have the changes take effect.
3. Optional: Run **`bparams -a | grep CLEAN_PERIOD_DONE`** to verify the parameter setting:

```
bparams -a | grep CLEAN_PERIOD_DONE  
  
CLEAN_PERIOD_DONE  
= 604800
```

4. Submit your job.
5. You can see the configured time period for which successfully finished jobs are kept in `mbatchd` memory with the **`bparams`** command:

```
$ bparams -a  
...  
SCHEDULER_THREADS = 0  
BJOBS_RES_REQ_DISPLAY = brief  
CLEAN_PERIOD_DONE = 604800  
  
$ bparams -l  
The amount of time during which successfully finished jobs are kept in memory:  
CLEAN_PERIOD_DONE = 604800
```

When changing the value for `CLEAN_PERIOD_DONE`, note the following:

- `CLEAN_PERIOD_DONE` is limited to one week.
- The value for `CLEAN_PERIOD_DONE` must be less than the value for `CLEAN_PERIOD`, or the value is ignored and a warning message appears.
- If `CLEAN_PERIOD_DONE` is defined and historical run time is enabled, then a `DONE` job's historical run time will be used to calculate dynamic user priority until the job reaches its clean period which is `CLEAN_PERIOD_DONE`.

Enable host-based resources

Learn how Portable Hardware Locality (`hwloc`) is integrated into LSF to detect hardware information. Enable LSF so applications can use NVIDIA Graphic Processing Units (GPUs) and Intel Xeon Phi (MIC) co-processors in a Linux environment.

- [Portable hardware locality](#)
Portable Hardware Locality (**`hwloc`**) is an open source software package that is distributed under BSD license. It provides a portable abstraction (across OS, versions, architectures, and so on) of the hierarchical topology of modern architectures, including NUMA memory nodes, socket, shared caches, cores, and simultaneous multithreading (SMT). **`hwloc`** is integrated into LSF to detect hardware information, and can support most of the platforms that LSF supports.
- [Define GPU resources \(Obsolete\)](#)
Enable LSF so applications can use NVIDIA Graphic Processing Unit (GPU) resources in a Linux environment. LSF supports parallel jobs that request GPUs to specify some GPUs on each node at run time, based on availability.
- [Define Intel Xeon Phi resources](#)
Enable LSF so applications can use Intel Xeon Phi co-processors (previously referred to as Intel Many Integrated Core Architecture, or MIC, co-processors) in a Linux environment. LSF supports parallel jobs that request Xeon Phi resources, so you can specify some co-processors on each node at run time, based on availability.

Portable hardware locality

Portable Hardware Locality (**`hwloc`**) is an open source software package that is distributed under BSD license. It provides a portable abstraction (across OS, versions, architectures, and so on) of the hierarchical topology of modern architectures, including NUMA memory nodes, socket, shared caches, cores, and simultaneous multithreading (SMT). **`hwloc`** is integrated into LSF to detect hardware information, and can support most of the platforms that LSF supports.

Functionality

The **hwloc** package gathers various system attributes such as cache and memory information as well as the locality of I/O device such as network interfaces. It primarily aims at helping applications with gathering information about computing hardware.

It also detects each host hardware topology when the LIM starts and the host topology information is changed. The management host LIM detects the topology of the management host. The server host LIM detects the topology of the local host. It updates the topology information to the management host when it joins the cluster or sends topology information to the management host LIM for host configuration. Host topology information is updated once the hardware topology changes. Hardware topology changes if any NUMA memory node, caches, socket, core, PU and so on, changes. Sometimes topology information changes even though the core number did not change.

Use the **lim -T** and **lshosts -T** commands to display host topology information. The **lim -t** command displays the total number of NUMA nodes, total number of processors, total number of cores, and total number of threads.

Structure of topology

A NUMA node contains sockets. Each socket contains cores (processes) which contain threads. If there is no hwloc library, LSF uses the PCT logic. Some AMD CPUs have the opposite structure where socket nodes contain NUMA nodes. The hierarchies of the topology is similar to a tree. Therefore, the host topology information (NUMA memory nodes, caches, sockets, cores, PUs, and so on) from hwloc is organized as a tree. Each tree node has its type. The type includes host, NUMA, socket, cache, core, and pu. Each tree node also includes its attributes.

In the following example, `hostA` has 64 GB of memory and two NUMA nodes. Each socket node has one NUMA, eight cores, 16 PUs (two PUs per core), and 32 GB of memory. Both the NUMA nodes and the PUs are numbered in series that is provided by the system. LSF displays NUMA information based on the level it detects from the system. The output format displays as a tree, and the NUMA information displays as `NUMA[ID: memory]`. The PU displays as `parent_node(ID ID ...)`, where `parent_node` may be host, NUMA, socket, or core.

In the following example, `NUMA[0: 32G]` means that the NUMA ID is 0 and has 32 GB of memory. `core0(0 16)` means that there are two PUs under the parent core node, and the ID of the two PUs are 0 and 16.

```
Host[64G] hostA
Socket0
  NUMA[0: 32G]
    core0(0 16)
    core1(1 17)
    core2(2 18)
    core3(3 19)
    core4(4 20)
    core5(5 21)
    core6(6 22)
    core7(7 23)
Socket1
  NUMA[1: 32G]
    core8(8 24)
    core9(9 25)
    core10(10 26)
    core11(11 27)
    core12(12 28)
    core13(13 29)
    core14(14 30)
    core15(15 31)
```

Some CPUs, especially old ones, may have incomplete hardware topology in terms of missing information for NUMA, socket, or core. Therefore, their topology is incomplete.

For example,

- `hostB` (with one Intel Pentium 4 CPU) has 2G of memory, one socket, one core, and two PUs per core. Information on `hostB` is displayed as follows:

```
Host[2G] hostB
Socket
```



```
core (0 1)
```

- `hostC` (with one Intel Itanium CPU) has 4 GB of memory, and two PUs. Information on `hostC` is displayed as follows:

```
Host[4G] (0 1) hostC
```

Some platforms or operating system versions will only report a subset of topology information.

For example, `hostD` has the same CPU as `hostB`, but `hostD` is running RedHat Linux 4, which does not supply core information. Therefore, information on `hostD` is displayed as follows:

```
Host[1009M] hostD
Socket (0 1)
```

Dynamically load the `hwloc` library

You can configure LSF to dynamically load the **hwloc** library from the system library paths to detect newer hardware. This allows you to use the latest supported version of the **hwloc** and LSF integration at any time if there are no compatibility issues between this version of the **hwloc** library and header file for **hwloc**. If LSF fails to load the library, LSF defaults to using the **hwloc** functions in the static library.

Enable the dynamic loading of the **hwloc** library by enabling the `LSF_HWLOC_DYNAMIC` parameter in the `lsf.conf` file.

Define GPU resources (Obsolete)

Enable LSF so applications can use NVIDIA Graphic Processing Unit (GPU) resources in a Linux environment. LSF supports parallel jobs that request GPUs to specify some GPUs on each node at run time, based on availability.

Note: This topic is deprecated. For updated information on working with GPU resources, see [GPU resources](#). Specifically, LSF supports the following GPU features:

- NVIDIA GPUs for serial and parallel jobs. Use the **blaunch** command to start parallel jobs.
- CUDA 4.0 to CUDA 8.0, and later
- Linux-based GPU support on x64 for REHL/Fedora/SLES

LSF also supports the collection of metrics for GPUs by using ELIMs and predefined LSF resources.

The following information is collected by the **elim.gpu** ELIM:

```
ngpus
    Total number of GPUs.
ngpus_shared
    Number of GPUs in share mode. The resource ngpus_shared is a consumable resource in the LIM. Its value is set to the same number of CPU cores. You can place any number of tasks on the shared mode GPU, but more tasks might degrade performance.
ngpus_excl_t
    Number of GPUs in exclusive thread mode.
ngpus_excl_p
    Number of GPUs in exclusive process mode.
ngpus_physical
    Number of physical GPUs specified by the bsub -gpu "num" option. The resource ngpus_physical is a consumable resource in the LIM.
```

The following information is collected by the optional **elim.gpu.ext** ELIM:

```
ngpus_prohibited
    Number of GPUs prohibited.
gpu_driver
    GPU driver version.
gpu_mode*
    Mode of each GPU.
gpu_temp*
```

Temperature of each GPU.
`gpu_ecc*`
 ECC errors for each GPU.
`gpu_model*`
 Model name of each GPU.
`gpu_ut*`
 GPU utilization.
`gpu_shared_avg_ut`
 Average utilization of all GPUs in shared mode on the host.
`gpu_mut*`
 Percentage of GPU memory utilization.
`gpu_mtotal*`
 Total GPU memory size.
`gpu_mused*`
 Used GPU memory size.
`gpu_pstate*`
 Performance state of the GPU. The performance state is directly correlated to power consumption.
`gpu_shared_avg_mut`
 Average memory usage across all shared mode GPUs.
`gpu_status*`
 Status of the GPU, if the `LSF_DCGM_PORT` parameter is defined in the `lsf.conf` file.
`gpu_error*`
 Error message for the GPU, if the `LSF_DCGM_PORT` parameter is defined in the `lsf.conf` file. If the GPU status (`gpu_status*`) is `ok`, the message is a hyphen (-).

The asterisk (*) in the resource names indicates that if more than 1 GPU is reported, an index is appended to the resource name, starting at 0. For example, for `gpu_mode` you might see `gpu_mode0`, `gpu_mode1`, `gpu_mode2`.

The optional **elim.gpu.topology** ELIM detects GPU topology on the host by using the Portable Hardware Locality feature and enabling its NVML plug-ins. The Hardware Locality plug-in mechanism uses `libtool` to load dynamic libraries.

The `elim.gpu.topology` ELIM is precompiled with Hardware Locality version 1.8, which is compiled based on the `libtool` 1.5.2 library. If the `libtool` library in your environment is not 1.5.2, you need to build your own `elim.gpu.topology` ELIM. See the README file under `LSF_TOP/10.1.0/misc/examples/elim.gpu.ext/` for details.

LSF support for GPUs has the following conditions:

- In LSF 9.1.2 and later, the old `elim.gpu` ELIM is replaced with a new `elim.gpu` ELIM.
- Checkpoint and restart are not supported.
- Preemption is not supported.
- Resource duration and decay are not supported.
- ELIMs for CUDA 4.0 can work with CUDA 5.5.

Configure and use GPU resources

To configure and use GPU resources:

1. Binary files for base `elim.gpu` are located under `$LSF_SERVERDIR`. The binary for optional `elim.gpu.ext.c` and `elim.gpu.topology.c`, their makefiles and README files are under `LSF_TOP/10.1.0/misc/examples/elim.gpu.ext/`. See the README file for steps to build, install, configure, and debug the ELIMs. Make sure **elim** executable files are in the `LSF_SERVERDIR` directory.

For GPU support, make sure the following third-party software is installed correctly:

- CUDA driver
- CUDA toolkit
- Tesla Deployment Kit
- NVIDIA Management Library (NVML)
- CUDA sample is optional.
- CUDA version must be 4.0 or later.
- From CUDA 5.0, the CUDA driver, CUDA toolkit, and CUDA samples are in one package.
- Nodes must have at least one NVIDIA GPU from the Fermi/Kepler family. Earlier Tesla and desktop GPUs of 8800 and later cards are supported. Not all features are available for the earlier cards. Cards earlier than Fermi cards do not support ECC errors, and some do not support Temperature queries.

2. Optionally, enable integration with NVIDIA Data Center GPU Manager (DCGM).

The NVIDIA Data Center GPU Manager (DCGM) is a suite of data center management tools that allow you to manage and monitor GPU resources in an accelerated data center.

Enable the DCGM integration by defining the LSF_DCGM_PORT parameter in the lsf.conf file.

3. Configure the LSF cluster that contains the GPU resources:

- Configure lsf.shared.

For GPU support, define the following resources in the Resource section, assuming that the maximum number of GPUs per host is three. The first four GPUs are provided by base ELIMs. The others are optional. The `ngpus` resource is not consumable. Remove changes that are related to the old GPU solution before you define the new one:

Begin Resource					
RESOURCENAME	TYPE	INTERVAL	INCREASING	CONSUMABLE	DESCRIPTION
ngpus_prohibited (Prohibited Mode)	Numeric	60	N	N	(Number of GPUs in Prohibited Mode)
ngpus	Numeric	60	N	N	(Number of GPUs)
ngpus_shared (Shared Mode)	Numeric	60	N	Y	(Number of GPUs in Shared Mode)
ngpus_excl_t (Exclusive Thread Mode)	Numeric	60	N	Y	(Number of GPUs in Exclusive Thread Mode)
ngpus_excl_p (Exclusive Process Mode)	Numeric	60	N	Y	(Number of GPUs in Exclusive Process Mode)
ngpus_physical (GPUs)	Numeric	60	N	Y	(Number of physical GPUs)
gpu_driver	String	60	()	()	(GPU driver version)
gpu_mode0	String	60	()	()	(Mode of 1st GPU)
gpu_temp0 (GPU)	Numeric	60	Y	()	(Temperature of 1st GPU)
gpu_ecc0 (GPU)	Numeric	60	N	()	(ECC errors on 1st GPU)
gpu_model0 (GPU)	String	60	()	()	(Model name of 1st GPU)
gpu_mode1	String	60	()	()	(Mode of 2nd GPU)
gpu_temp1 (GPU)	Numeric	60	Y	()	(Temperature of 2nd GPU)
gpu_ecc1 (GPU)	Numeric	60	N	()	(ECC errors on 2nd GPU)
gpu_model1 (GPU)	String	60	()	()	(Model name of 2nd GPU)
gpu_mode2	String	60	()	()	(Mode of 3rd GPU)
gpu_temp2 (GPU)	Numeric	60	Y	()	(Temperature of 3rd GPU)
gpu_ecc2 (GPU)	Numeric	60	N	()	(ECC errors on 3rd GPU)
gpu_model2 (GPU)	String	60	()	()	(Model name of 3rd GPU)
gpu_ut0 (1st GPU)	Numeric	60	Y	()	(GPU utilization of 1st GPU)
gpu_ut1 (2nd GPU)	Numeric	60	Y	()	(GPU utilization of 2nd GPU)
gpu_ut2 (3rd GPU)	Numeric	60	Y	()	(GPU utilization of 3rd GPU)
gpu_shared_avg_ut (shared mode GPUs utilization)	Numeric	60	Y	()	(Average of all shared mode GPUs utilization)
gpu_topology (host)	String	60	()	()	(GPU topology on host)
gpu_mut0 (utilization of 1st GPU)	Numeric	60	Y	()	(GPU memory utilization of 1st GPU)
gpu_mut1 (utilization of 2nd GPU)	Numeric	60	Y	()	(GPU memory utilization of 2nd GPU)
gpu_mut2 (utilization of 3rd GPU)	Numeric	60	Y	()	(GPU memory utilization of 3rd GPU)
gpu_mtotal0 (GPU)	Numeric	60	Y	()	(Memory total of 1st GPU)
gpu_mtotal1 (GPU)	Numeric	60	Y	()	(Memory total of 2nd GPU)
gpu_mtotal2 (GPU)	Numeric	60	Y	()	(Memory total of 3rd GPU)
gpu_mused0	Numeric	60	Y	()	(Memory used of 1st GPU)

GPU)						
gpu_mused1	Numeric	60	Y	()		(Memory used of 2nd GPU)
gpu_mused2	Numeric	60	Y	()		(Memory used of 3rd GPU)
gpu_pstate0	String	60	()	()		(Performance state of 1st GPU)
gpu_pstate1	String	60	()	()		(Performance state of 2nd GPU)
gpu_pstate2	String	60	()	()		(Performance state of 3rd GPU)
gpu_shared_avg_mut	Numeric	60	Y	()		(Average memory of all shared mode GPUs)
gpu_status0	String	60	()	()		(GPU status)
gpu_status1	String	60	()	()		(GPU status)
gpu_status2	String	60	()	()		(GPU status)
gpu_error0	String	60	()	()		(GPU error)
gpu_error1	String	60	()	()		(GPU error)
gpu_error2	String	60	()	()		(GPU error)
...						
End Resource						

The gpu_status* and gpu_error* resources are only available if you enabled the DCGM integration by defining the LSF_DCGM_PORT parameter in the lsf.conf file.

- Configure the lsf.cluster.cluster_name file.
For GPU support, define the following resources in the ResourceMap section. The first four GPUs are provided by the **elims.gpu** ELIM. The others are optional. Remove changes that are related to the old GPU solution before you define the new one:

```
Begin ResourceMap
RESOURCENAME      LOCATION
...

ngpus_prohibited  ([default])
ngpus              ([default])
ngpus_shared      ([default])
ngpus_excl_t      ([default])
ngpus_excl_p      ([default])
ngpus_physical    ([hostA] [hostB])
gpu_mode0         ([default])
gpu_temp0         ([default])
gpu_ecc0          ([default])
gpu_mode1         ([default])
gpu_temp1         ([default])
gpu_ecc1          ([default])
gpu_mode2         ([default])
gpu_temp2         ([default])
gpu_ecc2          ([default])
gpu_model0        ([default])
gpu_model1        ([default])
gpu_model2        ([default])
gpu_driver        ([default])
gpu_ut0           ([default])
gpu_ut1           ([default])
gpu_ut2           ([default])
gpu_shared_avg_ut ([default])
gpu_topology      ([default])
gpu_mut0          ([default])
gpu_mut1          ([default])
gpu_mut2          ([default])
gpu_mtotal0       ([default])
gpu_mtotal1       ([default])
gpu_mtotal2       ([default])
gpu_mused0        ([default])
gpu_mused1        ([default])
gpu_mused2        ([default])
gpu_pstate0       ([default])
gpu_pstate1       ([default])
gpu_pstate2       ([default])
gpu_shared_avg_mut ([default])
gpu_status0       ([default])
```

```

gpu_status1      ([default])
gpu_status2      ([default])
gpu_error0       ([default])
gpu_error1       ([default])
gpu_error2       ([default])
...
End ResourceMap

```

The `gpu_status*` and `gpu_error*` resources are only available if you enabled the DCGM integration by defining the `LSF_DCGM_PORT` parameter in the `lsf.conf` file.

- Optionally, configure `lsb.resources`.
For the `ngpus_shared`, `gpuexcl_t` and `gpuexcl_p` resources, you can set attributes in the `ReservationUsage` section with the following values:

```

Begin ReservationUsage
RESOURCE      METHOD      RESERVE
ngpus_shared  PER_HOST      N
ngpus_excl_t  PER_HOST      N
ngpus_excl_p  PER_HOST      N
nmics         PER_TASK      N
End ReservationUsage

```

If this file has no configuration for GPU resources, by default LSF considers all resources as `PER_HOST`.

- Run the **lsadmin reconfig** and **badmin mbdrestart** commands to make configuration changes take effect. If you configure the resource `gpu_topology`, run the **bctrl restart sbd** command too.

- Use the **lsload -l** command to show GPU resources:

```

$ lsload -I ngpus:ngpus_shared:ngpus_excl_t:ngpus_excl_p
HOST_NAME      status  ngpus  ngpus_shared  ngpus_excl_t  ngpus_excl_p
hostA          ok      3.0    12.0          0.0           0.0
hostB          ok      -      -             -             -
hostC          ok      -      -             -             -
hostD          ok      -      -             -             -
hostE          ok      -      -             -             -
hostF          ok      3.0    12.0          0.0           0.0
hostG          ok      3.0    12.0          0.0           1.0
hostH          ok      3.0    12.0          1.0           0.0
hostI          ok      -      -             -             -

```

- Use the **bhost -l** command to see how the LSF scheduler allocated the GPU resources. These resources are treated as normal host-based resources:

```

$ bhosts -l hostA
HOST hostA
STATUS  CPUF  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV  DISPATCH_WINDOW
ok      60.00 -    12   2      2     0     0     0   -

CURRENT LOAD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem  slots  nmics
Total    0.0   0.0  0.0   0%  0.0  3   4   0  28G  3.9G  22.5G  10   0.0
Reserved 0.0   0.0  0.0   0%  0.0  0   0   0   0M   0M   0M    -   -

          ngpus  ngpus_shared  ngpus_excl_t  ngpus_excl_p
Total      3.0   10.0          0.0           0.0
Reserved   0.0    2.0          0.0           0.0

LOAD THRESHOLD USED FOR SCHEDULING:
          r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem
loadSched -    -    -    -    -    -    -    -    -    -    -
loadStop  -    -    -    -    -    -    -    -    -    -    -

          nmics  ngpus  ngpus_shared  ngpus_excl_t  ngpus_excl_p
loadSched -    -    -          -             -
loadStop  -    -    -          -             -

```

- Use the **lshosts -l** command to see the information for GPUs collected by **elim**:

```

$ lshosts -l hostA

```

```

HOST_NAME:  hostA
type      model      cpuf ncpus ndisks maxmem maxswp maxtmp rexpri server nprocs ncores
nthreads
X86_64    Intel_EM64T 60.0 12    1      23.9G 3.9G  40317M 0      Yes    2      6      1

RESOURCES: (mg)
RUN_WINDOWS: (always open)

LOAD_THRESHOLDS:
r15s  r1m  r15m ut pg io ls it tmp swp mem nmics ngpus ngpus_shared ngpus_excl_t
ngpus_excl_p
-      3.5 - - - - - - - - - - - - - - - -

```

You can also use the **bpost** command to display which GPUs are allocated to the job.

7. Submit jobs with GPU resources in resource requirements.

Use the `select[]` string in a resource requirement (-R) to choose the hosts that have GPU resources. Use the `rusage[]` resource requirement to tell LSF how many GPU resources to use.

Note: If the `LSB_GPU_NEW_SYNTAX=Y` parameter is specified in the `lsf.conf` file, you must submit your job with the **bsub -gpu** option. You cannot use the GPU resources `ngpus_shared`, `ngpus_excl_t` and `ngpus_excl_p`.

Examples:

- Use a GPU in shared mode:

```
bsub -R "select[ngpus>0] rusage [ngpus_shared=2]" gpu_app
```

- Use a GPU in exclusive thread mode for a PMPI job:

```
bsub -n 2 -R 'select[ngpus>0] rusage[ngpus_excl_t=2]' mpirun -lsf gpu_app1
```

- Use a GPU in exclusive process mode for a PMPI job:

```
bsub -n 4 -R "select[ngpus>0] rusage[ngpus_excl_p=2]" mpirun -lsf gpu_app2
```

- Run a job on 1 host with 8 tasks on it, using 2 `ngpus_excl_p` in total:

```
bsub -n 8 -R "select[ngpus > 0] rusage[ngpus_excl_p=2] span[hosts=1]" mpirun -lsf
gpu_app2
```

- Run a job on 8 hosts with 1 task per host, where every task uses 2 `gpushared` per host:

```
bsub -n 8 -R "select[ngpus > 0] rusage[ngpus_shared=2] span[ptile=1]" mpirun -lsf
gpu_app2
```

- Run a job on 4 hosts with 2 tasks per host, where the tasks use a total of 2 `ngpus_excl_t` per host.

```
bsub -n 8 -R "select[ngpus > 0] rusage[ngpus_excl_t=2] span[ptile=2]" mpirun -lsf
gpu_app2
```

8. Submit jobs with the **bsub -gpu** option.

The `LSB_GPU_NEW_SYNTAX=Y` parameter must be specified in the `lsf.conf` file to submit your job with the **bsub -gpu** option.

GPU access enforcement

LSF can enforce GPU access on systems that support the Linux cgroup devices subsystem. To enable GPU access through Linux cgroups, configure the `LSB_RESOURCE_ENFORCE="gpu"` parameter in the `lsf.conf` file. LSF creates devices to contain job processes if the job has GPU resource requirements so that the job processes cannot escape from the allocated GPUs. Each GPU job device includes only the GPUs that LSF distributes. Linux cgroup devices are only created for GPU jobs.

GPU enforcement for Linux cgroup device subsystems is supported on Red Hat Enterprise Linux 6.2 and later, and SuSe Linux Enterprise Linux 11 SP2 and later.

Note: When GPU enforcement is enabled, the GPUs that are contained in one device cgroup are reallocated new GPU IDs, beginning with 0. CUDA Version 7.0 or later supports cgroup completely.

Jobs can specify how job processes are to be bound to these computing elements. LSF uses the environment variable `CUDA_VISIBLE_DEVICES` to tell user applications which GPUs are allocated. It is possible for user applications to escape from the allocated GPUs by changing the `CUDA_VISIBLE_DEVICES` variable to use other GPUs directly.

For example, the following command submits a job with one exclusive thread GPU requirement:

```
bsub -R "rusage[ngpus_excl_t=1]" ./myapp
```

LSF creates a device that contains one exclusive thread GPU and attaches the process ID of the application **./myapp** to this device. The device serves as a strict container for job processes so that the application **./myapp** cannot use other GPUs.

Decrease GPU power consumption when a GPU is not in use

A GPU consumes significant power even when it idles. LSF provides configuration parameters to decrease the GPU power that is consumed if GPU is not in use within a specified time. By default, LSF does not power off a GPU even when it is idle.

Set the `LSB_GPU_POWEROFF_DURATION` parameter in the `lsf.conf` file to specify the minimum number of seconds before LSF can power off an idle GPU. When the `LSB_GPU_POWEROFF_DURATION` parameter is set, LSF tries to allocate the GPU that is not running in "MIN power limit" mode. If not enough GPUs are in "MAX power limit" mode, LSF allocates the GPUs that are in "MIN power limit" mode and switches those GPUs to run in "MAX power limit" mode.

If the `LSB_GPU_POWEROFF_DURATION=0` parameter is set, LSF powers off GPUs immediately after the job finishes.

LSF uses the following criteria to allocate the GPU flow:

- All GPUs are in the same PCI.
- Check whether the "MAX power limit" mode GPUs meets job requirements. If they do, LSF does not allocate the "MIN power limit" mode GPUs first. If they do not meet the requirements, LSF allocates all the GPUs to the job, including both "MAX power limit" and "MIN power limit" mode GPUs.

If the **sbatchd** daemon is restarted, the GPU idle time is recalculated.

NVIDIA K80 hardware supports switch power limits. The NVML library must be Version 6.340 or newer.

NVIDIA Data Center GPU Manager (DCGM) features

The NVIDIA Data Center GPU Manager (DCGM) is a suite of data center management tools that allow you to manage and monitor GPU resources in an accelerated data center. LSF integrates with NVIDIA DCGM to work more effectively with GPUs in the LSF cluster. DCGM provides additional functionality when working with jobs that request GPU resources by:

- providing GPU usage information for the jobs.
- checking the status of GPUs to automatically filter out unhealthy GPUs when the job allocates GPU resources. This ensures that jobs are running on healthy GPUs. DCGM provides mechanisms to check the GPU health and LSF integrates these mechanisms to check the GPU status before, during, and after the job is running to meet the GPU requirements. If the execution host's DCGM status is not valid, the **bjobs -l** command shows an error message. The job still runs, but GPU resource usage reports are not available from that host.
- automatically adding back any previously-unhealthy GPUs that are healthy again so that these GPUs are available for job allocation.
- synchronizing the GPU auto-boost feature to support jobs that run across multiple GPUs, including jobs that run across multiple GPUs on a single host.

Enable the DCGM integration by defining the `LSF_DCGM_PORT` parameter in the `lsf.conf` file. After enabling the parameter, you must start up DCGM to use the features.

Run the `-gpu` option with the **bjobs**, **bhist**, and **bacct** commands to display GPU usage information from DCGM after the job finishes. The `-gpu` option must be used with the following command options:

- For the **bjobs** command, you must run the `-gpu` option with the `-l` or `-UF` options.
- For the **bhist** command, you must run the `-gpu` option with the `-l` option.
- For the **bacct** command, you must run the `-gpu` option with the `-l` option.

Define Intel Xeon Phi resources

Enable LSF so applications can use Intel Xeon Phi co-processors (previously referred to as Intel Many Integrated Core Architecture, or MIC, co-processors) in a Linux environment. LSF supports parallel jobs that request Xeon Phi resources, so you can specify some co-processors on each node at run time, based on availability.

Specifically, LSF supports the following environments:

- Intel Xeon Phi co-processors for serial and parallel jobs. Use the **blaunch** command to launch parallel jobs.
- Intel Xeon Phi co-processor for LSF jobs in offload mode, both serial and parallel.
- CUDA 4.0 to CUDA 8.0 and later.
- Intel Xeon Phi co-processors support Linux x64.

LSF also supports the collection of metrics for Xeon Phi co-processors by using ELIMs and predefined LSF resources.

The **elim.mic** ELIM collects the following information:

- **elim.mic** detects the number of Intel Xeon Phi co-processors (`nmics`)
- For each co-processor, the optional **elim** detects the following resources:

```
mic_ncores*
    Number of cores.
mic_temp*
    Co-processor temperature.
mic_freq*
    Co-processor frequency.
mic_freemem*
    Co-processor free memory.
mic_util*
    Co-processor utilization.
mic_power*:
    Co-processor total power.
```

* If the resource consists of more than one resource, an index is displayed, starting at 0. For example, for `mic_ncores`, you might see `mic_ncores0`, `mic_ncores1`, and `mic_ncores2`, and so on.

When you enable LSF support for Intel Xeon Phi resources, note the following support:

- Checkpoint and restart are not supported.
- Preemption is not supported.
- Resource duration and decay are not supported.
- ELIMs for CUDA 4.0 can work with CUDA 8.0 or later.

Configure and use Intel Xeon Phi resources

Configure and use Intel Xeon Phi resources:

1. Binary files for the base `elim.mic` file are located under `$LSF_SERVERDIR`. The binary for `elim.mic.ext` script file is located under `LSF_TOP/10.1.0/util/elim.mic.ext`. Make sure that the **elim** executable files are in the `LSF_SERVERDIR` directory.

For Intel Xeon Phi co-processor support, make sure that the following third-party software is installed correctly:

- Intel Xeon Phi co-processor (Knights Corner).
 - Intel MPSS version 2.1.4982-15 or later.
 - Runtime support library/tools from Intel Xeon Phi offload support.
2. Configure the LSF cluster that contains the Intel Xeon Phi resources.
 - Configure the `lsf.shared` file.
For Intel Xeon Phi support, define the following resources in the `Resource` section. The first resource (`nmics`) is required. The others are optional:

Begin Resource	RESOURCENAME	TYPE	INTERVAL	INCREASING	CONSUMABLE	DESCRIPTION
	<code>nmics</code>	Numeric	60	N	Y	(Number of MIC devices)
	<code>mic_temp0</code>	Numeric	60	Y	N	(MIC device 0 CPU temp)
	<code>mic_temp1</code>	Numeric	60	Y	N	(MIC device 1 CPU temp)
	<code>mic_freq0</code>	Numeric	60	N	N	(MIC device 0 CPU freq)


```

mic_freq1      Numeric 60      N      N      (MIC device 1 CPU freq)
mic_power0     Numeric 60      Y      N      (MIC device 0 total power)
mic_power1     Numeric 60      Y      N      (MIC device 1 total power)
mic_freemem0   Numeric 60      N      N      (MIC device 0 free memory)
mic_freemem1   Numeric 60      N      N      (MIC device 1 free memory)
mic_util0      Numeric 60      Y      N      (MIC device 0 CPU utility)
mic_util1      Numeric 60      Y      N      (MIC device 1 CPU utility)
mic_ncores0    Numeric 60      N      N      (MIC device 0 number cores)
mic_ncores1    Numeric 60      N      N      (MIC device 1 number cores)
...
End Resource

```

Note: The `mic_util` resource is a numeric resource, so the `lsload` command does not display it as the internal resource.

- Configure the `lsf.cluster.cluster_namefile`.

For Intel Xeon Phi support, define the following lines in the `ResourceMap` section. The first resource (`nmics`) is provided by the `elim.mic`. The others are optional:

```

Begin ResourceMap
RESOURCENAME  LOCATION
...
nmics         [default]
mic_temp0     [default]
mic_temp1     [default]
mic_freq0     [default]
mic_freq1     [default]
mic_power0    [default]
mic_power1    [default]
mic_freemem0  [default]
mic_freemem1  [default]
mic_util0     [default]
mic_util1     [default]
mic_ncores0   [default]
mic_ncores1   [default]
...
End ResourceMap

```

- Configure the `nmics` resource in the `lsb.resources` file. You can set attributes in the `ReservationUsage` section with the following values:

```

Begin ReservationUsage
RESOURCE      METHOD      RESERVE
...
nmics         PER_TASK    N
...
End ReservationUsage

```

If this file has no configuration for Intel Xeon Phi resources, by default LSF considers all resources as `PER_HOST`.

3. Use the `lsload -l` command to show Intel Xeon Phi resources:

```

lsload -I nmics:ngpus:ngpus_shared:ngpus_excl_t:ngpus_excl_p
HOST_NAME      status nmics  ngpus  ngpus_shared  ngpus_excl_t  ngpus_excl_p
hostA          ok      -      3.0    12.0          0.0           0.0
hostB          ok      1.0    -      -             -             -
hostC          ok      1.0    -      -             -             -
hostD          ok      1.0    -      -             -             -
hostE          ok      1.0    -      -             -             -
hostF          ok      -      3.0    12.0          0.0           0.0
hostG          ok      -      3.0    12.0          0.0           1.0
hostH          ok      -      3.0    12.0          1.0           0.0
hostI          ok      2.0    -      -             -             -

```

4. Use the `bhost -l` command to see how the LSF scheduler allocated Intel Xeon Phi resources. These resources are treated as normal host-based resources:

```

bhosts -l hostA
HOST hostA
STATUS  CPUF  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV  DISPATCH_WINDOW
ok      60.00 -     12   2      2      0      0      0    -

```

CURRENT LOAD USED FOR SCHEDULING:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem	slots	nmics
Total	0.0	0.0	0.0	0%	0.0	3	4	0	28G	3.9G	22.5G	10	0.0
Reserved	0.0	0.0	0.0	0%	0.0	0	0	0	0M	0M	0M	-	-

	ngpus	ngpus_shared	ngpus_excl_t	ngpus_excl_p
Total	3.0	10.0	0.0	0.0
Reserved	0.0	2.0	0.0	0.0

LOAD THRESHOLD USED FOR SCHEDULING:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

	nmics	ngpus	ngpus_shared	ngpus_excl_t	ngpus_excl_p
loadSched	-	-	-	-	-
loadStop	-	-	-	-	-

- Use the **lshosts -l** command to see the information for Intel Xeon Phi co-processors that are collected by the **elim**:

```
lshosts -l hostA
```

```
HOST_NAME: hostA
```

type	model	cpuf	ncpus	ndisks	maxmem	maxswp	maxtmp	rexpri	server	nprocs	ncores
nthreads											
X86_64	Intel_EM64T	60.0	12	1	23.9G	3.9G	40317M	0	Yes	2	6

```
RESOURCES: (mg)
```

```
RUN_WINDOWS: (always open)
```

LOAD THRESHOLDS:

r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem	nmics	ngpus	ngpus_shared	ngpus_excl_t	ngpus_excl_p
-	3.5	-	-	-	-	-	-	-	-	-	-	-	-	-	-

- Submit jobs. Use the `select[]` string in a resource requirement (`-R` option) to choose the hosts that have Intel Xeon Phi resources. Use the `rusage[]` string to tell LSF how many resources to use.

- Use Intel Xeon Phi resources in a LSF MPI job:

```
bsub -n 4 -R "rusage[nmics=2]" mpirun -lsf mic_app
```

- Request Intel Xeon Phi co-processors.

```
bsub -R "rusage[nmics=n]"
```

- Consume one Intel Xeon Phi resource on the execution host:

```
bsub -R "rusage[nmics=1]" mic_app
```

- Run the job on one host and consume 2 Intel Xeon Phi resources on that host:

```
bsub -R "rusage[nmics=2]" mic_app
```

Global fair share scheduling

The global fair share scheduling policy divides the processing power of the IBM® Spectrum LSF multicluster capability and the LSF/XL feature of IBM Spectrum LSF Advanced Edition among users to provide fair access to all resources, so that every user can use the resources of multiple clusters according to their configured shares.

Global fair share is supported in LSF Standard Edition and LSF Advanced Edition.

Global fair share supports the following features:

- Queue level user-based fair share.
- Cross queue user-based fair share. You configure the parent queue as a participant of global fair share. Participants can be any queues, users or user groups participating in the global fair share policy. There is no need to configure a child queue as a participant since it does not synchronize data for the global fair share policy.

- Parallel fair share: LSF can consider the number of CPUs when using global fair share scheduling with parallel jobs.

Global fair share supports 4096 user groups in a fair share tree.

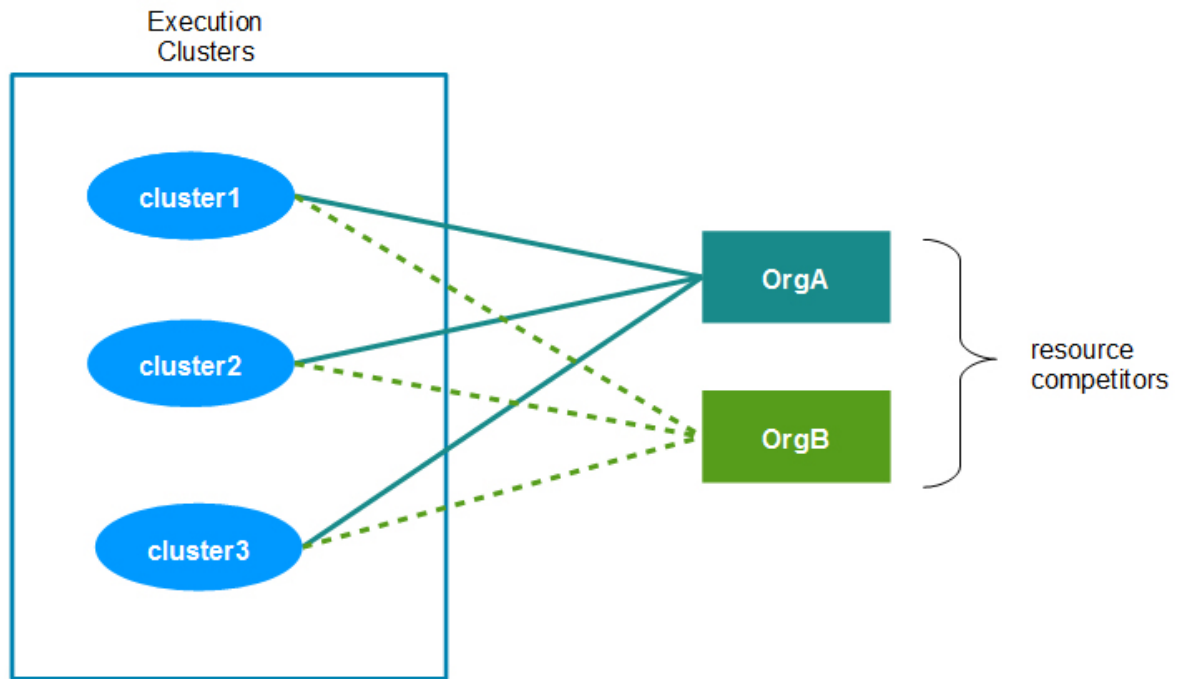
Global fair share scheduling is based on queue-level user-based fair share scheduling. LSF clusters running in geographically separate sites connected by LSF multicluster capability can maximize resource utilization and throughput.

- [Global fair share background](#)
You can run LSF clusters in geographic sites connected by LSF multicluster capability to maximize resource utilization and throughput. Most common is to configure hierarchy fair share to ensure resource fairness among projects and users. The same fair share tree may be configured in all clusters for the same organization because users may be mobile and can log into multiple clusters. However, fair share is local to each cluster, and each user's resource usage may be fair from one cluster angle, but completely unfair from global perspective.
- [Remote fair share load](#)
In a global fair share participating queue, each share account (`user_group/project/user`) has a property called remote fair share load. Remote share load indicates the aggregated resource usage of the same share account on all other global fair share participating queues within same global fair share policy. It is a floating-point value and it grows if the share account (`user_group/project/user`) uses more resources on other clusters.
- [Sync mode of global fair share policy](#)
The global fair share policy has a property called sync mode. Sync mode controls which share accounts in the global fair share participating queues can become global share accounts. There are two sync modes: **all-mode** and **partial-mode**.
- [Global fair share setup and configuration](#)
- [Global policy daemon](#)
Global fair share uses a daemon called global policy daemon (**gpolicyd**) to control global policy management across clusters.
- [Global fair share policy](#)
A global fair share policy is defined in the **Globalfair** share section of the configuration file in `$LSF_ENVDIR/lsbatch/cluster_name/configdir/lsb.globalpolicies`. The global fair share policy uses the **gpolicyd** daemon, which control which fair share queues, from which clusters, can participate and exchange resource usage. The local cluster does not need any configuration for the queue to participate.
- [Global fair share dynamic user priority](#)
When a local fair share policy schedules jobs, the dynamic user priority calculation also considers resource consumption of users in remote clusters. This prevents fair share from accruing locally to each cluster and ensures fair usage across clusters.
- [Share load synchronization rules](#)
- [Configure queue level user-based global fair share](#)
The following example describes how to apply global fair share scheduling for all nodes in a tree called **HighIO Orgs** among three clusters, using queue level user-based fair share scheduling and all-mode synchronization.
- [Configure cross-queue user-based global fair share](#)
Cross-queue user-based fair share works with global fair share. Configure cross-queue user-based global fair share in the same way as queue-level user-based global fair share, except that you can configure only the parent queue as a participant of the global fair share policy. If you configure a child queue as a participant, the child queue never synchronizes with global fair share.
- [Global fair share scheduling constraints](#)
When applying global fair share scheduling, there are factors to note.

Global fair share background

You can run LSF clusters in geographic sites connected by LSF multicluster capability to maximize resource utilization and throughput. Most common is to configure hierarchy fair share to ensure resource fairness among projects and users. The same fair share tree may be configured in all clusters for the same organization because users may be mobile and can log into multiple clusters. However, fair share is local to each cluster, and each user's resource usage may be fair from one cluster angle, but completely unfair from global perspective.

Figure 1. Typical LSF multicluster capability Environment



[Figure 1](#) illustrates a typical multicluster environment. **OrgA** and **OrgB** are resource competitors and they both can use resources in the three clusters **cluster1**, **cluster2**, and **cluster3**. Under the current LSF fair share scheduling, when **OrgA** and **OrgB** compete resources in **cluster1** at the same time, LSF determines which one can get the resources first only by the resource usages for **OrgA** and **OrgB** in **cluster1**. Their resource usage in **cluster2** and **cluster3** are ignored. This is fair just from cluster **cluster1**'s perspective. But from global perspective, if **OrgA** is using more resources than **OrgB** in **cluster2** and **cluster3** now, it is unfair for **OrgB**. To improve the resource usage fairness among clusters, a new LSF feature Global fair share Scheduling is introduced. Global fair share scheduling is a new LSF job scheduling policy, under which job scheduling order of competitors is determined by resource usage among clusters.

Definitions used in this section

Global fair share scheduling

A job scheduling policy based on queue-level user-based fair share scheduling policy. Global fair share scheduling can balance users' resource usage across clusters.

Global policy daemon

The **gpolicyd** daemon is responsible for exchanging resource usage across clusters.

Global fair share policy

A policy that controls which queues from which cluster can exchange resource usage of share accounts with each other.

It is defined in new configuration file: `$LSF_ENVDIR/lsbatch/cluster_name/configdir/lsb.globalpolicies`.

Global policy daemon cluster (GPD cluster)

A cluster that is configured to run **gpolicyd** on it.

Global fair share participating cluster

A cluster that is configured to be able to connect to **gpolicyd**.

Global fair share participating queue

A fair share queue in a global fair share participating cluster, and it is a participant of a global fair share policy.

Global share account

In a fair share tree of a global fair share participating queue, if a share account is configured to be able to participating the global fair share policy, it is called as a global share account.

Local share account

In a fair share tree of a global fair share participating queue, if a share account is not configured to be able to participating the global fair share policy, it is called as a local share account.

Remote fair share load

In a global fair share participating queue, each share account (`user_group/project/user`) has a property called remote fair share load. Remote share load indicates the aggregated resource usage of the same share account on all other global fair share participating queues within same global fair share policy. It is a floating-point value and it grows if the share account (`user_group/project/user`) uses more resources on other clusters.

Command `bqueues -r` can show each share account's remote fair share load.

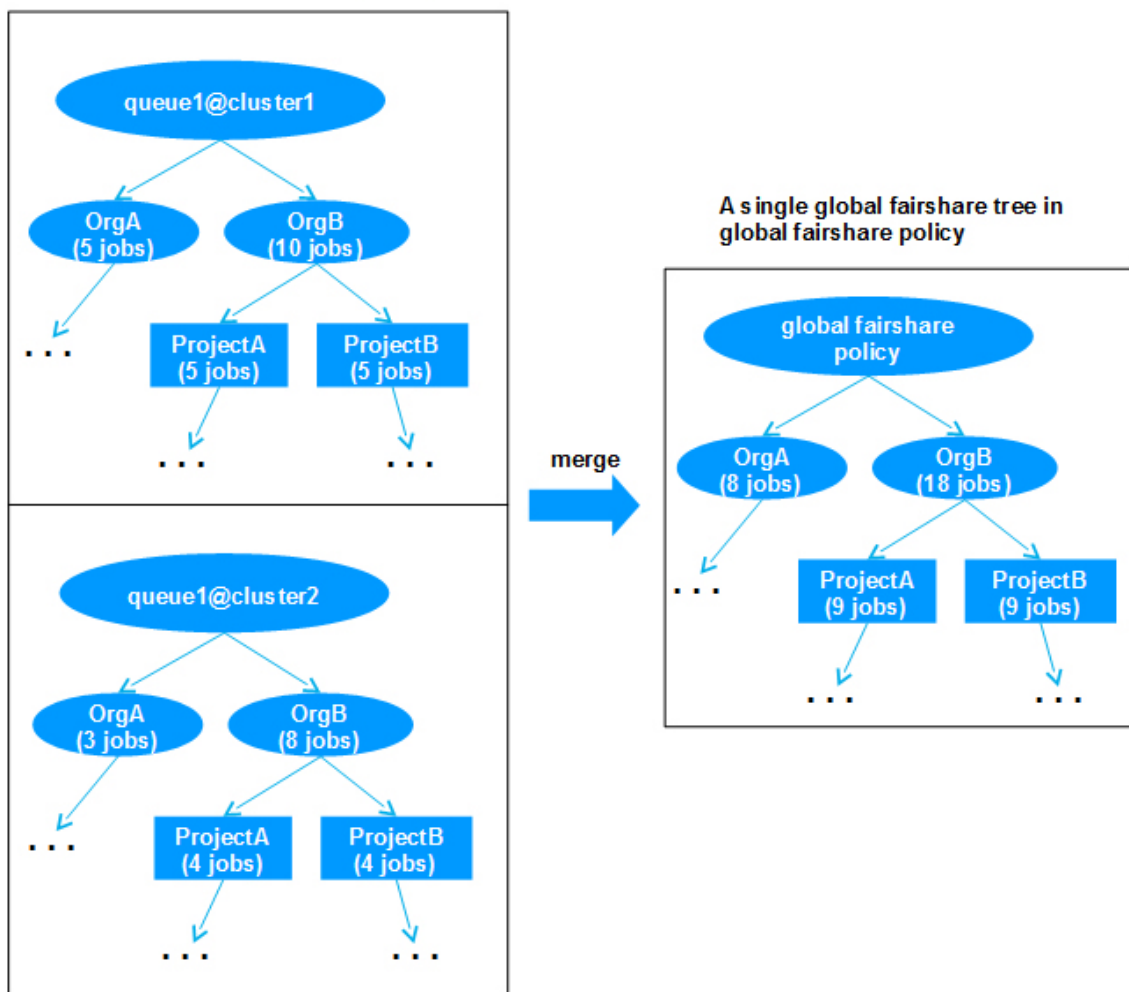
Sync mode of global fair share policy

The global fair share policy has a property called sync mode. Sync mode controls which share accounts in the global fair share participating queues can become global share accounts. There are two sync modes: **all-mode** and **partial-mode**.

If a global fair share policy is configured as **all-mode**, all share accounts of each participating queue will become global share accounts.

Figure 1. Global fair share tree

Local fairshare trees in each participants



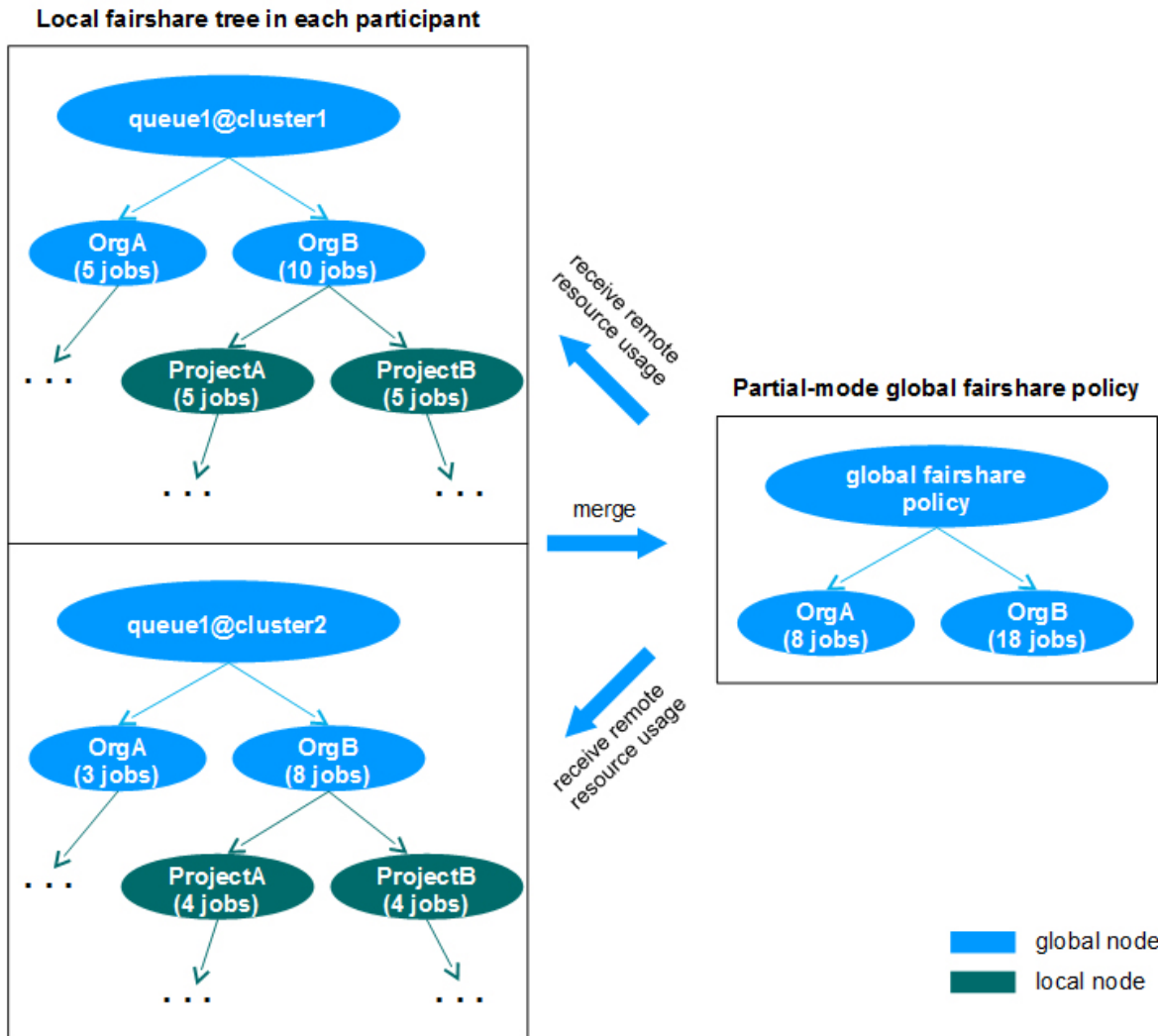
This diagram illustrates an **all-mode** global fair share policy. In the illustration, all share accounts in each participating queues are global.

If a global fair share policy is configured as **partial-mode**, which share accounts can become global share accounts, is controlled by fair share tree configuration of each participating queue. In participating queues, only the share accounts who meet all the following conditions can become global share accounts:

- In `lsb.users`, `FS_POLICY` parameter is configured for the share accounts.
- The name of the global fair share policy, which the queue participates into, is a member of `FS_POLICY`.

The following diagram illustrates a partial-mode global fair share policy. In this illustration only the global nodes are global share accounts.

Figure 2. Partial-mode global fair share policy



Configure all-mode global fair share policy

Just set `SYNC_MODE` to `all` explicitly for the global fair share policy in `lsb.globalpolicies`:

```
Begin GlobalFairshare
Name = policy1
PARTICIPANTS = queue1@cluster1 queue1@cluster2
SYNC_MODE = all
End GlobalFairshare
```

Configure partial-mode global fair share policy

Using [Figure 2](#), as an example. The following steps show how to configure such a partial-mode global fair share policy (assume that fair share tree of `queue1@cluster1` and `queue1@cluster2` have been configured):

1. Set `SYNC_MODE` to `partial` for the policy:

```

Begin GlobalFairshare
Name = policy1
PARTICIPANTS = queue1@cluster1 queue1@cluster2
SYNC_MODE = partial
End GlobalFairshare

```

2. In each participating cluster, ensure that the share accounts **OrgA** and **OrgB** are global by setting the global fair share policy name `policy1` into the `FS_POLICY` parameter in the `lsb.users` file:

```

Begin UserGroup
GROUP_NAME  GROUP_MEMBER      USER_SHARES      FS_POLICY
ProjectA    (user1 user2)      ([default,10])    ()
ProjectB    (user3 user4)      ([default,10])    ()
OrgB        (ProjectA ProjectB) ([default,10])    ()
top         (OrgA OrgB)       ([OrgA,40] [OrgB,60]) (policy1)
End UserGroup

```

Global fair share setup and configuration

Complete the following steps to set up global fair share:

1. [Install and configure LSF global fair share](#)
2. [Configure fair share queues](#)

The following describes steps to configure global fair share:

- [Disable global fair share scheduling](#)
- [Change the GPD cluster to another cluster](#)
- [Change LSB_GPD_PORT](#)

Install and configure LSF global fair share

LSF global fair share files are automatically installed by LSF's regular setup program (`lsfinstall`). But global fair share is disabled by default. To enable global fair share feature, the following steps are necessary:

1. Choose a cluster as the GPD cluster and ensure the GPD cluster can connect with other clusters that are to be global fair share participating clusters by LSF MC.
2. Configure fair share queues in global fair share participating clusters.
3. Configure the `LSB_GPD_PORT` and `LSB_GPD_CLUSTER` parameters in the `lsf.conf` file in the GPD cluster and all global fair share participating clusters.
 - The value of `LSB_GPD_PORT` must be the same in all clusters.
 - The value of `LSB_GPD_CLUSTER` must be the name of the GPD cluster in all clusters.
4. In the GPD cluster, create a configuration file named `lsb.globalpolicies` in directory `$LSF_ENVDIR/lsbatch/cluster_name/configdir/` and then configure global fair share policies in it. For the format of `lsb.globalpolicies`, see `lsb.globalpolicies` parameters.
5. Start the `gpolicyd` daemon on the GPD cluster.
 - If the GPD cluster is down, start the cluster. `gpolicyd` will be started automatically when the cluster starts.
 - If the GPD cluster is already on, run `bctrl restart sbd` to restart `sbatchd` on the management host and all management candidate hosts.
6. Restart the `mbd` daemon on each global fair share participating cluster. (If the cluster is down, start the cluster.)

Configure fair share queues

For participating queues in the same global fair share policy, keep fair share configuration the same on all clusters. The advantage is that there is only one copy of fair share configuration, and dynamic user priority of a share account will be the same on all participating clusters. This way, it is closer than real theoretical global fair share: one share account one dynamic user priority.

A different fair share configuration in participating queues is also acceptable by LSF and global fair share scheduling will work as expected.

Disable global fair share scheduling

To disable global fair share scheduling in a global fair share enabled multiple cluster environment, follow these following steps:

1. In the GPD cluster, comment out or remove the `LSB_GPD_CLUSTER` and `LSB_GPD_PORT` parameters from the `lsf.conf` file.
2. In the GPD cluster, run `bctrld restart sbd` to restart `sbatchd` on the management host and all management candidate hosts.
3. In the GPD cluster, restart `mbatchd` using `badmin mbdrestart`.
4. In all global fair share participating clusters, comment out or remove the `LSB_GPD_CLUSTER` and `LSB_GPD_PORT` parameters from the `lsf.conf` file.
5. In all global fair share participating clusters, restart `mbatchd` using `badmin mbdrestart`.

Change the GPD cluster to another cluster

1. Disable global fair share scheduling.
2. In the new GPD cluster, set the `LSB_GPD_CLUSTER` and `LSB_GPD_PORT` parameters in the `lsf.conf` file.
3. In the new GPD cluster, run `bctrld restart sbd` to restart `sbatchd` on the management host and all management candidate hosts.
4. In the global fair share participating clusters, set the `LSB_GPD_CLUSTER` and `LSB_GPD_PORT` parameters in the `lsf.conf` file.
5. In the global fair share participating clusters, restart `mbatchd` for the changes to take effect:

```
badmin mbdrestart
```

Change LSB_GPD_PORT

1. Disable global fair share scheduling.
2. In the GPD cluster, configure the new value of for the `LSB_GPD_CLUSTER` parameter in the `lsf.conf` file.
3. In the GPD cluster, run `bctrld restart sbd` to restart `sbatchd` on the management host and all management candidate hosts.
4. In the global fair share participating clusters, configure the new value of `LSB_GPD_PORT` in `lsf.conf`.
5. In the global fair share participating clusters, restart `mbatchd` for the changes to take effect:

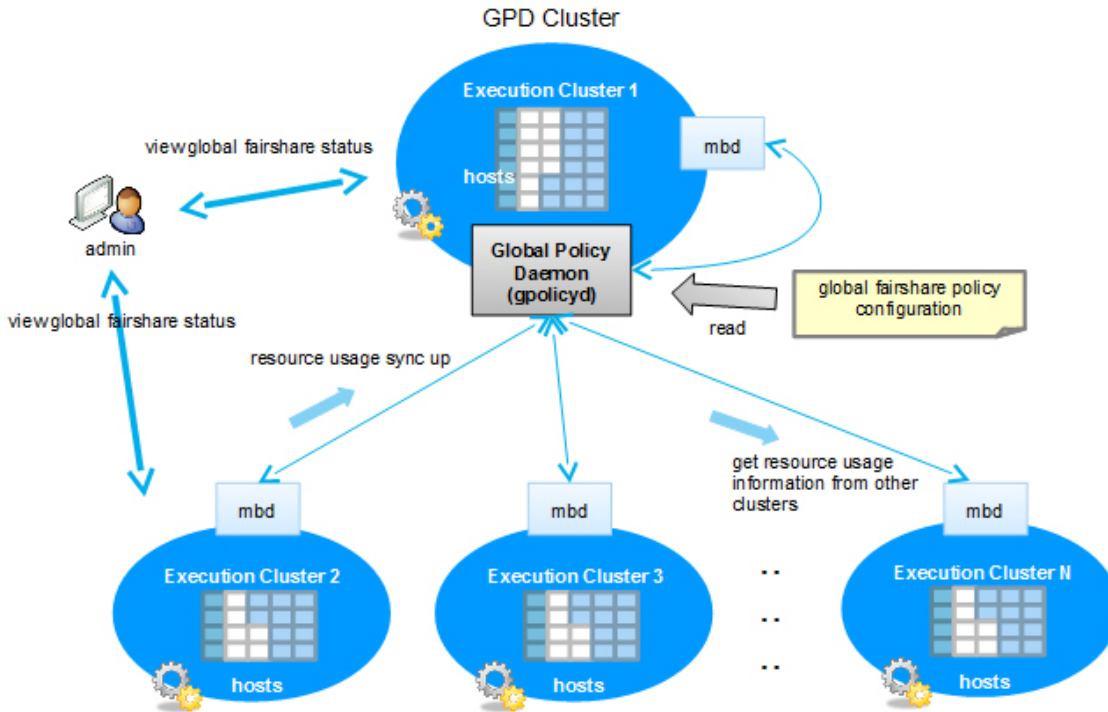
```
badmin mbdrestart
```

Global policy daemon

Global fair share uses a daemon called global policy daemon (**gpolicyd**) to control global policy management across clusters.

gpolicyd collects and broadcasts resource usage among clusters when using the LSF multicluster capability or LSF/XL feature. When a cluster schedules users' jobs, it applies global resource usage to determine the scheduling order. **gpolicyd** listens on the port you define in `LSB_GPD_PORT` (in `lsf.conf`) for synchronizing global fair share data among clusters and serving command line request. It receives fair share loads from all clusters which participate in global fair share. **gpolicyd** can synchronize global fair share load for at least 32 clusters. **gpolicyd** then broadcasts remote fair share load to all clusters.

gpolicyd only runs on the management host of one cluster that is regarded as the global policy daemon cluster (GPD cluster), and this host must be a UNIX host. **gpolicyd** is started by **sbatchd** on that management host. To configure the management host to start **gpolicyd**, specify a cluster name in `LSB_GPD_CLUSTER` (in `lsf.conf`). If **gpolicyd** dies, **sbatchd** restarts it.



Global fair share policy

A global fair share policy is defined in the **Globalfair** share section of the configuration file in `$LSF_ENVDIR/lsbatch/cluster_name/configdir/lsb.globalpolicies`. The global fair share policy uses the **gpolicyd** daemon, which control which fair share queues, from which clusters, can participate and exchange resource usage. The local cluster does not need any configuration for the queue to participate.

The example below shows a global fair share policy configuration. The name of the policy is `policy1`. It has two participants, `queue1@cluster1` and `queue1@cluster2`. Only share accounts in **queue1** at **cluster1** and **queue1** at **cluster2** can exchange resource usage with each other.

```

Begin Globalfair share
NAME = policy1
PARTICIPANTS = queue1@cluster1 queue1@cluster2
End Globalfair share

```

Multiple global fair share policies can be configured and coexist. For each global fair share policy, only one fair share queue per cluster can be configured as a participant.

You can display remote fair share load with the **bgpinfo** command. The remote fair share load impacts users' dynamic priority for job scheduling.

Global fair share dynamic user priority

When a local fair share policy schedules jobs, the dynamic user priority calculation also considers resource consumption of users in remote clusters. This prevents fair share from accruing locally to each cluster and ensures fair usage across clusters.

For a global fair share participating queue, remote fair share load will be a factor in dynamic user priority for the share account in the queue. If a share account uses many resources on other clusters, its dynamic user priority will be lower compared with its competitors that use fewer resources on other clusters. Dynamic user priority is calculated by global resource usage to achieve scheduling fairness among clusters.

How LSF calculates dynamic priority for global fair share

For a global fair share participating queue, the formula for calculating dynamic user priority of a share account is as follows:

```
dynamic priority = number_shares /  
(cpu_time * CPU_TIME_FACTOR  
+ (historical_run_time + run_time) * RUN_TIME_FACTOR  
+ (committed_run_time - run_time) * COMMITTED_RUN_TIME_FACTOR  
+ (1 + job_slots) * RUN_JOB_FACTOR  
+ (1 + fwd_job_slots) * FWD_JOB_FACTOR  
+ fair_share_adjustment * fair_share_ADJUSTMENT_FACTOR  
+ remote_fair_share_load)  
+ ((historical_gpu_run_time + gpu_run_time) * ngpus_physical) * GPU_RUN_TIME_FACTOR
```

where:

```
remote_fair_share_load = cpu_time_remote * CPU_TIME_FACTOR  
+ (historical_run_time_remote + run_time_remote) * RUN_TIME_FACTOR  
+ (committed_run_time_remote - run_time_remote) * COMMITTED_RUN_TIME_FACTOR  
+ job_slots_remote * RUN_JOB_FACTOR  
+ (1 + fwd_job_slots_remote) * FWD_JOB_FACTOR  
+ fair_share_adjustment_remote * fair_share_ADJUSTMENT_FACTOR  
+ ((historical_gpu_run_time_remote + gpu_run_time_remote) * ngpus_physical) *  
GPU_RUN_TIME_FACTOR
```

Whether or not `ENABLE_HIST_RUN_TIME` is set for a global fair share queue, the historical run time for share accounts in the global fair share queue is reported to GPD. When GPD receives historical run time from one cluster, it broadcasts the historical run time to other clusters. The local configuration determines whether the remote historical run time received from GPD is used in the calculation for fair share scheduling priority for the queue. That is, if `ENABLE_HIST_RUN_TIME` is set in the local cluster, the remote historical run time is used in the calculation for fair share scheduling priority for the queue.

As with local fair share, you can give additional weight to the various factors in the priority calculation by setting the following parameters for the queue in `lsb.queues` or for the cluster in `lsb.params`:

- `CPU_TIME_FACTOR`
- `RUN_TIME_FACTOR`
- `ENABLE_HIST_RUN_TIME`
- `COMMITTED_RUN_TIME_FACTOR`
- `NUM_RESERVE_JOBS`
- `NUM_START_JOBS`
- `SHARE_ADJUSTMENT`

When the queue value is not defined, the cluster-wide value from `lsb.params` is used.

Share load synchronization rules

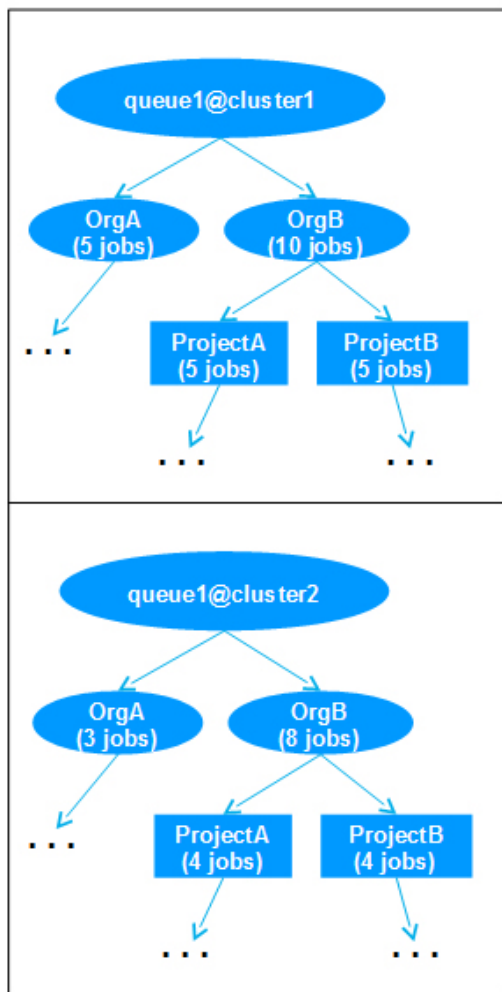
The basic rule is, only share accounts from global fair share participating clusters, which have the same SAAP (share attribute account path) can sync up share load with each other through `gpolicyd`. For example, a share account (`/ug1/user1`) on `cluster1` can sync up share load only with share account (`/ug1/user1`) on `cluster2` and share account (`/ug1/user1`) on `cluster3` through `gpolicyd`.

This rule can be applied to any global share accounts including the share accounts that are created by keywords (default, all, others, group_name@) configured in `lsb.users` or `lsb.queues`.

Global fair share policy holds a global fair share tree that is merged from the fair share trees of each participant. The global fair share tree holds global resource usage for share accounts.

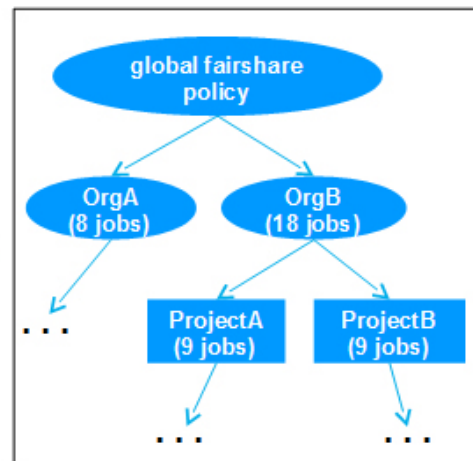
The figure below shows the merging process of a global fair share tree. `OrgB` runs 10 jobs in `cluster1` and 8 jobs in `cluster2`. In the global fair share tree, the node for `OrgB` holds the global resource usage (18 jobs) for the share account.

Local fairshare trees in each participants



A single global fairshare tree in global fairshare policy

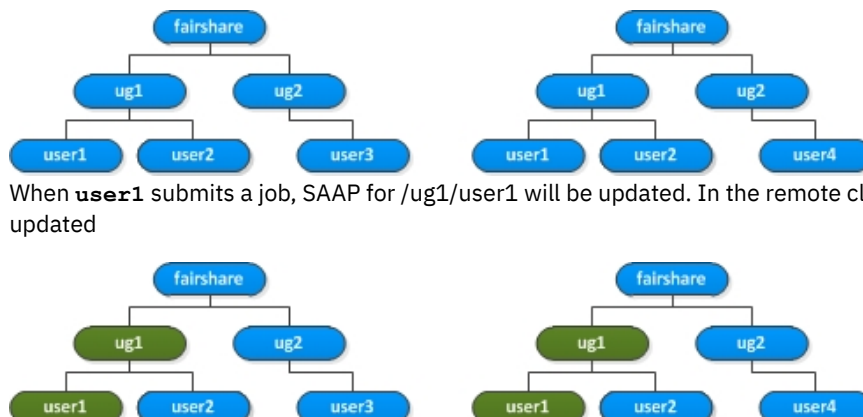
merge



Only share accounts with the same share attribute account path (SAAP) from clusters participating in global fair share can synchronize their share loads with each other through **gpolicyd**.

For example, a share account (/ug1/user1) on cluster1 can only synchronize share load with a share account (/ug1/user1) on cluster2 and a share account (/ug1/user1) on cluster3 through **gpolicyd**.

Since global fair share is distributed, the fair share tree may be different. In that case, only the nodes with matching SAAPs are updated. The unmatched share load information is dropped. For example, assume clusters A and B participate in one global fair share. All of the nodes need to synchronized their fair share data for global fair share. They have similar fair share trees. Only **ug2** has a different configuration:



When **user1** submits a job, SAAP for /ug1/user1 will be updated. In the remote cluster, the corresponding SAAP will also be updated



When **user3** submits a job, SAAP /ug2/user3 will be updated. In the remote cluster, only the valid corresponding SAAP **ug2** will be updated.



Global fair share synchronizes load data when **mbatchd** is connected to **gpolicyd**. **mbatchd** reports its fair share load to **gpolicyd** every 30 seconds by default. **gpolicyd** also broadcasts global fair share load to each cluster every 30 seconds by default. You can configure this time interval with the SYNC_INTERVAL parameter in the lsb.globalpolicies configuration file.

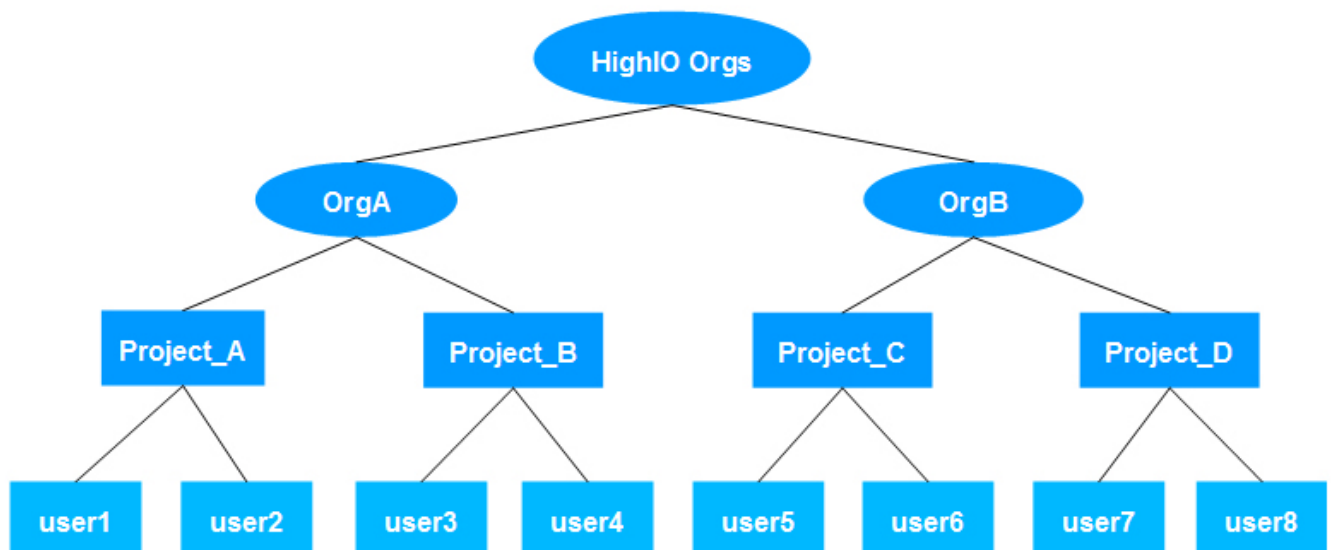
Delays are typical for data synchronization in distributed systems: For global fair share, when the local **mbatchd** receives remote fair share loads, the loads do not reflect the real time loads in other clusters due to any inherent delays and the delay you specified in SYNC_INTERVAL.

Configure queue level user-based global fair share

The following example describes how to apply global fair share scheduling for all nodes in a tree called **HighIO Orgs** among three clusters, using queue level user-based fair share scheduling and all-mode synchronization.

In this tree:

- There are three execution clusters (cluster1, cluster2 and cluster3) connected together with an LSF multicluster environment.
- **OrgA** and **OrgB** share the same resources in all execution clusters.
- Users in **OrgA** and **OrgB** can log in to all the execution clusters and submit jobs.



To apply global fair share scheduling:

1. In each cluster, configure queue level user-based fair share scheduling for each node in the **HighIO Orgs** tree. Assign each node the same number of shares in lsb.users and lsb.queues.

In lsb.users:

```
Begin UserGroup
GROUP_NAME      GROUP_MEMBER      USER_SHARES
project_a       (user1 user2)      ([default,100])
project_b       (user3 user4)      ([default,100])
project_c       (user5 user6)      ([default,100])
project_d       (user7 user8)      ([default,100])
org_a           (project_a project_b) ([default,100])
org_b           (project_c project_d) ([default,100])
high_io_orgs    (org_a org_b)
End UserGroup
```

In lsb.queues:

```

Begin Queue
QUEUE_NAME    = high_io_queue
...
FAIRSHARE     = USER_SHARES [[high_io_orgs, 100]]
End Queue

```

2. Choose one of the three clusters as the GPD cluster in which daemon **gpolicyd** will run. Assuming that cluster1 is the GPD cluster, configure the correct LSB_GPD_CLUSTER and LSB_GPD_PORT in lsf.conf for all clusters:

```

LSB_GPD_CLUSTER="cluster1"
LSB_GPD_PORT=8885

```

3. In cluster1, configure a global fair share policy in the configuration file \$LSF_ENVDIR/lsbatchd/cluster_name/lsb.globalpolicies:

```

Begin GlobalFairshare
NAME = high_io_policy
PARTICIPANTS = high_io_queue@cluster1 high_io_queue@cluster2 high_io_queue@cluster3
SYNC_MODE = all
End GlobalFairshare

```

The global fair share policy **high_io_policy** includes **high_io_queue** for each of the three participating clusters so that each **high_io_queue** in each cluster can share resource usage with each other.

4. After configuring lsb.globalpolicies, use the **badmadmin gpdckconfig** command to check if the configuration is correct:

```

$ badmadmin gpdckconfig
Checking configuration files ...
No errors found.

```

5. Start **cluster1**, **cluster2**, and **cluster3**.
Global fair share scheduling then takes effect for each node in the **HighIO Orgs** tree for the three clusters.

Once global fair share scheduling is applied, you can:

- Check the status of the global fair share policies by running the **bgpinfo** command in one of the three clusters:

```

$ bgpinfo status -l
GPOLICYD CLUSTER: cluster1
GPOLICYD HOST: management_host

GLOBAL FAIRSHARE POLICY

POLICY NAME: high_io_policy
PARTICIPANTS      STATUS
high_io_queue@cluster1    ok
high_io_queue@cluster2    ok
high_io_queue@cluster3    ok

```

- Check the configuration of global fair share policies by running the **bgpinfo conf** command in one of the three clusters:

```

$ bgpinfo conf -l

POLICY NAME: high_io_policy
-- No description provided.

TYPE: fairshare
PARAMETERS:
SYNC_INTERVAL: 30 seconds
SYNC_MODE: all
PARTICIPANTS: high_io_queue@cluster1 high_io_queue@cluster2 high_io_queue@cluster3

```

- Use the **bqueues** command to check if the scheduling policy for **high_io_queue** is set for global fair share scheduling in each cluster:

```

$ bqueues -rl high_io_queue
...
SCHEDULING POLICIES: GLOBAL_FAIRSHARE(high_io_policy)
USER_SHARES: [high_io_orgs, 100]

SHARE_INFO_FOR: high_io_queue/

```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME	ADJUST	REMOTE_LOAD
high_io_orgs	100	33.333	0	0	0.0	0	.000	0.000
...								

- User1 can submit a job to `high_io_queue` in `cluster1` and `cluster3`. After a few moments, run the `bgpinfo` command in any cluster to view the fair share resource usage from a global view:

```
$ bgpinfo fsload
```

```
POLICY_NAME: high_io_policy
```

```
SHARE_INFO_FOR: /
```

USER/GROUP	STARTED	RESERVED	CPU_TIME	RUN_TIME	HIST_RUN_TIME	ADJUST
high_io_orgs	2	0	0.0	220	0	0.000

```
SHARE_INFO_FOR: /high_io_orgs/
```

USER/GROUP	STARTED	RESERVED	CPU_TIME	RUN_TIME	HIST_RUN_TIME	ADJUST
org_a	2	0	0.0	220	0	0.000

```
SHARE_INFO_FOR: /high_io_orgs/org_a/
```

USER/GROUP	STARTED	RESERVED	CPU_TIME	RUN_TIME	HIST_RUN_TIME	ADJUST
project_a	2	0	0.0	220	0	0.000

```
SHARE_INFO_FOR: /high_io_orgs/org_a/project_a/
```

USER/GROUP	STARTED	RESERVED	CPU_TIME	RUN_TIME	HIST_RUN_TIME	ADJUST
user1	2	0	0.0	220	0	0.000

- Check the dynamic user priority of fair share accounts in `high_io_queue` in `cluster2`. You can see that the dynamic user priority for `user1` in `org_a`, `project_a` is low because `user1` has a high `REMOTE_LOAD` value, meaning that `user1` uses more resources than other users in remote clusters:

```
$ bqueues -r high_io_queue
```

```
QUEUE: high_io_queue
```

```
-- "A queue for high-IO projects"
```

```
...
```

```
SCHEDULING POLICIES: GLOBAL_FAIRSHARE(high_io_policy)
```

```
USER_SHARES: [high_io_orgs, 100]
```

```
SHARE_INFO_FOR: high_io_queue/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME	ADJUST
high_io_orgs	100	10.368	0	0	0.0	0	0.000

```
REMOTE_LOAD
```

```
6.645
```

```
SHARE_INFO_FOR: high_io_queue/high_io_orgs/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME	ADJUST
org_b	100	33.333	0	0	0.0	0	0.000

```
REMOTE_LOAD
```

```
0.000
```

org_a	100	10.368	0	0	0.0	0	0.000
-------	-----	--------	---	---	-----	---	-------

```
6.645
```

```
SHARE_INFO_FOR: high_io_queue/high_io_orgs/org_b/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME	ADJUST
project_c	100	33.333	0	0	0.0	0	0.000

```
REMOTE_LOAD
```

```
0.000
```

project_d	100	33.333	0	0	0.0	0	0.000
-----------	-----	--------	---	---	-----	---	-------

```
0.000
```

```
SHARE_INFO_FOR: high_io_queue/high_io_orgs/org_b/project_c/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME	ADJUST
user5	100	33.333	0	0	0.0	0	0.000

```
REMOTE_LOAD
```

```
0.000
```

user6	100	33.333	0	0	0.0	0	0.000
-------	-----	--------	---	---	-----	---	-------

```
0.000
```

```
SHARE_INFO_FOR: high_io_queue/high_io_orgs/org_b/project_d/
```

USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME	ADJUST
user1	100	10.368	0	0	0.0	0	0.000

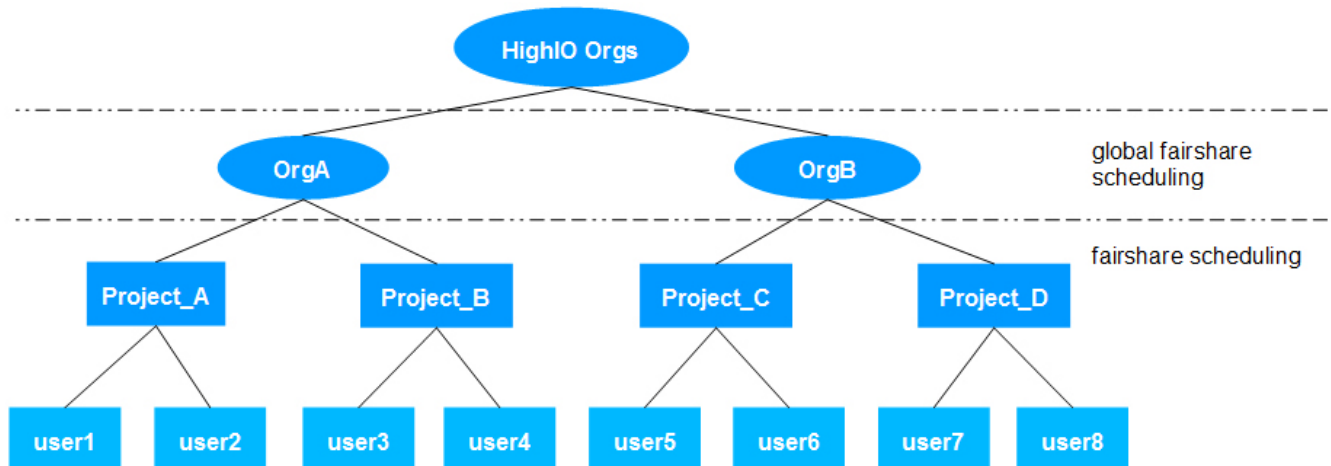
```
REMOTE_LOAD
```

user7	100	33.333	0	0	0.0	0	0.000
0.000							
user8	100	33.333	0	0	0.0	0	0.000
0.000							
SHARE_INFO_FOR: high_io_queue/high_io_orgs/org_a/							
USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME	ADJUST
REMOTE_LOAD							
project_b	100	33.333	0	0	0.0	0	0.000
0.000							
project_a	100	10.368	0	0	0.0	0	0.000
6.645							
SHARE_INFO_FOR: high_io_queue/high_io_orgs/org_a/project_b/							
USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME	ADJUST
REMOTE_LOAD							
user3	100	33.333	0	0	0.0	0	0.000
0.000							
user4	100	33.333	0	0	0.0	0	0.000
0.000							
SHARE_INFO_FOR: high_io_queue/high_io_orgs/org_a/project_a/							
USER/GROUP	SHARES	PRIORITY	STARTED	RESERVED	CPU_TIME	RUN_TIME	ADJUST
REMOTE_LOAD							
user2	100	33.333	0	0	0.0	0	0.000
0.000							
user1	100	10.368	0	0	0.0	0	0.000
6.645							

From a global perspective, the resource usage of all nodes in **HighIO Orgs** is fair for all three clusters. The following example describes how to apply global fair share scheduling for all nodes in a tree called **HighIO Orgs** among three clusters using queue level user-based fair share scheduling and partial-mode synchronization.

In this tree:

- There are three execution clusters (**cluster1**, **cluster2**, and **cluster3**) connected together in an LSF multicluster environment.
- **OrgA** and **OrgB** share the same resources in all execution clusters.
- Users in **OrgA** and **OrgB** can log in to all the execution clusters and submit jobs.



Global fair share scheduling is only applied to organization level nodes. Project nodes and user nodes will still use local fair share scheduling in each cluster.

1. Configure queue level user-based fair share in each cluster for each node under the **HighIO Orgs** tree, and configure FS_POLICY for those nodes in which global fair share scheduling will be used. Assign each node the same number of shares in `lsb.users` and `lsb.queues`.

In `lsb.users`:

```
Begin UserGroup
GROUP_NAME      GROUP_MEMBER      USER_SHARES      FS_POLICY
project_a       (user1 user2)      ([default,100])  ()
project_b       (user3 user4)      ([default,100])  ()
```

```

project_c      (user5 user6)      ([default,100])      ()
project_d      (user7 user8)      ([default,100])      ()
org_a          (project_a project_b) ([default,100])      ()
org_b          (project_c project_d) ([default,100]) (high_io_policy)
high_io_orgs   (org_a org_b)
End UserGroup:

```

In lsb. queues:

```

Begin Queue
QUEUE_NAME     = high_io_queue
...
FAIRSHARE      = USER_SHARES [[high_io_orgs, 100]]
End Queue

```

2. Choose one of the three clusters as the GPD cluster in which the **gpolicyd** daemon will run. Assuming that **cluster1** is the GPD cluster, configure the correct LSB_GPD_CLUSTER and LSB_GPD_PORT in lsf.conf for all clusters:

```

LSB_GPD_CLUSTER="cluster1"
LSB_GPD_PORT=8885

```

3. In cluster1, configure a global fair share policy in the configuration file \$LSF_ENVDIR/lsbatchd/cluster_name/lsb.globalpolicies:

```

Begin GlobalFairshare
NAME = high_io_policy
PARTICIPANTS = high_io_queue@cluster1 high_io_queue@cluster2 high_io_queue@cluster3
SYNC_MODE = partial
End GlobalFairshare

```

The global fair share policy **high_io_policy** includes **high_io_queue** of all the three clusters as its participants and its SYNC_MODE is partial. Only those nodes whose FS_POLICY is configured with **high_io_policy** in each **high_io_queue** of each cluster can share resource usage across clusters.

The global fair share policy **high_io_policy** includes **high_io_queue** for each of the three participating clusters. Its SYNC_MODE is partial so that only those nodes whose FS_POLICY is configured with **high_io_policy** in each **high_io_queue** for each cluster can share resource usage across clusters.

4. Start **cluster1**, **cluster2**, and **cluster3**.
Global fair share scheduling then takes effect for only **OrgA** and **OrgB**.

Configure cross-queue user-based global fair share

Cross-queue user-based fair share works with global fair share. Configure cross-queue user-based global fair share in the same way as queue-level user-based global fair share, except that you can configure only the parent queue as a participant of the global fair share policy. If you configure a child queue as a participant, the child queue never synchronizes with global fair share.

Global fair share scheduling constraints

When applying global fair share scheduling, there are factors to note.

Specifically, note the following:

- Live configuration: when a fair share tree is changed by **bconf**, and if that tree is participating in global fair share, the changes also take effect for global fair share. **bconf** will not support keyword FS_POLICY for object user in lsb.users.
- LSF/XL feature of LSF Advanced Edition: the execution clusters can be configured to use global fair share. However, the submission cluster does not take part in global fair share as it is not connected to **gpolicyd**.
- LSF multicluster capability job forward mode: a job submitted from the local cluster is executed in the remote cluster. If fair share is applied to both the local cluster and remote cluster, the job is only counted once, in the fair share policy of remote cluster. Therefore, global fair share only applies to multicluster job forward mode.

- LSF multicluster capability lease mode: a job submitted from the local cluster can run on a lease-in host. Such jobs will only take effect in the local cluster fair share policy. When global fair share is enabled, such jobs are counted in the local fair share load and synchronized with other clusters.
- Using **bconf** to remove users: **bconf** cannot remove users in USER_SHARES for a user group that has FS_POLICY defined in lsb.users. To enable **bconf** to remove these users, edit lsb.users to remove USER_SHARES and FS_POLICY from the user group, then run **badadmin reconfig** to apply the changes.

Manage LSF on EGO

The enterprise grid orchestrator capability (EGO) enables enterprise applications to benefit from sharing of resources across the enterprise grid. When LSF on EGO is configured, EGO serves as the central resource broker for LSF.

- [About IBM Spectrum LSF on EGO](#)
Use EGO to share a collection of distributed software and hardware resources on a computing infrastructure (cluster) as parts of a single virtual computer. EGO enhances the scalability, robustness, and reliability of LSF clusters.
- [LSF and EGO directory structure](#)
Learn about the purpose of each LSF and EGO sub-directory and whether they are writable or non-writable by LSF.
- [Configure LSF and EGO](#)
Learn about EGO configuration files for LSF daemon management and how to handle parameters in lsf.conf and ego.conf.
- [LSF features with EGO-enabled SLA scheduling \(Obsolete\)](#)
LSF with EGO-enabled SLA scheduling is no longer supported and is obsolete.
- [Logging and troubleshooting](#)
Learn about EGO log files and how to troubleshoot LSF on EGO.
- [Frequently asked questions](#)
Answers to basic deployment usage questions about EGO.

About IBM Spectrum LSF on EGO

Use EGO to share a collection of distributed software and hardware resources on a computing infrastructure (cluster) as parts of a single virtual computer. EGO enhances the scalability, robustness, and reliability of LSF clusters.

- *Scalability*—EGO enhances LSF scalability. Currently, the scheduler has to deal with a large number of jobs. EGO provides management functionality for multiple schedulers that co-exist in one environment. In LSF 10, although only a single instance of LSF is available on EGO, the foundation is established for greater scalability in follow-on releases that will allow multiple instances of LSF on EGO.
- *Robustness* — In previous releases, LSF functioned as both scheduler and resource manager. EGO decouples these functions, making the entire system more robust. EGO reduces or eliminates downtime for LSF users while resources are added or removed.
- *Reliability* — In situations where service is degraded due to noncritical failures such as **sbatchd** or **RES**, by default, LSF does not automatically restart the daemons. The EGO service controller (**egosc**) can monitor all LSF daemons and automatically restart them if they fail. Similarly, the EGO service controller can also monitor and restart other critical processes such as **lmgrd**.
- *Additional scheduling functionality* — EGO provides the foundation for EGO-enabled SLA, which provides LSF with additional and important scheduling functionality.
- *Centralized management and administration framework.*
- *Single reporting framework* — across various application heads built around EGO.

What is EGO?

EGO assesses the demands of competing business services (consumers) operating within a cluster and dynamically allocates resources so as to best meet a company's overriding business objectives. These objectives might include

- Reducing the time or the cost of providing key business services
- Maximizing the revenue generated by existing computing infrastructure

- Configuring, enforcing, and auditing service plans for multiple consumers
- Ensuring high availability and business continuity through disaster scenarios
- Simplifying IT management and reducing management costs
- Consolidating divergent and mixed computing resources into a single virtual infrastructure that can be shared transparently between many business users

EGO also provides a full suite of services to support and manage resource orchestration. These include cluster management, configuration and auditing of service-level plans, resource facilitation to provide fail-over if a management host goes down, monitoring and data distribution.

EGO is only sensitive to the *resource requirements* of business services; EGO has no knowledge of any run-time dynamic parameters that exist for them. This means that EGO does not interfere with how a business service chooses to use the resources it has been allocated.

How EGO works

IBM® Spectrum Computing products work in various ways to match business service (consumer) demands for resources with an available supply of resources. While a specific clustered application manager or consumer (for example, an LSF cluster) identifies what its resource demands are, EGO is responsible for supplying those resources. EGO determines the number of resources each consumer is entitled to, takes into account a consumer's priority and overall objectives, and then allocates the number of required resources (for example, the number of slots, virtual machines, or physical machines).

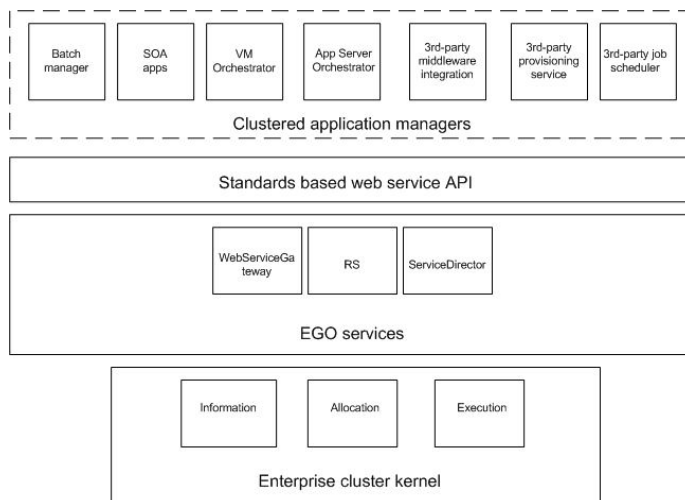
Once the consumer receives its allotted resources from EGO, the consumer applies its own rules and policies. How the consumer decides to balance its workload across the fixed resources allotted to it is not the responsibility of EGO.

So how does EGO know the demand? Administrators or developers use various EGO interfaces (such as the SDK or CLI) to tell EGO what constitutes a demand for more resources. When LSF identifies that there is a demand, it then distributes the required resources based on the resource plans given to it by the administrator or developer.

For all of this to happen smoothly, various components are built into EGO. Each EGO component performs a specific job.

EGO components

EGO comprises a collection of cluster orchestration software components. The following figure shows overall architecture and how these components fit within a larger system installation and interact with each other:



Key EGO concepts

Consumers

A consumer represents an entity that can demand resources from the cluster. A consumer might be a business service, a business process that is a complex collection of business services, an individual user, or an entire line of business.

EGO resources

Resources are physical and logical entities that can be requested by a client. For example, an application (client) requests a processor (resource) in order to run.

Resources also have attributes. For example, a host has attributes of memory, processor utilization, operating systems type, etc.

Resource distribution tree

The resource distribution tree identifies consumers of the cluster resources, and organizes them into a manageable structure.

Resource groups

Resource groups are logical groups of hosts. Resource groups provide a simple way of organizing and grouping resources (hosts) for convenience; instead of creating policies for individual resources, you can create and apply them to an entire group. Groups can be made of resources that satisfy a specific requirement in terms of OS, memory, swap space, CPU factor and so on, or that are explicitly listed by name.

Resource distribution plans

The resource distribution plan, or resource plan, defines how cluster resources are distributed among consumers. The plan takes into account the differences between consumers and their needs, resource properties, and various other policies concerning consumer rank and the allocation of resources.

The distribution priority is to satisfy each consumer's reserved ownership, then distribute remaining resources to consumers that have demand.

Services

A service is a self-contained, continuously running process that accepts one or more requests and returns one or more responses. Services may have multiple concurrent service instances running on multiple hosts. All EGO services are automatically enabled by default at installation.

Run **egosh** to check service status.

If EGO is disabled, the **egosh** command cannot find ego.conf or cannot contact **vemkd** (not started), and the following message is displayed:

```
You cannot run the egosh command because the administrator has  
chosen not to enable EGO in lsf.conf: LSF_ENABLE_EGO=N.
```

EGO user accounts

A user account is a system user who can be assigned to any role for any consumer in the tree. User accounts include optional contact information, a name, and a password.

LSF and EGO directory structure

Learn about the purpose of each LSF and EGO sub-directory and whether they are writable or non-writable by LSF.

Directories under LSF_TOP

Directory Path	Description	Attribute
LSF_TOP/10.1.0	LSF 10.1.0 binaries and other machine dependent files	Non-writable
LSF_TOP/conf	LSF 10.1.0 configuration files You must be LSF administrator or root to edit files in this directory	Writable by the LSF administrator, management host, and management candidate hosts
LSF_TOP/log	LSF 10.1.0 log files	Writable by all hosts in the cluster
LSF_TOP/work	LSF 10.1.0 working directory	Writable by the management host and management candidate hosts, and is accessible to server hosts

EGO directories

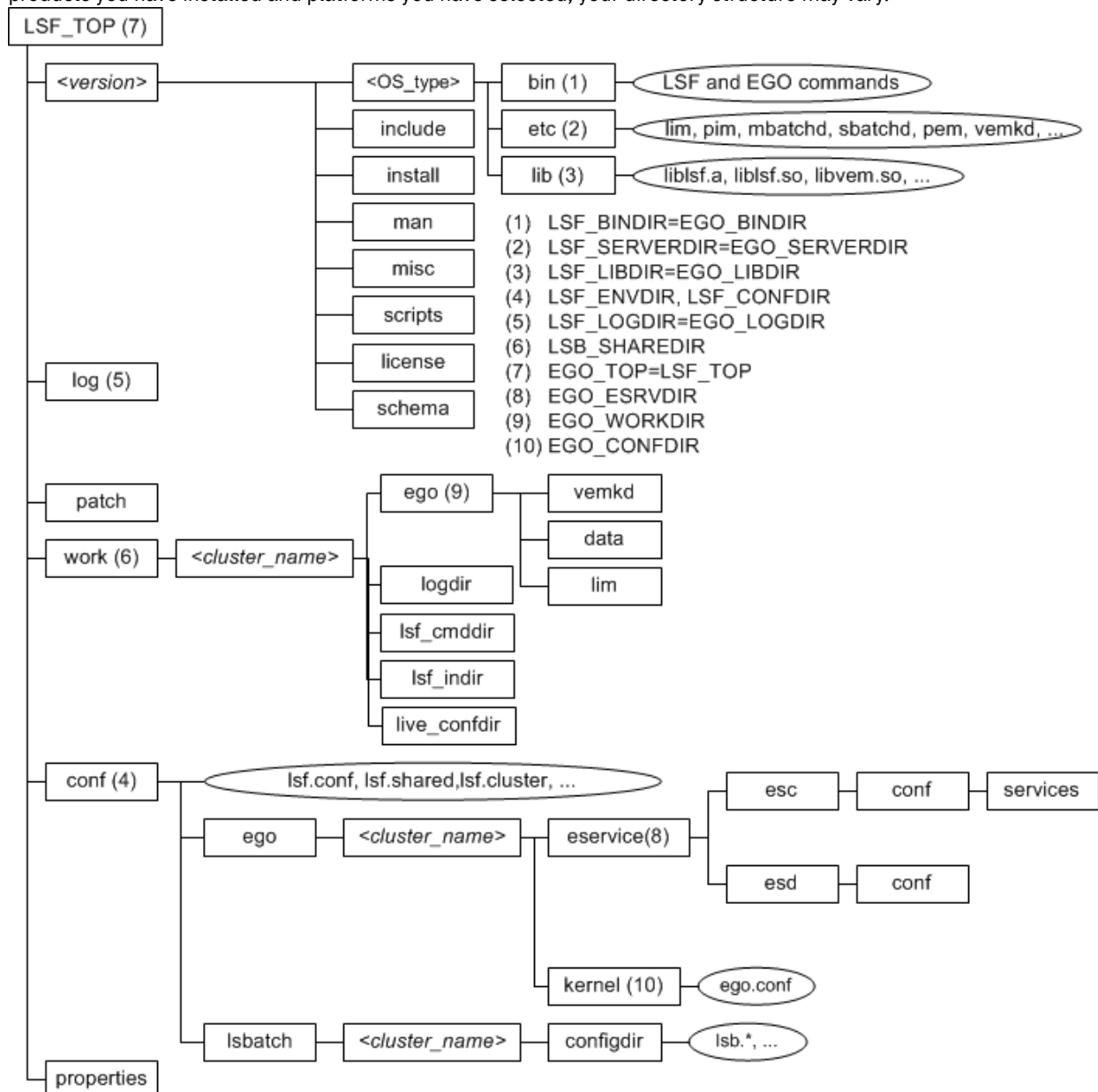
Directory Path	Description	Attribute
----------------	-------------	-----------

Directory Path	Description	Attribute
LSF_BINDIR	EGO binaries and other machine dependent files	Non-writable
LSF_CONFDIR/ego/cluster_name/eservice (EGO_ESRVDIR)	EGO services configuration and log files.	Writable
LSF_CONFDIR/ego/cluster_name/kernel (EGO_CONFDIR, LSF_EGO_ENVDIR)	EGO kernel configuration, log files and working directory, including conf/log/work	Writable
LSB_SHARED/cluster_name/ego (EGO_WORKDIR)	EGO working directory	Writable

Example directory structures

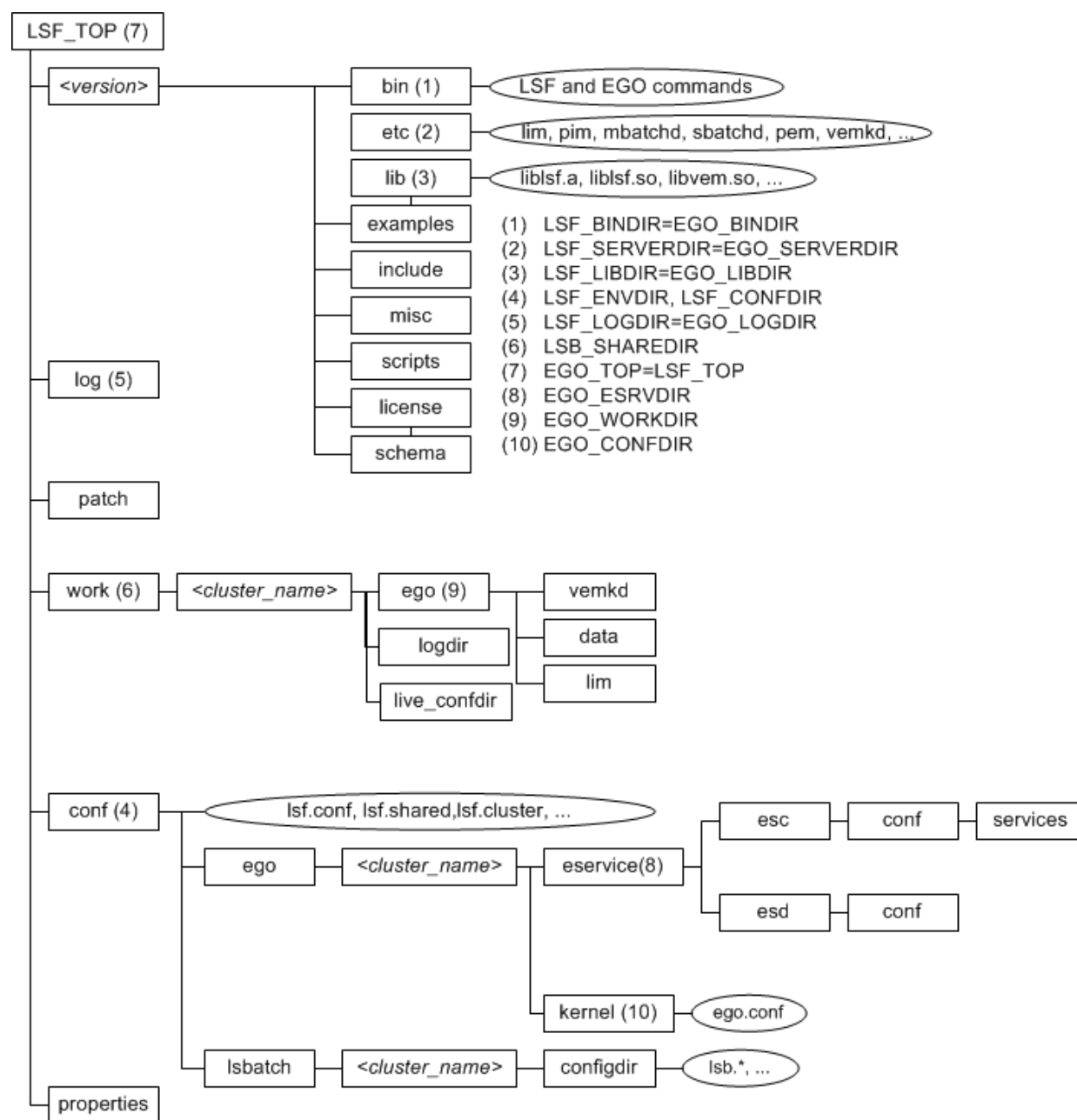
UNIX and Linux

The following figures show typical directory structures for a new UNIX or Linux installation with **lsfinstall**. Depending on which products you have installed and platforms you have selected, your directory structure may vary.



Microsoft Windows

The following diagram shows an example directory structure for a Windows installation.



Configure LSF and EGO

Learn about EGO configuration files for LSF daemon management and how to handle parameters in lsf.conf and ego.conf.

EGO configuration files for LSF daemon management (res.xml and sbatchd.xml)

The following files are located in EGO_ESRVDIR/esc/conf/services/:

- res.xml—EGO service configuration file for **res**.
- sbatchd.xml—EGO service configuration file for **sbatchd**.

When LSF daemon control through EGO Service Controller is configured, **lsadmin** uses the reserved EGO service name **res** to control the LSF **res** daemon, and **badmin** uses the reserved EGO service name **sbatchd** to control the LSF **sbatchd** daemon.

How to handle parameters in lsf.conf with corresponding parameters in ego.conf

In lsf.conf, LSF parameter names begin with LSB_ or LSF_. In ego.conf, EGO parameter names begin with EGO_. When EGO is enabled, existing LSF parameters that are set only in lsf.conf operate as usual because LSF daemons and commands read both lsf.conf and ego.conf.

Some existing LSF parameters have corresponding EGO parameter names in ego.conf (LSF_CONFDIR/lsf.conf is a separate file from LSF_CONFDIR/ego/cluster_name/kernel/ego.conf). You can keep your existing LSF parameters in lsf.conf, or you can set the corresponding EGO parameters in ego.conf that have not already been set in lsf.conf.

You cannot set LSF parameters in ego.conf, but you can set the following EGO parameters related to LIM, PIM, and ELIM in either lsf.conf or ego.conf:

- EGO_DAEMONS_CPUS
- EGO_DEFINE_NCPUS
- EGO_SLAVE_CTRL_REMOTE_HOST
- EGO_WORKDIR
- EGO_PIM_SWAP_REPORT

You cannot set any other EGO parameters in lsf.conf. If EGO is not enabled, you can only set these parameters in lsf.conf.

Note:

If you specify a parameter in lsf.conf and you also specify the corresponding parameter in ego.conf, the parameter value in ego.conf takes precedence over the conflicting parameter in lsf.conf.

If the parameter is not set in either lsf.conf or ego.conf, the default takes effect depends on whether EGO is enabled. If EGO is not enabled, then the LSF default takes effect. If EGO is enabled, the EGO default takes effect. In most cases, the default is the same.

Some parameters in lsf.conf do not have exactly the same behavior, valid values, syntax, or default value as the corresponding parameter in ego.conf, so in general, you should not set them in both files. If you need LSF parameters for backwards compatibility, you should set them only in lsf.conf.

If you have LSF 6.2 hosts in your cluster, they can only read lsf.conf, so you must set LSF parameters only in lsf.conf.

- [LSF and EGO corresponding parameters](#)
- [Parameters that have changed in LSF 10](#)
- [Special resource groups for LSF management hosts](#)
- [Manage LSF daemons through EGO](#)
- [Set the command-line environment](#)

LSF and EGO corresponding parameters

The following table summarizes existing LSF parameters that have corresponding EGO parameter names. You must continue to set other LSF parameters in lsf.conf.

lsf.conf parameter	ego.conf parameter
LSF_API_CONNTIMEOUT	EGO_LIM_CONNTIMEOUT
LSF_API_RECVTIMEOUT	EGO_LIM_RECVTIMEOUT
LSF_CLUSTER_ID (Windows)	EGO_CLUSTER_ID (Windows)
LSF_CONF_RETRY_INT	EGO_CONF_RETRY_INT
LSF_CONF_RETRY_MAX	EGO_CONF_RETRY_MAX
LSF_DEBUG_LIM	EGO_DEBUG_LIM

lsf.conf parameter	ego.conf parameter
LSF_DHPC_ENV	EGO_DHPC_ENV
LSF_DYNAMIC_HOST_TIMEOUT	EGO_DYNAMIC_HOST_TIMEOUT
LSF_DYNAMIC_HOST_WAIT_TIME	EGO_DYNAMIC_HOST_WAIT_TIME
LSF_ENABLE_DUALCORE	EGO_ENABLE_DUALCORE
LSF_GET_CONF	EGO_GET_CONF
LSF_GETCONF_MAX	EGO_GETCONF_MAX
LSF_LIM_DEBUG	EGO_LIM_DEBUG
LSF_LIM_PORT	EGO_LIM_PORT
LSF_LOCAL_RESOURCES	EGO_LOCAL_RESOURCES
LSF_LOG_MASK	EGO_LOG_MASK
LSF_MASTER_LIST	EGO_MASTER_LIST
LSF_PIM_INFODIR	EGO_PIM_INFODIR
LSF_PIM_SLEEPTIME	EGO_PIM_SLEEPTIME
LSF_PIM_SLEEPTIME_UPDATE	EGO_PIM_SLEEPTIME_UPDATE
LSF_RSH	EGO_RSH
LSF_STRIP_DOMAIN	EGO_STRIP_DOMAIN
LSF_TIME_LIM	EGO_TIME_LIM

Parameters that have changed in LSF 10

The default for `LSF_LIM_PORT` has changed to accommodate EGO default port configuration. On EGO, default ports start with `lim` at 7869, and are numbered consecutively for **pem**, **vemkd**, and **egosc**.

This is different from previous LSF releases where the default `LSF_LIM_PORT` was 6879. **res**, **sbatchd**, and **mbatchd** continue to use the default pre-version 7 ports 6878, 6881, and 6882.

Upgrade installation preserves any existing port settings for **lim**, **res**, **sbatchd**, and **mbatchd**. EGO **pem**, **vemkd**, and **egosc** use default EGO ports starting at 7870, if they do not conflict with existing **lim**, **res**, **sbatchd**, and **mbatchd** ports.

EGO connection ports and base port

LSF and EGO require exclusive use of certain ports for communication. EGO uses the same four consecutive ports on every host in the cluster. The first of these is called the base port.

The default EGO base connection port is 7869. By default, EGO uses four consecutive ports starting from the base port. By default, EGO uses ports 7869-7872.

The ports can be customized by customizing the base port. For example, if the base port is 6880, EGO uses ports 6880-6883.

LSF and EGO needs the same ports on every host, so you must specify the same base port on every host.

Special resource groups for LSF management hosts

By default, IBM® Spectrum LSF installation defines a special resource group named `ManagementHosts` for the IBM Spectrum LSF management host. (In general, IBM Spectrum LSF management hosts are dedicated hosts; the `ManagementHosts` EGO resource group serves this purpose.)

IBM Spectrum LSF management hosts must not be subject to any lend, borrow, or reclaim policies. They must be exclusively owned by the IBM Spectrum LSF consumer.

The default EGO configuration is such that the `LSF_MASTER_LIST` hosts and the execution hosts are in different resource groups so that different resource plans can be applied to each group.

Manage LSF daemons through EGO

EGO daemons

Daemons in LSF_SERVERDIR	Description
vemkd	Started by lim on management host
pem	Started by lim on every host
egosc	Started by vemkd on management host

LSF daemons

Daemons in LSF_SERVERDIR	Description
lim	lim runs on every host. On UNIX, lim is either started by lsadmin through rsh/ssh or started through rc file. On Windows, lim is started as a Windows service.
pim	Started by lim on every host
mbatchd	Started by sbatchd on management host
mbschd	Started by mbatchd on management host
sbatchd	Under OS startup mode, sbatchd is either started by lsadmin through rsh/ssh or started through rc file on UNIX. On Windows, sbatchd is started as a Windows service. Under EGO Service Controller mode, sbatchd is started by pem as an EGO service on every host.
res	Under OS startup mode, res is either started by lsadmin through rsh/ssh or started through rc file on UNIX. On Windows, res is started as a Windows service. Under EGO Service Controller mode, res is started by pem as an EGO service on every host.

Operating System daemon control

Operating system startup mode is the same as previous releases:

- On UNIX, administrators configure the autostart of **sbatchd** and **res** in the operating system (/etc/rc file or inittab) and use **lsadmin** and **badmin** to start LSF daemons manually through **rsh** or **ssh**.
- On Windows, **sbatchd** and **res** are started as Windows services.

EGO Service Controller daemon control

Under EGO Service Control mode, administrators configure the EGO Service Controller to start **res** and **sbatchd**, and restart them if they fail.

You can still run **lsadmin** and **badmin** to start LSF manually, but internally, **lsadmin** and **badmin** communicates with the EGO Service Controller, which actually starts **sbatchd** and **res** as EGO services.

If EGO Service Controller management is configured and you run **bctrl stop sbd** and **bctrl stop res** to manually shut down LSF, the LSF daemons are not restarted automatically by EGO. You must run **bctrl start res** and **bctrl start sbd** to start the LSF daemons manually.

Permissions required for daemon control

To control all daemons in the cluster, you must

- Be logged on as root or as a user listed in the /etc/lsf.sudoers file. See the *LSF Configuration Reference* for configuration details of lsf.sudoers.

- Be able to run the `rsh` or `ssh` commands across all LSF hosts without having to enter a password. See your operating system documentation for information about configuring the `rsh` and `ssh` commands. The shell command specified by `LSF_RSH` in `lsf.conf` is used before `rsh` is tried.
- [Bypass EGO login at startup \(lsf.sudoers\)](#)

Bypass EGO login at startup (lsf.sudoers)

Before you begin

To use the `lsf.sudoers` file, you must be the LSF administrator (`lsfadmin`) or `root`, and you must enable the `setuid` bit for the LSF administration commands. Run the `hostsetup --setuid` command option on the LSF management and candidate hosts to enable the `setuid` bit. Since this allows daemons to run with root privileges, do not enable the `setuid` bit if you do not want LSF daemons to run with root privileges.

The `hostsetup --setuid` command enables the `setuid` bit for the following LSF executable files: **badadmin**, **lsadmin**, **egosh**, **utmpreg**, **swtbl_api**, **ntbl_api**, **lstbl_nid**, and **swtbl_poe**.

About this task

When LSF daemons control through EGO Service Controller is configured, users must have EGO credentials for EGO to start **res** and **sbatchd** services. By default, **lsadmin** and **badadmin** invoke the `egosh user login` command to prompt for the user name and password of the EGO administrator to get EGO credentials.

Procedure

Configure `lsf.sudoers` to bypass EGO login to start **res** and **sbatchd** automatically.
Set the following parameters:

- `LSF_EGO_ADMIN_USER`—User name of the EGO administrator. The default administrator name is `Admin`.
- `LSF_EGO_ADMIN_PASSWD`—Password of the EGO administrator.

Set the command-line environment

About this task

On Linux hosts, set the environment before you run any LSF or EGO commands. You need to do this once for each session you open. `root`, `lsfadmin`, and `egoadmin` accounts use LSF and EGO commands to configure and start the cluster.

You need to reset the environment if the environment changes during your session, for example, if you run `egoconfig mghost`, which changes the location of some configuration files.

Procedure

- For **csh** or **tcsh**, use `cshrc.lsf`.
`source LSF_TOP/conf/cshrc.lsf`
- For **sh**, **ksh**, or **bash**, use `profile.lsf`:
`. LSF_TOP/conf/profile.lsf`

Results

If enterprise grid orchestrator is enabled in the LSF cluster (LSF_ENABLE_EGO=Y and LSF_EGO_ENVDIR are defined in lsf.conf), cshrc.lsf and profile.lsf, set the following environment variables:

- EGO_BINDIR
- EGO_CONFDIR
- EGO_ESRVDIR
- EGO_LIBDIR
- EGO_LOCAL_CONFDIR
- EGO_SERVERDIR
- EGO_TOP

See the *enterprise grid orchestrator Reference* for more information about these variables.

See the *LSF Configuration Reference* for more information about cshrc.lsf and profile.lsf.

LSF features with EGO-enabled SLA scheduling (Obsolete)

LSF with EGO-enabled SLA scheduling is no longer supported and is obsolete.

Several LSF features are tested for EGO-enabled SLA scheduling and might be fully supported or require further configuration to work effectively.

The following LSF features are supported on EGO:

- Job arrays
- Job dependencies
- Queue-level user-based fair share
- Parallel jobs
- Slot reservation for parallel jobs
- [Supported LSF features with EGO-enabled SLA scheduling \(Obsolete\)](#)
LSF with EGO-enabled SLA scheduling is no longer supported and is obsolete.
- [LSF features that require modification to work with EGO-enabled SLA scheduling \(Obsolete\)](#)
LSF with EGO-enabled SLA scheduling is no longer supported and is obsolete.
- [Unsupported LSF features with EGO-enabled SLA scheduling \(Obsolete\)](#)
LSF with EGO-enabled SLA scheduling is no longer supported and is obsolete.

Supported LSF features with EGO-enabled SLA scheduling (Obsolete)

LSF with EGO-enabled SLA scheduling is no longer supported and is obsolete.

The following LSF features are fully supported with EGO-enabled SLA scheduling (that is, when ENABLE_DEFAULT_EGO_SLA=Y is defined in the lsb.params file).

Job arrays

LSF on EGO supports the submission of job arrays (**bsub -J**).

```
bsub -J "array1[1-10]" myjob1
```

Job dependencies

LSF on EGO supports job dependency scheduling (**bsub -w**).

For example,

```
bsub myjob1
Job <1090> is submitted to default queue <normal>.
bsub -w "done(1090)" myjob2
bsub -J "array1[1-10]" myjob1
bsub -w "ended(array1[*])" -J "array2[1-10]" myjob2
```

Queue-level user-based fair share

LSF on EGO supports queue-level user-based fair share policies. You can configure a user-based fair share queue by defining fair share in `lsb.queues` and specifying a share assignment for all users of the queue (`USER_SHARES`), then submit jobs to the queue in an EGO-enabled LSF service class (`bsub -sla service_class -q queue_name`).

For example, if the EGO-enabled service class is `LSF_Normal`, and the queue with user-based fair share policies enabled is `license`:

```
bsub -sla LSF_Normal -q license -J "array1[1-10]" myjob
bsub -sla LSF_Normal -q license -J "array2[1-10]" myjob
```

LSF features that require modification to work with EGO-enabled SLA scheduling (Obsolete)

LSF with EGO-enabled SLA scheduling is no longer supported and is obsolete.

The following LSF features require modification to work properly with EGO-enabled SLA scheduling (that is, when `ENABLE_DEFAULT_EGO_SLA=Y` is defined in the `lsb.params` file).

Parallel jobs

LSF dynamically gets job sizes (number of tasks) from EGO based on either the velocity or the total number of pending and running jobs in a service class, whichever is larger. Therefore, if the number of pending and running jobs in a service class is small, LSF requests only the velocity as configured in the service class. However, if the velocity is smaller than the number of tasks that are required by a parallel job (as requested by using the **bsub -n** option), the job pends indefinitely.

To prevent the parallel job from pending indefinitely, set a velocity goal to a higher value than the job size required by the parallel job so that any parallel jobs in the service class are scheduled instead of pending indefinitely. For more information about setting velocity goals, see [Service classes for SLA scheduling](#).

Job size reservation for a parallel job

Configure job size (number of tasks) reservation in a queue by defining `SLOT_RESERVE=MAX_RESERVE_TIME[integer]` in `lsb.queues`. LSF reserves the job size for a large parallel job without being starved by other jobs that require a smaller job size than the large parallel job.

For example, if the service class for parallel jobs is `LSF_Parallel`, and the queue with job size reservation configured for parallel jobs `Parallel_Reserve`,

```
bsub -sla LSF_Parallel -J "array1[1-10]" myjob
bsub -sla LSF_Parallel -q Parallel_Reserve -n 4 myjob
bsub -sla LSF_Parallel -J "array2[1-10]" myjob
```

Resource requirements

A job level resource requirement (specified by using **bsub -R**) is not passed from LSF to EGO when you request job sizes. Resource requirements are only passed from LSF at the LSF service class or EGO consumer level.

Ensure all jobs that are submitted to the LSF service class can run on the job slots or hosts that are allocated by EGO according to the resource requirement in the service class or the corresponding EGO consumer.

Resource preemption

Use EGO resource reclaim between consumers according to the resource sharing plans for the resource preemption between jobs. When a slot is reclaimed by EGO according to the resource sharing plan, the job that is running on the slot can be killed or queued in LSF so that the job slot can be used by other high priority workload.

LSF parallel job consumers

Do not configure a consumer of large LSF parallel jobs to borrow slots from other EGO consumers because a job that is running on a job slot are killed and if the job slot is reclaimed by EGO.

Configure the LSF parallel job consumer to own job slots, then lend the slots to other consumers that have small impact if their workload is preempted.

Unsupported LSF features with EGO-enabled SLA scheduling (Obsolete)

LSF with EGO-enabled SLA scheduling is no longer supported and is obsolete.

The following LSF features are not supported with EGO-enabled SLA scheduling (that is, when `ENABLE_DEFAULT_EGO_SLA=Y` is defined in the `lsb.params` file).

Most of the LSF features that are not supported with EGO-enabled SLA scheduling are related to hosts or host lists that must be specified in configuration files or on the command line. These features are not supported because hosts and job slots in LSF with EGO-enabled SLA scheduling are all dynamically allocated on demand. LSF cannot request specific hosts in these cases.

- Resource limits on hosts or host groups
- Advance reservation on hosts or host groups
- Guaranteed resource pool
- Compute unit
- Host partition
- User-based fair share at the LSF service class or host partition level
- Any configuration or job specification where a list of hosts or host groups can be specified, such as queues, host groups, or **bsub -m** (run the job on one of the specified hosts or host groups)
- Resizable parallel jobs
- `RES_REQ` in a queue, application profile, or **bsub -R** (run the job on a host that meets the specified resource requirements)
- Guaranteed service level agreements (SLAs)
- IBM® Spectrum LSF multicluster capability

Logging and troubleshooting

Learn about EGO log files and how to troubleshoot LSF on EGO.

- [EGO log files](#)
Log files contain important runtime information about the general health of EGO daemons, workload submissions, and other EGO system events. Log files are an essential troubleshooting tool during production and testing.

- [Troubleshoot using multiple EGO log files](#)

If a service does not start as expected, open the appropriate service log file and review the runtime information that is contained within it to discover the problem. Look for relevant entries such as insufficient disk space, lack of memory, or network problems that result in unavailable hosts.

EGO log files

Log files contain important runtime information about the general health of EGO daemons, workload submissions, and other EGO system events. Log files are an essential troubleshooting tool during production and testing.

The naming convention for most EGO log files is the name of the daemon plus the host name the daemon is running on.

The following table outlines the daemons and their associated log file names. Log files on Windows hosts have a .txt extension.

Daemon	Log file name
ESC (EGO service controller)	esc.log. <i>hostname</i>
named	named.log. <i>hostname</i>
PEM (Process Execution Manager)	pem.log. <i>hostname</i>
VEMKD (enterprise grid orchestrator kernel daemon)	vemkd.log. <i>hostname</i>
WSG (web service gateway)	wsg.log

Most log entries are informational in nature. It is not uncommon to have a large (and growing) log file and still have a healthy cluster.

EGO log file locations

By default, most IBM® Spectrum LSF log files are found in *LSF_LOGDIR*.

- The service controller log files are found in *LSF_LOGDIR/ego/cluster_name/eservice/esc/log* (Linux) or *LSF_LOGDIR\ego\cluster_name\eservice\esc\log* (Windows).
- Web service gateway log files are found in the following locations:
 - On UNIX and Linux: *LSF_LOGDIR/ego/cluster_name/eservice/wsg/log*
 - On Windows: *LSF_LOGDIR\ego\cluster_name\eservice\wsg\log*
- The service directory log files, logged by BIND, are found in the following locations:
 - On UNIX and Linux: *LSF_LOGDIR/ego/cluster_name/eservice/esd/conf/named/namedb/named.log.hostname*
 - On Windows: *LSF_LOGDIR\ego\cluster_name\eservice\esd\conf\named\namedb\named.log.hostname*

EGO log entry format

Log file entries have the following format

```
date time_zone log_level [process_id:thread_id] action:description/message
```

where the *date* is expressed in *YYYY-MM-DD hh-mm-ss.sss*.

For example,

```
2006-03-14 11:02:44.000 Eastern Standard Time ERROR [2488:1036] vemkdexit: vemkd is halting
```

EGO log classes

Every log entry belongs to a log class. You can use log class as a mechanism to filter log entries by area. Log classes in combination with log levels allow you to troubleshoot using log entries that only address, for example, configuration.

Use **egosh debug** to adjust log classes at run time.

Valid logging classes are as follows:

Class	Description
LC_ALLOC	Logs messages related to the resource allocation engine
LC_AUTH	Logs messages related to users and authentication
LC_CLIENT	Logs messages related to clients
LC_COMM	Logs messages related to communications
LC_CONF	Logs messages related to configuration
LC_CONTAINER	Logs messages related to activities
LC_EVENT	Logs messages related to the event notification service
LC_MEM	Logs messages related to memory allocation
LC_PEM	Logs messages related to the process execution manager (pem)
LC_PERF	Logs messages related to performance
LC_QUERY	Logs messages related to client queries
LC_RECOVER	Logs messages related to recovery and data persistence
LC_RSRC	Logs messages related to resources, including host status changes
LC_SYS	Logs messages related to system calls
LC_TRACE	Logs the steps of the program

EGO log levels

There are nine log levels that allow administrators to control the level of event information that is logged.

When you are troubleshooting, increase the log level to obtain as much detailed information as you can. When you are finished troubleshooting, decrease the log level to prevent the log files from becoming too large.

Valid logging levels are as follows:

Number	Level	Description
0	LOG_EMERG	Log only those messages in which the system is unusable.
1	LOG_ALERT	Log only those messages for which action must be taken immediately.
2	LOG_CRIT	Log only those messages that are critical.
3	LOG_ERR	Log only those messages that indicate error conditions.
4	LOG_WARNING	Log only those messages that are warnings or more serious messages. This is the default level of debug information.
5	LOG_NOTICE	Log those messages that indicate normal but significant conditions or warnings and more serious messages.
6	LOG_INFO	Log all informational messages and more serious messages.
7	LOG_DEBUG	Log all debug-level messages.
8	LOG_TRACE	Log all available messages.

EGO log level and class information retrieved from configuration files

When EGO is enabled, the **pem** and **vemkd** daemons read the `ego.conf` file to retrieve the following information (as corresponds to the particular daemon):

- EGO_LOG_MASK: The log level used to determine the amount of detail logged.
- EGO_DEBUG_PEM: The log class setting for pem.
- EGO_DEBUG_VEMKD: The log class setting for vemkd.

The **wsg** daemon reads the `wsg.conf` file to retrieve the following information:

- WSG_PORT: The port on which the web service gateway runs.
- WSG_SSL: Whether the daemon should use Secure Socket Layer (SSL) for communication.

- WSG_DEBUG_DETAIL: The log level used to determine the amount of detail logged for debugging purposes.
- WSG_LOGDIR: The directory location where the wsg.log files are written.

The service director daemon (**named**) reads named.conf to retrieve the log class and severity information. The configured severity log class controlling the level of event information that is logged (critical, error, warning, notice, info, debug, or dynamic). In the case of a log class set to debug, a log level is required to determine the amount of detail logged for debugging purposes.

Why do log files grow so quickly?

Every time an EGO system event occurs, a log file entry is added to a log file. Most entries are informational in nature, except when there is an error condition. If your log levels provide entries for all information (for example, if you have set them to LOG_DEBUG), the files will grow quickly.

The following are suggested settings:

- During regular EGO operation, set your log levels to LOG_WARNING. With this setting, critical errors are logged but informational entries are not, keeping the log file size to a minimum.
- For troubleshooting purposes, set your log level to LOG_DEBUG. Because of the quantity of messages you will receive when subscribed to this log level, change the level back to LOG_WARNING as soon as you are finished troubleshooting.

Note: If your log files are too long, you can always rename them for archive purposes. New, fresh log files will then be created and will log all new events.

How often should I maintain log files?

The growth rate of the log files is dependent on the log level and the complexity of your cluster. If you have a large cluster, daily log file maintenance may be required.

You should use a log file rotation utility to do unattended maintenance of your log files. Failure to do timely maintenance could result in a full file system which hinders system performance and operation.

Troubleshoot using multiple EGO log files

If a service does not start as expected, open the appropriate service log file and review the runtime information that is contained within it to discover the problem. Look for relevant entries such as insufficient disk space, lack of memory, or network problems that result in unavailable hosts.

EGO log file locations and content

Log file	Default location	What it contains
esc.log	<ul style="list-style-type: none"> • Linux: LSF_LOGDIR/ esc.log.<i>host_name</i> • Windows: LSF_LOGDIR\ esc.log.<i>host_name</i> 	Logs service failures and service instance restarts based on availability plans.

Log file	Default location	What it contains
named.log	<ul style="list-style-type: none"> Linux: LSF_LOGDIR/named.log.<i>host_name</i> Windows: LSF_LOGDIR\named.log.<i>host_name</i> 	Logs information that is gathered during the updating and querying of service instance location; logged by BIND, a DNS server.
pem.log	<ul style="list-style-type: none"> Linux: LSF_LOGDIR/pem.log.<i>host_name</i> Windows: LSF_LOGDIR\pem.log.<i>host_name</i> 	Logs remote operations (start, stop, control activities, failures). Logs tracked results for resource usage of all processes that are associated with the host, and information for accounting or chargeback.
vemkd.log	<ul style="list-style-type: none"> Linux: LSF_LOGDIR/vemkd.log.<i>host_name</i> Windows: LSF_LOGDIR\vemkd.log.<i>host_name</i> 	Logs aggregated host information about the state of individual resources, status of allocation requests, consumer hierarchy, resource assignment to consumers, and started operating system-level processes.
wsg.log	<ul style="list-style-type: none"> Linux: LSF_LOGDIR/wsg.log.<i>host_name</i> Windows: LSF_LOGDIR\wsg.log.<i>host_name</i> 	Logs service failures in web services interfaces for web service client applications.

Match service error messages and corresponding log files

Message	Problem	Review this log file
failed to create vem working directory	Cannot create work directory during startup	vemkd.log
failed to open lock file	Cannot get lock file during startup	vemkd.log
failed to open host event file	Cannot recover during startup because event file cannot be opened	vemkd.log
lim port is not defined	EGO_LIM_PORT in ego.conf is not defined	lim.log
management candidate can not set GET_CONF=lim	Wrong parameter value for management candidate host (for example, EGO_GET_CONF=LIM)	lim.log
there is no valid host in EGO_MASTER_LIST	No valid host in the management host list	lim.log
ls_getmyhostname fails	Cannot get local host name during startup	pem.log
temp directory (%s) not exist or not accessible, exit	Temporary directory does not exist	pem.log
incorrect EGO_PEM_PORT value %s, exit	EGO_PEM_PORT is a negative number	pem.log
chdir(%s) fails	Temporary directory does not exist	esc.log
cannot initialize the listening TCP port %d	Socket error	esc.log

Message	Problem	Review this log file
cannot log on	Log on to vemkd failed	esc.log
vem_register: error in invoking vem_register function	VEM service registration failed	wsg.log
you are not authorized to unregister a service	Either you are not authorized to unregister a service, or no registry client exists	wsg.log
request has invalid signature: TSIG service.ego: tsig verify failure (BADTIME)	Failed to update a resource record	named.log

Frequently asked questions

Answers to basic deployment usage questions about EGO.

Question

Does LSF 10 on EGO support a grace period when reclamation is configured in the resource plan?

Answer

No. Resources are immediately reclaimed even if you set a resource reclaim grace period.

Question

Does LSF 10 on EGO support upgrade of the management host only?

Answer

Yes

Question

*Under EGO service controller daemon management mode on Windows, does PEM start sbatchd and res directly or does it ask Windows to start **sbatchd** and RES as Windows Services?*

Answer

On Windows, LSF still installs **sbatchd** and RES as Windows services. If EGO service controller daemon control is selected during installation, the Windows service will be set up as Manual. PEM will start up the **sbatchd** and RES directly, not as Windows Services.

Question

What's the benefit of LSF daemon management through the EGO service controller?

Answer

The EGO service controller provides high availability services to **sbatchd** and RES, and faster cluster startup than startup with **lsadmin** and **badmin**.

Question

*How does the **hostsetup** script work in LSF 10?*

Answer

LSF 10 **hostsetup** script functions essentially the same as previous versions. It sets up a host to use the LSF cluster and configures LSF daemons to start automatically. In LSF 10, running **hostsetup --top=/path --boot="y"** will check the EGO service definition files sbatchd.xml and res.xml. If **res** and **sbatchd** startup is set to "Automatic", the host rc setting will only start **lim**. If set to "Manual", the host rc setting will start **lim**, **sbatchd**, and **res** as in previous versions.

Question

Is non-shared mixed cluster installation supported, for example, adding UNIX hosts to a Windows cluster, or adding Windows hosts to a UNIX cluster?

Answer

In LSF 10, non-shared installation is supported. For example, to add a UNIX host to a Windows cluster, set up the Windows cluster first, then run **lsfinstall -s -f server.config**. In server.config, put the Windows hosts in LSF_MASTER_LIST. After startup, the UNIX host will become an LSF host. Adding a Windows host is even simpler. Run the Windows installer, enter the current UNIX management host name. After installation, all daemons will automatically start and the host will join the cluster.

Question

As EGO and LSF share base configuration files, how are other resources handled in EGO in addition to hosts and slots?

Answer

Same as previous releases. LSF 10 **mbatchd** still communicates with LIM to get available resources. By default, LSF can schedule jobs to make use of all resources started in cluster. If EGO-enabled SLA scheduling is configured, LSF only schedules jobs to use resources on hosts allocated by EGO.

Question

How about compatibility for external scripts and resources like elim, melim, esub and others?

Answer

LSF 10 supports full compatibility for these external executables. elim.xxx is started under LSF_SERVERDIR as usual. By default, LIM is located under LSF_SERVERDIR.

Question

Can IBM® Spectrum LSF multicluster capability share one EGO base?

Answer

No, each LSF cluster must run on top of one EGO cluster.

Question

Can EGO consumer policies replace MultiCluster lease mode?

Answer

Conceptually, both define resource borrowing and lending policies. However, current EGO consumer policies can only work with slot resources within one EGO cluster. IBM Spectrum LSF multicluster capability lease mode supports other load indices and external resources between multiple clusters. If you are using LSF multicluster capability lease mode to share only slot resources between clusters, and you are able to merge those clusters into a single cluster, you should be able to use EGO consumer policy and submit jobs to EGO-enabled SLA scheduling to achieve the same goal.

Load sharing X applications

- [Start an xterm](#)
- [xterm on a PC](#)
- [Set up Exceed to log on the least loaded host](#)
- [Start an xterm in Exceed](#)
- [Examples](#)

Start an xterm

Procedure

If you are using the X Window System, you can start an xterm that opens a shell session on the least loaded host by entering:

```
lsrun sh -c xterm &
```

The `&` in this command line is important as it frees resources on the host once xterm is running, by running the X terminal in the background.

In this example, no processes are left running on the local host. The **lsrun** command exits as soon as xterm starts, and the xterm on the remote host connects directly to the X server on the local host.

xterm on a PC

Each X application makes a separate network connection to the X display on the user's desktop. The application generally gets the information about the display from the DISPLAY environment variable.

X-based systems such as eXceed start applications by making a remote shell connection to the UNIX server, setting the DISPLAY environment variable, and then invoking the X application. Once the application starts, it makes its own connection to the display and the initial remote shell is no longer needed.

This approach can be extended to allow load sharing of remote applications. The client software running on the X display host makes a remote shell connection to any server host in the LSF cluster. Instead of running the X application directly, the client invokes a script that uses LSF to select the best available host and starts the application on that host. Because the application then makes a direct connection to the display, all of the intermediate connections can be closed. The client software on the

display host must select a host in the cluster to start the connection. You can choose an arbitrary host for this; once LSF selects the best host and starts the X application there, the initial host is no longer involved. There is no ongoing load on the initial host.

Set up Exceed to log on the least loaded host

About this task

If you are using a PC as a desktop machine and are running an X Window server on your PC, then you can start an X session on the least loaded host.

The following steps assume you are using Exceed from Hummingbird Communications. This procedure can be used to load share any X-based application.

You can customize host selection by changing the resource requirements specified with `-R "..."`. For example, a user could have several icons in the `xterm` program group: one called `Best`, another called `Best_Sun`, another `Best_HP`.

Procedure

1. Click the Xstart icon in the Exceed program group.
2. Choose REXEC (TCP/IP, ...) as start method, program type is X window.
3. Set the host to be any server host in your LSF cluster:

```
lsrun -R "type==any order[cpu:mem:login]" xterm -sb -ls -display your_PC:0.0
```

4. Set description to be `Best`.
5. Click Install in the Xstart window.
This installs `Best` as an icon in the program group you chose (for example, `xterm`).

The user can now log on to the best host by clicking `Best` in the Xterm program group.

Start an xterm in Exceed

About this task

To start an `xterm`:

Procedure

Double-click `Best`.
An `xterm` starts on the least loaded host in the cluster and is displayed on your screen.

Examples

Run any application on the least loaded host

To run `appY` on the best machine for it, you could set the command line in Exceed to be the following and set the description to `appY`:

```
lsrun -R "type==any && appY order[mem:cpu]" sh -c "appY -display your_PC:0.0 &"
```

You must make sure that all the UNIX servers for `appY` are configured with the resource "`appY`". In this example, `appY` requires a lot of memory when there are embedded graphics, so we make "`mem`" the most important consideration in selecting the best host among the eligible servers.

Start an X session on the least loaded host in any X desktop environment

The above approach also applies to other X desktop environments. In general, if you want to start an X session on the best host, run the following on an LSF host:

```
lsrun -R "resource_requirement" my_Xapp -display your_PC:0.0
```

where

`resource_requirement` is your resource requirement string

Script for automatically specifying resource requirements

The above examples require the specification of resource requirement strings by users. You may want to centralize this such that all users use the same resource specifications.

You can create a central script (for example `lslaunch`) and place it in the `/lsf/bin` directory. For example:

```
#!/bin/sh
lsrun -R "order[cpu:mem:login]" "$@"
exit $?
```

Which would simplify the command string to:

```
lslaunch xterm -sb -ls -display your_PC:0.0
```

Taking this one step further, you could create a script named `lsxterm`:

```
#!/bin/sh
lsrun -R "order[cpu:mem:login]" xterm -sb -ls "$@"
exit $?
```

Which would simplify the command string to:

```
lsxterm -display your_PC:0.0
```

Using LSF with the Etnus TotalView Debugger

- [How IBM Spectrum LSF Works with TotalView](#)
IBM® Spectrum LSF is integrated with Etnus TotalView® multiprocess debugger. You should already be familiar with using TotalView software and debugging parallel applications.
- [Running jobs for TotalView debugging](#)
- [Controlling and monitoring jobs being debugged in TotalView](#)

How IBM Spectrum LSF Works with TotalView

IBM® Spectrum LSF is integrated with Etnus TotalView® multiprocess debugger. You should already be familiar with using TotalView software and debugging parallel applications.

Debugging LSF jobs with TotalView

Etnus TotalView is a source-level and machine-level debugger for analyzing, debugging and tuning multiprocessor or multithreaded programs. LSF works with TotalView two ways:

- Use LSF to start TotalView together with your job
- Start TotalView separately, submit your job through LSF and attach the processes of your job to TotalView for debugging

Once your job is running and its processes are attached to TotalView, you can debug your program as you normally would.

Installing LSF for TotalView

lsfinstall installs the application-specific **esub** program `esub.tvpoe` for debugging POE jobs in TotalView. It behaves like `esub.poe` and runs the `poejob` script, but it also sets the appropriate TotalView options and environment variables for POE jobs.

lsfinstall also configures `hpc_ibm_tv` queue for debugging POE jobs in `lsb.queues`. The queue is not rerunnable, does not allow interactive batch jobs (**bsub -I**), and specifies the following `TERMINATE_WHEN` action:

```
TERMINATE_WHEN=LOAD PREEMPT WINDOW
```

lsfinstall installs the following application-specific **esub** programs to use TotalView with LSF:

- Configures `hpc_linux_tv` queue for debugging MPICH-GM jobs in `lsb.queues`. The queue is not rerunnable, does not allow interactive batch jobs (**bsub -I**), and specifies the following `TERMINATE_WHEN` action:

```
TERMINATE_WHEN=LOAD PREEMPT WINDOW
```

- `esub.tvmpich_gm` for debugging MPICH-GM jobs in TotalView; behaves like `esub.mpich_gm`, but also sets the appropriate TotalView options and environment variables for MPICH-GM jobs, and sends the job to the `hpc_linux_tv` queue

Environment variables for TotalView

On the submission host, make sure that:

- The path to the TotalView binary is in your `$PATH` environment variable
- `$DISPLAY` is set to `console_name:0.0`

Setting TotalView preferences

Before running and debugging jobs with TotalView, you should set the following options in your `$HOME/.preferences.tvd` file:

- `dset ignore_control_c {false}` to allow TotalView to respond to **<CTRL-C>**
- `dset ask_on_dlopen {false}` to tell TotalView not to prompt about stopping processes that use the `dlopen` system call

Limitations

While your job is running and you are using TotalView to debug it, you cannot use LSF job control commands:

- **bchkpnt** and **bmig** are not supported.
- Default TotalView signal processing prevents **bstop** and **bresume** from suspending and resuming jobs, and **bkill** from terminating jobs.
- **brequeue** causes TotalView to display all jobs in error status. Click Go and the jobs will rerun.
- Load thresholds and host dispatch windows do not affect jobs running in TotalView.
- Preemption is not visible to TotalView.
- Rerunning jobs within TotalView is not supported.

Running jobs for TotalView debugging

Submitting jobs

You can submit jobs two ways:

- Start a job and TotalView together through LSF
- Start TotalView and attach the LSF job

You must set the path to the TotalView binary in the \$PATH environment variable on the submission host, and the \$DISPLAY environment variable to console_name:0.0.

Compiling your program for debugging

Before using submitting your job in LSF for debugging in TotalView, compile your source code with the -g compiler option. This option generates the appropriate debugging information in the symbol table.

Any multiprocess programs that call fork(), vfork(), or execve() should be linked to the dbfork library.

Starting a job and TotalView together through LSF

The following syntax applies when starting a job and TotalView together through LSF:

```
bsub -a tvapplication [bsub_options] mpirun.lsf job [job_options] [-tvopt tv_options]
```

Where:

- -a tvapplication specifies the application you want to run through LSF and debug in TotalView.
- -tvopt tv_options specifies options to be passed to TotalView. Use any valid TotalView command option, except -a (LSF uses this option internally). See the TotalView Users Guide for information about TotalView command options and setting up parallel debugging sessions.

For example:

- To submit a POE job and run TotalView:

```
% bsub -a tvpoe -n 2 mpirun.lsf myjob -tvopt -no_ask_on_dlopen
```

The method name tvpoe uses the special **esub** for debugging POE jobs with TotalView (LSF_SERVERDIR/esub.tvpoe). -no_ask_on_dlopen is a TotalView option that tells TotalView not to prompt about stopping processes that use the dlopen system call.

Running TotalView and submitting a job with LSF-PE integration

You can submit jobs with LSF-PE integration running TotalView. Below are some examples:

- % bsub -a tvpoe -n 2 mpirun.lsf myjob -tvopt -no_ask_on_dlopen
- % bsub -a tvpoe -n 2 poe myjob -tvopt -no_ask_on_dlopen
- % bsub -network "" -n 2 totalview -no_ask_on_dlopen poe -a myjob

The above three **bsub** patterns are equivalent. For the latter two **bsub** examples above, the general patterns should be:

- bsub -a tvpoe <other bsub options> poe <program> [program options] [poe options] [-tvopt [totalview options]]

For example:

```
bsub -a tvpoe -n 2 poe myjob myjob_arg1 -eulib ip -tvopt -no_ask_on_dlopen
```

- `bsub -network <network options> <other bsub options> <totalview command line>`

For example:

```
bsub -network "mode=ip" -n 2 totalview -no_ask_on_dlopen poe -a myjob myjob_arg1 -euilib ip
```

Viewing source code while debugging

Use View > Lookup Function to view the source code of your application while debugging. Enter main in the Name field and click OK. TotalView finds the source code for the main() function and displays it in the Source Pane.

Controlling and monitoring jobs being debugged in TotalView

Controlling jobs

While your job is running and you are using TotalView to debug it, you cannot use LSF job control commands:

- **bchkpnt** and **bmig** are not supported.
- Default TotalView signal processing prevents **bstop** and **bresume** from suspending and resuming jobs, and **bkill** from terminating jobs.
- **bqueue** causes TotalView to display all jobs in error status. Click Go and the jobs will rerun.
- Job rerun within TotalView is not supported. Do not submit jobs for debugging to a rerunnable queue.

Register LSF host names and IP addresses to LSF servers

Register the IP address and host name of your local LSF host with LSF servers so that LSF servers do not need to use the DNS server to resolve your local host.

This addresses the issues of resolving the host name and IP address of LSF hosts with non-static IP addresses in environments where the DNS server is not able to properly resolve these hosts after their IP addresses change.

To enable host registration, specify `LSF_REG_FLOAT_HOSTS=Y` in the `lsf.conf` file on each LSF server, or on one LSF server if all servers have access to the `LSB_SHAREDIR` directory. This parameter enables LSF daemons to look for records in the `reghostscache` file when attempting to look up host names or IP addresses.

By default, the `reghostscache` file is stored in the file path as defined by the `LSB_SHAREDIR` parameter in the `lsf.conf` file. Define the `LSB_SHAREDIR` parameter so that the `reghostscache` file can be shared with as many LSF servers as possible. For all LSF servers that have access to the shared directory defined by `LSB_SHAREDIR`, only one of these servers need to receive the registration request from the local host. This reduces network load by reducing the number of servers to which the registration request must be sent. If all hosts in the cluster can access `LSB_SHAREDIR`, the registration only needs to be sent to the management host LIM, which records the host information in the shared `reghostscache` file that all other servers can access. If `LSB_SHAREDIR` is not defined, the `reghostscache` file is placed in the `LSF_TOP` directory.

A typical record in the `reghostscache` file might look like the following:

```
MyHost1 192.168.1.2 S-1-5-21-5615612300-9789239785-9879786971
```

Windows hosts that register have their computer SID included as part of the record. If a registration request is received from an already registered host but its SID does not match with the corresponding record's SID in the `reghostscache` file. This new registration request is rejected, which prevents malicious hosts from imitating another host's name and registering itself as another host.

After enabling host registration, you can register LSF hosts by running the **lsreghost** command from the local host while specifying a path to the `hostregsetup` file:

- UNIX: `lsreghost -s file_path/hostregsetup`
You must run the UNIX command with root privileges. If you want to register the local host at regular intervals, set up a cron job to run this command.
- Windows: `lsreghost -i file_path\hostregsetup`
The Windows command installs **lsreghost** as a Windows service that automatically starts up when the host starts up.

The `hostregsetup` file is a text file with the names of the LSF servers to which the local host must register itself. Each line in the file contains the host name of one LSF server. Empty lines and `#comment` text are ignored.

For example:

```
# Rack 1 hosts
host1A
host1B
host1C

# Rack 2 hosts
host2A
host2B
host2C
```

FLOAT_CLIENTS

The floating clients feature is optional and is an add-on to normal static clients. `FLOAT_CLIENTS_ADDR_RANGE` only applies to floating clients. This includes client hosts with DHCP assigned IP addresses (i.e. laptops).

When LSF is started, the static servers and clients defined under the Host section in the `lsf.cluster` file are added to the cluster. A floating client can be joined only when an LSF command is run from this host and LSF has verified it against `FLOAT_CLIENTS` (total number of floating clients) and `FLOAT_CLIENTS_ADDR_RANGE` (IP range). After verification, they are listed by the **lshosts** command. A floating client expires at 12:00 midnight or when the management host LIM is reconfigured or restarted.

IBM Spectrum LSF performance tuning

Tune the performance and scalability of your LSF cluster.

- [Tune your IBM Spectrum LSF cluster](#)
Tune LIM policies and parameters to improve performance. Use `badadmin mbdrestart -p` to enable parallel restart to Improve `mbatchd` response time after `mbatchd` restart. On UNIX platforms that support thread programming, change default `mbatchd` behavior to use multithreading and increase performance of `bjobs` query requests. Configure Hard CPU affinity to specify the management host CPUs on which `mbatchd` child query processes can run. Use the `JOB_SWITCH2_EVENT` parameter in `lsb.params` to log `mbatchd` performance metrics. Enable `DISPATCH_BY_QUEUE` to increase queue responsiveness.
- [Achieve performance and scalability](#)
Tune LSF for large clusters and monitor performance metrics in real time. Optimize performance in large sites by tuning queries, scheduling, and event logging.

Tune your IBM Spectrum LSF cluster

Tune LIM policies and parameters to improve performance. Use `badadmin mbdrestart -p` to enable parallel restart to Improve `mbatchd` response time after `mbatchd` restart. On UNIX platforms that support thread programming, change default `mbatchd` behavior to use multithreading and increase performance of `bjobs` query requests. Configure Hard CPU affinity to specify the management host CPUs on which `mbatchd` child query processes can run. Use the `JOB_SWITCH2_EVENT` parameter in `lsb.params` to log `mbatchd` performance metrics. Enable `DISPATCH_BY_QUEUE` to increase queue responsiveness.

- [Tune LIM](#)
- [Improve mbatchd response time after mbatchd restart](#)

- [Improve performance of mbatchd query requests on UNIX](#)
Improve **mbatchd** query performance on UNIX systems by using **mbatchd** multithreading, hard CPU affinity, or by configuring the query proxy daemon (**lsproxyd**).
- [Logging mbatchd performance metrics](#)
- [Logging mbatchd and mbschd profiling information](#)
- [Improve performance of mbatchd for job array switching events](#)
- [Increase queue responsiveness](#)
- [Automatically bind LSF daemons to specific CPU cores](#)
Bind a defined set of LSF daemons to CPU cores to distribute CPU resources among critical functions and improve scheduling performance.
- [Use LSF Explorer to improve the performance of the bacct and bhist commands, or to retrieve additional data](#)
Use LSF Explorer to get information for the **bacct** and **bhist** commands more efficiently, or to retrieve additional data for LSF Explorer energy accounting or IBM® Spectrum Scale I/O accounting.
- [Improve slot utilization by preventing bwait from running in jobs](#)

Tune LIM

LIM provides critical services to all LSF components. In addition to the timely collection of resource information, LIM provides host selection and job placement policies. If you are using IBM® MultiCluster, LIM determines how different clusters should exchange load and resource information. You can tune LIM policies and parameters to improve performance.

LIM uses load thresholds to determine whether to place remote jobs on a host. If one or more LSF load indices exceeds the corresponding threshold (too many users, not enough swap space, etc.), then the host is regarded as busy and LIM will not recommend jobs to that host. You can also tune LIM load thresholds.

- [Load thresholds](#)
- [How LSF works with LSF MASTER LIST](#)

Adjust LIM Parameters

There are two main goals in adjusting LIM configuration parameters: improving response time, and reducing interference with interactive use. To improve response time, tune LSF to correctly select the best available host for each job. To reduce interference, tune LSF to avoid overloading any host.

LIM policies are advisory information for applications. Applications can either use the placement decision from LIM, or make further decisions that are based on information from LIM.

Most of the LSF interactive tools use LIM policies to place jobs on the network. LSF uses load and resource information from LIM and makes its own placement decisions based on other factors in addition to load information.

Files that affect LIM are `lsf.shared`, `lsf.cluster.cluster_name`, where `cluster_name` is the name of your cluster.

RUNWINDOW parameter

LIM thresholds and run windows affect the job placement advice of LIM. Job placement advice is not enforced by LIM.

The **RUNWINDOW** parameter defined in `lsf.cluster.cluster_name` specifies one or more time windows during which a host is considered available. If the current time is outside all the defined time windows, the host is considered locked and LIM will not advise any applications to run jobs on the host.

Load thresholds

Load threshold parameters define the conditions beyond which a host is considered busy by LIM and are a major factor in influencing performance. No jobs will be dispatched to a busy host by LIM's policy. Each of these parameters is a load index value, so that if the host load goes beyond that value, the host becomes busy.

LIM uses load thresholds to determine whether to place remote jobs on a host. If one or more LSF load indices exceeds the corresponding threshold (too many users, not enough swap space, etc.), then the host is regarded as busy and LIM will not recommend jobs to that host.

Thresholds can be set for any load index supported internally by the LIM, and for any external load index.

If a particular load index is not specified, LIM assumes that there is no threshold for that load index. Define looser values for load thresholds if you want to aggressively run jobs on a host.

Load indices that affect LIM performance

Load index	Description
r15s	15-second CPU run queue length
r1m	1-minute CPU run queue length
r15m	15-minute CPU run queue length
pg	Paging rate in pages per second
swp	Available swap space
it	Interactive idle time
ls	Number of users logged in

- [Compare LIM load thresholds](#)
- [LIM reports a host as busy](#)
- [Interactive jobs](#)
- [Multiprocessor systems](#)

Compare LIM load thresholds

About this task

Tune LIM load thresholds, compare the output of **lsload** to the thresholds reported by **lshosts -l**.

Procedure

1. Run **lshosts -l**
2. Run **lsload**
The **lsload** command displays an asterisk * next to each load index that exceeds its threshold.

Example

Consider the following output from **lshosts -l** and **lsload**:

```
lshosts -l
HOST_NAME: hostD
...
LOAD_THRESHOLDS:
  r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem
  -      3.5 -    -  15 -   -  -   -   2M  1M

HOST_NAME: hostA
...
LOAD_THRESHOLDS:
  r15s  r1m  r15m  ut  pg  io  ls  it  tmp  swp  mem
  -      3.5 -    -  15 -   -  -   -   2M  1M

lsload
HOST_NAME status r15s  r1m  r15m  ut  pg  ls  it  tmp  swp  mem
```

hostD	ok	0.0	0.0	0.0	0%	0.0	6	0	30M	32M	10M
hostA	busy	1.9	2.1	1.9	47%	*69.6	21	0	38M	96M	60M

In this example, the hosts have the following characteristics:

- hostD is ok.
- hostA is busy, the pg (paging rate) index is 69.6, above the threshold of 15.

LIM reports a host as busy

If LIM often reports a host as busy when the CPU utilization and run queue lengths are relatively low and the system is responding quickly, the most likely cause is the paging rate threshold. Try raising the pg threshold.

Different operating systems assign subtly different meanings to the paging rate statistic, so the threshold needs to be set at different levels for different host types. In particular, HP-UX systems need to be configured with significantly higher pg values; try starting at a value of 50.

There is a point of diminishing returns. As the paging rate rises, eventually the system spends too much time waiting for pages and the CPU utilization decreases. Paging rate is the factor that most directly affects perceived interactive response. If a system is paging heavily, it feels very slow.

Interactive jobs

If you find that interactive jobs slow down system response while LIM still reports your host as ok, reduce the CPU run queue lengths (r15s, r1m, r15m). Likewise, increase CPU run queue lengths if hosts become busy at low loads.

Multiprocessor systems

On multiprocessor systems, CPU run queue lengths (r15s, r1m, r15m) are compared to the effective run queue lengths as displayed by the **lsload -E** command.

CPU run queue lengths should be configured as the load limit for a single processor. Sites with a variety of uniprocessor and multiprocessor machines can use a standard value for r15s, r1m and r15m in the configuration files, and the multiprocessor machines will automatically run more jobs.

Note that the normalized run queue length displayed by **lsload -N** is scaled by the number of processors.

How LSF works with LSF_MASTER_LIST

The files `lsf.shared` and `lsf.cluster.cluster_name` are shared only among LIMs on hosts that are listed as candidates to be elected the management host with the parameter `LSF_MASTER_LIST`.

The preferred management host is no longer the first host in the cluster list in `lsf.cluster.cluster_name`, but the first host in the list specified by `LSF_MASTER_LIST` in `lsf.conf`.

Whenever you reconfigure, only LIMs on the management host candidates read `lsf.shared` and `lsf.cluster.cluster_name` to get updated information. The LIM on the elected management host sends configuration information to child LIMs on the server hosts.

The order in which you specify hosts in `LSF_MASTER_LIST` is the preferred order for selecting hosts to become the management host.

Non-shared file considerations

Generally, the files `lsf.cluster.cluster_name` and `lsf.shared` for hosts that are management candidates should be identical.

When the cluster is started up or reconfigured, LSF rereads configuration files and compares `lsf.cluster.cluster_name` and `lsf.shared` for hosts that are management candidates.

In some cases in which identical files are not shared, files may be out of sync. This section describes situations that may arise should `lsf.cluster.cluster_name` and `lsf.shared` for hosts that are management candidates not be identical to those of the elected management host.

LSF_MASTER_LIST host eligibility

LSF only rejects candidate management hosts listed in `LSF_MASTER_LIST` from the cluster if the number of load indices in `lsf.cluster.cluster_name` or `lsf.shared` for management candidates is different from the number of load indices in the `lsf.cluster.cluster_name` or `lsf.shared` files of the elected management host.

A warning is logged in the log file `lim.log.management_host_name` and the cluster continue to run, but without the hosts that were rejected.

If you want the hosts that were rejected to be part of the cluster, ensure the number of load indices in `lsf.cluster.cluster_name` and `lsf.shared` are identical for all management candidates and restart LIMs on the management and all management candidates:

```
bctrld restart lim hostA hostB hostC
```

Failover with ineligible management host candidates

If the elected management host goes down and if the number of load indices in `lsf.cluster.cluster_name` or `lsf.shared` for the new elected management host is different from the number of load indices in the files of the management host that went down, LSF will reject all management candidates that do not have the same number of load indices in their files as the newly elected management host. LSF will also reject all server-only hosts. This could cause a situation in which only the newly elected management host is considered part of the cluster.

A warning is logged in the log file `lim.log.new_management_host_name` and the cluster continue to run, but without the hosts that were rejected.

To resolve this, from the current management host, restart all LIMs:

```
lsadmin limrestart all
```

All server-only hosts will be considered part of the cluster. Candidate management hosts with a different number of load indices in their `lsf.cluster.cluster_name` or `lsf.shared` files will be rejected.

When the management host that was down comes back up, you need to ensure load indices defined in `lsf.cluster.cluster_name` and `lsf.shared` for all management candidates are identical and restart LIMs on all management candidates.

Using a DNS host cache

A cluster-wide DNS host cache is used to improve cluster startup performance.

To mitigate the burden on the DNS server when starting an LSF cluster, a cluster-based cache file (`$LSF_ENVDIR/.hosts.dnscache`) is used by all daemons on each host to reduce the number of times that LSF daemons directly call the DNS server.

The format of the cache file is the same as `$LSF_ENVDIR/hosts`:

```
IPAddress OfficialName AliasName
```

For shared installations, the management host LIM creates the DNS host cache file `$LSF_ENVDIR/.hosts.dnscache` if the file does not exist. The `mbatchd` daemon periodically flushes the local host cache information into the DNS host cache file.

For non-shared installations, LIM creates the DNS host cache file `$LSF_TMPDIR/.hosts.dnscache` if the file does not exist. LIM periodically flushes local host cache information into the DNS host cache file.

When the IP address or hostname is changed on the DNS server side, LSF daemons can directly call the DNS server to obtain the updated information before the DNS host cache file is flushed with the updated information.

Use the parameter [LSF_HOST_CACHE_DISABLE](#) in `lsf.conf` to disable the use of a cluster-wide DNS host cache file.

Improve mbatchd response time after mbatchd restart

Parallel restart is a mechanism to minimize the LSF downtime (i.e., not responding to user requests) for `mbatchd` restart. The root `mbatchd` is forked, creating a child `mbatchd` process to help with `mbatchd` restart performance. The child `mbatchd` processes regular start up logic, including reading configuration files and replaying events. Meanwhile, the old `mbatchd` can respond to client commands (**bsub**, **bjobs**, etc.), handle job scheduling and status updates, dispatching, and updating new events to event files. When complete, the child `mbatchd` process takes over as parent `mbatchd`, and the old parent `mbatchd` process dies.

While the new `mbatchd` is initializing, the old `mbatchd` is still able to respond to client commands. **admin showstatus** will display the parallel restart status. It helps the administrator know that there is a background `mbatchd` (by PID) doing a parallel restart.

Use **admin mbdrestart -p** to enable parallel restart.

Improve performance of mbatchd query requests on UNIX

Improve **mbatchd** query performance on UNIX systems by using **mbatchd** multithreading, hard CPU affinity, or by configuring the query proxy daemon (**lsproxyd**).

- [Configuring mbatchd to use multithreading](#)
On UNIX platforms that support thread programming, you can change default **mbatchd** behavior to use multithreading and increase performance of query requests when you use the **bjobs** command. Multithreading is beneficial for busy clusters with many jobs and frequent query requests. This may indirectly increase overall **mbatchd** performance.
- [Specify hard CPU affinity](#)
You can also specify the management host CPUs on which **mbatchd** child daemon query processes can run. This so-called *hard CPU affinity* improves **mbatchd** daemon scheduling and dispatch performance by binding query processes to specific CPUs so that higher priority **mbatchd** daemon processes can run more efficiently.
- [Offloading the mbatchd daemon using the LSF rate limiter \(lsproxyd daemon\)](#)
By default, all LSF batch commands contact the **mbatchd** daemon (or the **mbatchd** query child, if configured). When there are excessive requests, such as scripts with tight loop running **bjobs** commands, **mbatchd** can become overloaded, negatively affecting cluster performance. Starting in Fix Pack 14, to protect **mbatchd** from heavy loads, enable the LSF rate limiter (controlled by the **lsproxyd** daemon), which acts as a gatekeeper between the commands and the **mbatchd** daemon. The rate limiter is supported on Linux.

Configuring mbatchd to use multithreading

On UNIX platforms that support thread programming, you can change default **mbatchd** behavior to use multithreading and increase performance of query requests when you use the **bjobs** command. Multithreading is beneficial for busy clusters with many jobs and frequent query requests. This may indirectly increase overall **mbatchd** performance.

About this task

By default, **mbatchd** uses the port defined by the parameter `LSB_MBD_PORT` in the `lsf.conf` file or looks into the system services database for port numbers to communicate with LIM and job request commands.

It uses this port number to receive query requests from clients.

For every query service request received, **mbatchd** forks a child **mbatchd** to service the request. Each child **mbatchd** processes the request and then exits.

When **mbatchd** has a dedicated port specified by the parameter `LSB_QUERY_PORT` in the `lsf.conf` file, it forks a child **mbatchd** which in turn creates threads to process **bjobs** query requests.

As soon as **mbatchd** has forked a child **mbatchd**, the child **mbatchd** takes over, and listens on the port to process more **bjobs** query requests. For each query request, the child **mbatchd** creates a thread to process it.

If you specify `LSB_QUERY_ENH=Y` in `lsf.conf`, batch query multithreading is extended to all **mbatchd** query commands except for the following:

- **bread**
- **bstatus**
- **tspeek**

The child **mbatchd** continues to listen to the port number specified by `LSB_QUERY_PORT` and creates threads to service requests until the job status changes, a new job is submitted, or until the time specified in `MBD_REFRESH_TIME` in `lsb.params` has passed. For pending jobs that changed state (e.g., from `PEND` to `EXIT` caused by the automatic orphan job termination feature), a new child **mbatchd** is created based only on the time configured by the `MBD_REFRESH_TIME` parameter.

Specify a time interval, in seconds, when **mbatchd** will fork a new child **mbatchd** to service query requests to keep information sent back to clients updated. A child **mbatchd** processes query requests creating threads.

`MBD_REFRESH_TIME` has the following syntax:

`MBD_REFRESH_TIME=seconds [min_refresh_time]`

where *min_refresh_time* defines the minimum time (in seconds) that the child **mbatchd** will stay to handle queries. The valid range is 0 - 300. The default is 5 seconds.

- If `MBD_REFRESH_TIME` is $< min_refresh_time$, the child **mbatchd** exits at `MBD_REFRESH_TIME` even if the job changes status or a new job is submitted before `MBD_REFRESH_TIME` expires.
- If `MBD_REFRESH_TIME` $> min_refresh_time$
 - the child **mbatchd** exits at *min_refresh_time* if a job changes status or a new job is submitted before the *min_refresh_time*
 - the child **mbatchd** exits after the *min_refresh_time* when a job changes status or a new job is submitted
- If `MBD_REFRESH_TIME` $> min_refresh_time$ and no job changes status or a new job is submitted, the child **mbatchd** exits at `MBD_REFRESH_TIME`

The default for *min_refresh_time* is 10 seconds.

If you extend multithreaded query support to batch query requests (by specifying `LSB_QUERY_ENH=Y` in `lsf.conf`), the child **mbatchd** will also exit if any of the following commands are run in the cluster:

- **bconf**
- **badmin reconfig**
- **badmin** commands to change a queue's status (**badmin qopen**, **badmin qclose**, **badmin qact**, and **badmin qinact**)
- **badmin** commands to change a host's status (**badmin hopen** and **badmin hclose**)
- **badmin perfmon start**

If you use the **bjobs** command and do not get up-to-date information, you may want to decrease the value of `MBD_REFRESH_TIME` or *min_refresh_time* in `lsb.params` to make it likely that successive job queries could get the newly submitted job information.

Note: Lowering the value of `MBD_REFRESH_TIME` or *min_refresh_time* increases the load on **mbatchd** and might negatively affect performance.

Procedure

1. Specify a query-dedicated port for the **mbatchd** by setting `LSB_QUERY_PORT` in `lsf.conf`.
 2. Optional: Set an interval of time to indicate when a new child **mbatchd** is to be forked by setting `MBD_REFRESH_TIME` in `lsb.params`. The default value of `MBD_REFRESH_TIME` is 5 seconds, and valid values are 0-300 seconds.
 3. Optional: Use `NEWJOB_REFRESH=Y` in `lsb.params` to enable a child **mbatchd** to get up to date new job information from the parent **mbatchd**.
- [Multithread batch queries](#)
LSF supports multithread batch queries for several common batch query commands.
 - [Setting a dedicated query port for mbatchd](#)
To change the default **mbatchd** behavior so that **mbatchd** forks a child **mbatchd** that can create threads, specify a port number with the `LSB_QUERY_PORT` parameter in the `lsf.conf` file.
 - [Specify an expiry time for child mbatchds \(optional\)](#)
 - [Configure mbatchd to push new job information to child mbatchd](#)

Multithread batch queries

LSF supports multithread batch queries for several common batch query commands.

Earlier versions of LSF supported multithread for **bjobs** queries only, but not for other query commands. The following batch query commands do not support multithread batch queries:

- **bread**
- **bstatus**
- **tspeek**

The `LSB_QUERY_ENH` parameter in the `lsf.conf` file extends multithreaded query support to other batch query commands in addition to **bjobs**. In addition, the **mbatchd** system query monitoring mechanism starts automatically instead of being triggered by a query request. This ensures a consistent query response time within the system.

To extend multithread queries to other batch query commands, set the `LSB_QUERY_ENH=Y` parameter in the `lsf.conf` file and use **badmin mbdrestart** to make the change take effect.

Setting a dedicated query port for mbatchd

To change the default **mbatchd** behavior so that **mbatchd** forks a child **mbatchd** that can create threads, specify a port number with the `LSB_QUERY_PORT` parameter in the `lsf.conf` file.

About this task

Tip:

The `LSB_QUERY_PORT` parameter configuration works only on UNIX and Linux systems that support thread programming.

Procedure

1. Log on to the host as the primary LSF administrator.
2. Edit the `lsf.conf` file.
3. Add the `LSB_QUERY_PORT` parameter and specify a port number that is dedicated to receiving requests from hosts.
4. Save the `lsf.conf` file.
5. Reconfigure the cluster:
`badmin mbdrestart`

Specify an expiry time for child mbatchds (optional)

About this task

Use `MBD_REFRESH_TIME` in the `lsb.params` file to define how often the **mbatchd** daemon forks a new child **mbatchd** instance.

Procedure

1. Log on to the host as the primary LSF administrator.
2. Edit `lsb.params`.
3. Add the `MBD_REFRESH_TIME` parameter and specify a time interval in seconds to fork a child **mbatchd** instance.
The default value for this parameter is 5 seconds. Valid values are 0 - 300 seconds.
4. Save the `lsb.params` file.
5. Reconfigure the cluster:
`badmin reconfig`

Configure mbatchd to push new job information to child mbatchd

Before you begin

The `LSB_QUERY_PORT` must be defined in the `lsf.conf` file.

About this task

If you have enabled multithreaded **mbatchd** support, the **bjobs** command might not display up-to-date information if two consecutive query commands are issued before a child **mbatchd** expires because child **mbatchd** job information is not updated. Use the `NEWJOB_REFRESH=Y` in the `lsb.params` file to enable a child **mbatchd** to get up-to-date new job information from the parent **mbatchd**.

When the `NEWJOB_REFRESH=Y` parameter is configured, the parent **mbatchd** daemon pushes new job information to a child **mbatchd** daemon. Job queries with the **bjobs** command display new jobs submitted after the child **mbatchd** was created.

Procedure

1. Log on to the host as the primary LSF administrator.
2. Edit the `lsb.params` file.
3. Add the `NEWJOB_REFRESH=Y` parameter .
You should set the `MBD_REFRESH_TIME` parameter in the `lsb.params` file to a value greater than 10 seconds.
4. Save the `lsb.params` file.
5. Reconfigure the cluster as follows:
`badmin reconfig`

Specify hard CPU affinity

You can also specify the management host CPUs on which **mbatchd** child daemon query processes can run. This so-called *hard CPU affinity* improves **mbatchd** daemon scheduling and dispatch performance by binding query processes to specific CPUs so that higher priority **mbatchd** daemon processes can run more efficiently.

About this task

You can specify the management host CPUs on which **mbatchd** child query processes can run (hard CPU affinity). This improves **mbatchd** daemon scheduling and dispatch performance by binding query processes to specific CPUs so that higher priority **mbatchd** processes can run more efficiently.

When you define the `MBD_QUERY_CPUS` parameter in the `lsb.params` file, LSF runs **mbatchd** child query processes *only* on the specified CPUs. The operating system can assign other processes to run on the same CPU, however, if utilization of the bound CPU is lower than utilization of the unbound CPUs.

Procedure

1. Identify the CPUs on the management host that will run **mbatchd** child query processes.
 - On Linux, get a list of valid CPUs with the following command:
`/proc/cpuinfo`
 - On Solaris, get a list of valid CPUs with the following command:
`psrinfo`
2. In the file `lsb.params`, define the parameter `MBD_QUERY_CPUS`.
For example, the following parameter specifies that the **mbatchd** child query processes run only on CPU numbers 1 and 2 on the management host:

`MBD_QUERY_CPUS=1 2`

You can specify CPU affinity only for management hosts that use one of the following operating systems:
 - Linux 2.6 or higher
 - Solaris 10 or higherIf failover to a management host candidate occurs, LSF maintains the hard CPU affinity, provided that the management host candidate has the same CPU configuration as the original management host. If the configuration differs, LSF ignores the CPU list and reverts to default behavior.
3. Enable the LSF to bind a defined set of LSF daemons to CPUs by defining `LSF_INTELLIGENT_CPU_BIND=Y` in the `lsf.conf` file.
4. Verify that the **mbatchd** child query processes are bound to the correct CPUs on the management host.
 - a. Start up a query process by running a query command such as **bjobs**.
 - b. Check to see that the query process is bound to the correct CPU.
 - On Linux, run the command `taskset -p <pid>`
 - On Solaris, run the command `ps -AP`

Offloading the **mbatchd** daemon using the LSF rate limiter (**lsfproxyd** daemon)

By default, all LSF batch commands contact the **mbatchd** daemon (or the **mbatchd** query child, if configured). When there are excessive requests, such as scripts with tight loop running **bjobs** commands, **mbatchd** can become overloaded, negatively affecting cluster performance. Starting in Fix Pack 14, to protect **mbatchd** from heavy loads, enable the LSF rate limiter (controlled by the **lsfproxyd** daemon), which acts as a gatekeeper between the commands and the **mbatchd** daemon. The rate limiter is supported on Linux.

The rate limiter and the **lsfproxyd** daemon

The rate limiter is managed by the **lsfproxyd** daemon, which monitors and controls the number of requests and connections that can reach the **mbatchd** daemon, protecting it from excess requests. For a request to contact **mbatchd**, it must first obtain a request token from **lsfproxyd**. After completing the request, the token returns to **lsfproxyd**. The **lsfproxyd** daemon distributes tokens in a round-robin fashion, ensuring that each user connection has an fair chance to be served and processed, even under heavy loads.

You can configure to have multiple **lsfproxyd** daemons run within a single cluster; use the `LSF_PROXY_HOSTS` parameter to list the hosts on which you want **lsfproxyd** daemons to run. When multiple **lsfproxyd** daemons are defined for a cluster,

they work together to balance workload and provide high availability: the client command first randomly picks one to use, and if an **lsfproxyd** daemon is unavailable, then the command locates another one to use.

LIM controls starting and restarting the **lsfproxyd** daemon on the LSF hosts specified in the `LSF_PROXY_HOSTS` parameter in the `lsf.conf` file. When the **lsfproxyd** daemon starts, it binds to the listening port specified by the [LSF_PROXY_PORT](#) parameter in the `lsf.conf` file. **LIM** restarts the **lsfproxyd** daemon if it dies.

To control the number of connections to the **mbatchd** daemon, the **lsfproxyd** policy is governed by three attributes set in the [PROXYD_POLICIES](#) parameter of the `lsb.params` configuration file: **max**, **nominal**, and **throttle**. With the `PROXYD_POLICIES` configuration, the **lsfproxyd** policy ensures that users don't monopolize the rate limiter system.

For details on setting up your system for the rate limiter, and using it, see [Enabling and configuring the LSF rate limiter](#).

Daemon log files for diagnosing jobs

To troubleshoot the rate limiter and its interactions with the **lsfproxyd** daemon, see [Diagnostics for the LSF rate limiter and lsfproxyd daemon](#).

Temporarily block users and hosts for performance

Furthermore, to allow an administrator to temporarily block non-administrator and non-root users, hosts, or both, from performing **mbatchd** daemon operations when using the rate limiter, the **badadmin** command has been extended to support [badadmin lsfproxyd block](#). Administrators can run this command to temporarily stop abusive or misbehaving users from interacting with the LSF cluster, and to avoid performance impact on other users.

- [Enabling and configuring the LSF rate limiter](#)
To use the LSF rate limiter, first set parameters in the `lsf.conf` file to configure interaction with the **lsfproxyd** daemon, then configure the number of connections to the rate limiter, and finally, run the **badadmin lsfproxyd** command to control and check the rate limiter. The rate limiter is supported on Linux.
- [Diagnostics for the LSF rate limiter and lsfproxyd daemon](#)
To troubleshoot the rate limiter and its interactions with the **lsfproxyd** daemon, you have various diagnostic options.

Related concepts

- [daemons](#)

Related tasks

- [Enabling daemon log files for diagnostics](#)

Related reference

- [lsf.conf](#)
- [DIAGNOSE LOGDIR](#)
- [ENABLE_DIAGNOSE](#)
- [lsb.params](#)

Enabling and configuring the LSF rate limiter

To use the LSF rate limiter, first set parameters in the `lsf.conf` file to configure interaction with the **lsfproxyd** daemon, then configure the number of connections to the rate limiter, and finally, run the **badadmin lsfproxyd** command to control and check the rate limiter. The rate limiter is supported on Linux.

Procedure

1. Define the hosts and ports that the **lsfproxyd** daemon will connect to for the rate limiter. In the `lsf.conf` file:
 - a. Set the `LSF_PROXYD_HOSTS="hostname ..."`
For example:


```
LSF_PROXYD_HOSTS="hostA hostB"
```

 See [LSF_PROXYD_HOSTS](#) for details.
 - b. Set `LSF_PROXYD_PORT=port_number`
For example:


```
LSF_PROXYD_PORT=1234
```

 See [LSF_PROXYD_PORT](#) for details.
 - c. After setting the `lsf.conf` parameters, restart **LIM** on all hosts defined in your `LSF_PROXYD_HOSTS` parameter by running **lsdamin limrestart**.
2. The **lsfproxyd** policy ensures that users don't monopolize the rate limiter system, and that you can control the number of connections to between the **lsfproxyd** daemon and the **mbatchd** daemon (or the **mbatchd** query child, if configured). The **lsfproxyd** policy is governed by three attributes set in the [PROXYD_POLICIES](#) parameter of the `lsb.params` configuration file: **max**, **nominal**, and **throttle**:
 - a. Set the `PROXYD_POLICIES="QUERY[max=maximum_request_tokens nominal=nominal_request_tokens throttle=request_throttle_time]"`
For example, a policy of `PROXYD_POLICIES="QUERY[max=50 nominal=15 THROTTLE=100]"` specifies:
 - A maximum of 50 available tokens to be distributed from the **lsfproxyd** daemon.
 - A token is distributed immediately if the number of existing tokens used less than 15.
 - The rate limiter waits 100 milliseconds before granting new tokens, when there are currently more than 15 in-use tokens.
 - Once an in-use token reaches its **max** value, a request (from the same category) from a non-root or non-cluster administrator user will not be granted a token. After a certain number of retries (set by the `LSF_PROXYD_NOTOKEN_NTRIES` parameter value in the `lsf.conf` file), the command will then not contact **mbatchd** and will fail with an error message. Note that requests from root or LSF cluster administrators are exempt from this policy as they are always granted a token and their requests are always counted towards the tokens currently in use.
 - b. After setting the `PROXYD_POLICIES` parameter attributes, restart **mbatchd** (by running **badmin mbdrestart**) or reconfigure **mbatchd** (by running **badmin reconfig**).
3. Enable the rate limiter by running the [badmin lsfproxyd enable all|query|sub|other](#) command.
Once enabled, the **lsfproxyd** daemon distributes tokens for the specified request type. For example, to use the rate limiter for queries, run **badmin lsfproxyd enable query**. If disabled, or if all **lsfproxyd** daemons are down, then requests without tokens will be accepted by the **mbatchd** daemon instead.

Tip: To see which type of request type is enabled for the rate limiter, run [badmin lsfproxyd status](#). Here is an example output:

```
$ badmin lsfproxyd status
```

```
lsfproxyd service status:
```

```
QUERY:ENABLED
SUBMISSION:ENABLED
OTHER:ENABLED
```

```
lsfproxyd host status:
```

```
HOSTNAME:host1
STATUS:CONNECTED

      TOKEN_LIMIT  TOKENS_IN_USE
QUERY      100      20
SUBMISSION  -        -
OTHER       -        -
```

What to do next

Additionally, you can set up log files to help diagnose the rate limiter, according to [Diagnostics for the LSF rate limiter and lsfproxyd daemon](#).

You can also further manage the rate limiter with these optional `lsf.conf` configuration parameters:

[LSF_PROXYD_BYPASS](#)

Determine how the **mbatchd** daemon responds to requests without tokens for rate limiter (**lsfproxyd**) requests.

[LSF_PROXYD_HEARTBEAT_INTERVAL](#)

Sets the frequency that the rate limiter sends a heartbeat message to the **mbatchd** daemon.

[LSF_PROXYD_TOKEN_WAIT_TIME](#)

Sets the amount of time that a rate limiter token request can age before it is considered too old.

[LSF_ADDON_HOSTS](#)

Sets any requests for tokens received from the specified hosts as privileged requests. A request for a token will be granted regardless of the current token in use count unless the user or host has been explicitly blocked.

[LSF_PROXYD_NTOKEN_NTRIES](#)

Sets the number of times that the LSF command attempts to contact the **lsfproxyd** daemon after it is not granted a token before quitting.

Diagnostics for the LSF rate limiter and **lsfproxyd** daemon

To troubleshoot the rate limiter and its interactions with the **lsfproxyd** daemon, you have various diagnostic options.

Use the `query_info.queryproxylog.hostname` log file for diagnosing the **lsfproxyd** daemon. The log file shows who issued these requests, where the requests came from, the data size of the query, the receiving and processing times, and whether the request was rejected or accepted. To enable the logs, either statically configure the `ENABLE_DIAGNOSE` parameter in the `lsb.parms` configuration file (if not defined, the default LSF is used), or dynamically enable the diagnostic logs by running the **badadmin diagnose -c query** command, as described in [Enabling daemon log files for diagnostics](#).

To further help debug the **lsfproxyd** daemon, you can also set the debugging log class (level) for the **lsfproxyd** daemon, by configuring the [LSF_DEBUG_PROXYD](#) parameter in the `lsf.conf` file. For example, to set to the log level to log significant program walk steps, run:

```
LSF_DEBUG_PROXYD="LC_TRACE"
```

To specify multiple log classes, use a space-separated list enclosed in quotation marks. For example:

```
LSF_DEBUG_PROXYD="LC_TRACE LC_EXEC"
```

Additionally, if [LSB_DEBUG_MBD="LC2_LSF_PROXYD"](#) is set in the `lsf.conf` file, the **mbatchd** daemon captures information about the rate limiter **lsfproxyd** daemon requests received without a token, and logs them at the **LOG_WARNING** level. These warnings include the user, the request operation code, the LSF header version, and the host from where the request originated. This can be useful information for identifying the user of old binaries.

Related concepts

- [Offloading the mbatchd daemon using the LSF rate limiter \(lsfproxyd daemon\)](#)

Logging mbatchd performance metrics

LSF provides a feature that lets you log performance metrics for **mbatchd**. This feature is useful for troubleshooting large clusters where a cluster has performance problems. In such cases, **mbatchd** performance may be slow in handling high volume request such as:

- Job submission
- Job status requests
- Job rusage requests
- Client info requests causing mbatchd to fork

For example, the output for a large cluster may appear as follows:

```

Nov 14 20:03:25 2012 25408 4 10.1.0 sample period: 120 120
Nov 14 20:03:25 2012 25408 4 10.1.0 job_submission_log_jobfile logJobInfo: 14295 0
179 0 3280 0 10 0 160 0 10 0 990
Nov 14 20:03:25 2012 25408 4 10.1.0 job_submission do_submitReq: 14295 0 180 0 9409
0 100 0 4670 0 10 0 1750
Nov 14 20:03:25 2012 25408 4 10.1.0 job_status_update statusJob: 2089 0 1272 1 2840
0 10 0 170 0 10 0 120
Nov 14 20:03:25 2012 25408 4 10.1.0 job_dispatch_read_jobfile readLogJobInfo: 555 0
256 0 360 0 10 0 70 0 10 0 50
Nov 14 20:03:25 2012 25408 4 10.1.0 mbd_query_job fork: 0 0 0 0 0 0 0 0 0 0 0 0
Nov 14 20:03:25 2012 25408 4 10.1.0 mbd_channel chanSelect/chanPoll: 30171 0 358 0 30037
0 10 0 3930 0 10 0 1270
Nov 14 20:03:25 2012 25408 4 10.1.0 mbd_query_host fork: 0 0 0 0 0 0 0 0 0 0 0 0
Nov 14 20:03:25 2012 25408 4 10.1.0 mbd_query_queue fork: 0 0 0 0 0 0 0 0 0 0 0 0
Nov 14 20:03:25 2012 25408 4 10.1.0 mbd_query_child fork: 19 155 173 160 3058 0 0 0 0
150 170 160 3040
Nov 14 20:03:25 2012 25408 4 10.1.0 mbd_other_query fork: 0 0 0 0 0 0 0 0 0 0 0 0
Nov 14 20:03:25 2012 25408 4 10.1.0 mbd_non_query_fork fork: 0 0 0 0 0 0 0 0 0 0 0 0

```

In the first line (sample period: 120 120) the first value is the configured sample period in seconds. The second value is the real sample period in seconds.

The format for each remaining line is:

```

metricsCategoryName functionName count rt_min rt_max rt_avg rt_total ut_min ut_max ut_avg
ut_total st_min st_max st_avg st_total

```

Where:

- **Count:** Total number of calls to this function in this sample period
- **rt_min:** Min runtime of one call to the function in this sample period
- **rt_max:** Maximum runtime of one call to the function in this sample period
- **rt_avg:** Average runtime of the calls to the function in this sample period
- **rt_total:** Total runtime of all the calls to the function in this sample period
- **ut_min:** Minimum user mode CPU time of one call to the function in this sample period
- **ut_max:** Max user mode CPU time of one call to the function in this sample period
- **ut_avg:** Average user mode CPU time of the calls to the function in this sample period
- **ut_total:** Total user mode CPU time of all the calls to the function in this sample period
- **st_min:** Min system mode CPU time of one call to the function in this sample period
- **st_max:** Max system mode CPU time of one call to the function in this sample period
- **st_avg:** Average system mode CPU time of the calls to the function in this sample period
- **st_total:** Total system mode CPU time of all the calls to the function in this sample period

All time values are in milliseconds.

The `mbatchd` performance logging feature can be enabled and controlled statically through the following parameters in `lsf.conf`:

- **LSB_ENABLE_PERF_METRICS_LOG:** Lets you enable or disable this feature.
- **LSB_PERF_METRICS_LOGDIR:** Sets the directory in which performance metric data is logged.
- **LSB_PERF_METRICS_SAMPLE_PERIOD:** Determines the sampling period for performance metric data.

For more information on these parameters, see the IBM Platform Configuration Reference.

You can also enable the `mbatchd` performance metric logging feature dynamically with the **badadmin perflog** command. The **-t**, **-d** and **-f** command options let you specify the sample period, the duration for data logging, and the output directory. To turn off `mbatchd` performance metric logging, use the **badadmin perflog -o** command.

For more information, see [badadmin](#).

If you define this feature statically, performance metrics are logged in the `mbatchd.perflog.<hostname>` file. If you define the feature dynamically, performance metrics are logged in the log file defined in the command. If you define the feature statically, then dynamically, the data sample period, the log file directory, and the duration will be those defined by the command. After the duration expires, or you turn off the feature dynamically, the statically defined settings are restored.

Logging mbatchd and mbschd profiling information

LSF provides a feature to log profiling information for the **mbatchd** and **mbschd** daemons to track the time that the daemons spend on key functions. This can assist IBM Support with diagnosing daemon performance problems.

To enable daemon profiling with the default settings, edit the `lsf.conf` file, then specify `LSB_PROFILE_MBD=Y` for the **mbatchd** daemon or specify `LSB_PROFILE_SCH=Y` for the **mbschd** daemon. You can also add keywords within these parameters to further customize the daemon profilers.

The profile dump includes one line for each tracked function call stack, and for each thread. The format of the profile dump is as follows:

```
PROFILE "functionName": count rt_min rt_max rt_avg rt_total ut_min ut_max ut_avg ut_total st_min
st_max st_avg st_total start_time end_time thread_id stack_depth "stack"
```

The following is a description of each field:

- `functionName`: The name of the tracked function.
- `count`: The number of times that the function is invoked within the stack corresponding to this record, within the last sampling interval.
- `rt_min`: The minimum runtime of one call to the function in this sampling interval.
- `rt_max`: The maximum runtime of one call to the function in this sampling interval.
- `rt_avg`: The average runtime of the calls to the function in this sampling interval.
- `rt_total`: The total runtime of all the calls to the function in this sampling interval.
- `ut_min`: The minimum user mode CPU time of one call to the function in this sampling interval.
- `ut_max`: The maximum user mode CPU time of one call to the function in this sampling interval.
- `ut_avg`: The average user mode CPU time of the calls to the function in this sampling interval.
- `ut_total`: The total user mode CPU time of all the calls to the function in this sampling interval.
- `st_min`: The minimum system mode CPU time of one call to the function in this sampling interval.
- `st_max`: The maximum system mode CPU time of one call to the function in this sampling interval.
- `st_avg`: The average system mode CPU time of the calls to the function in this sampling interval.
- `st_total`: The total system mode CPU time of all the calls to the function in this sampling interval.
- `start_time`: The start time of this sampling interval in UTC time.
- `end_time`: The end time of this sampling interval in UTC time.
- `thread_id`: The thread ID.
- `stack_depth`: The depth of the stack.
- `stack`: The stack trace. This is a list of function names separated by # characters.

All time values are in milliseconds.

Improve performance of mbatchd for job array switching events

You can improve **mbatchd** performance when switching large job arrays to another queue by enabling the `JOB_SWITCH2_EVENT` in `lsb.params`. This lets **mbatchd** generate the `JOB_SWITCH2` event log. `JOB_SWITCH2` logs the switching of the array to another queue as one event instead of logging the switching of each individual array element. If this parameter is not enabled, **mbatchd** generates the old `JOB_SWITCH` event instead. The `JOB_SWITCH` event is generated for each array element. If the job array is very large, many `JOB_SWITCH` events are generated. **mbatchd** then requires large amounts of memory to replay all the `JOB_SWITCH` events, which can cause performance problems when **mbatchd** starts up.

`JOB_SWITCH2` has the following advantages:

- Reduces memory usage of **mbatchd** when replaying **bswitch** `destination_queue job_ID`, where `job_ID` is the job ID of the job array on which to operate.
- Reduces the time for reading records from `lsb.events` when **mbatchd** starts up.
- Reduces the size of `lsb.events`.

Management batch scheduler performance is also improved when switching large job arrays to another queue. When you **bswitch** a large job array, **mbatchd** no longer signals **mbschd** to switch each job array element individually, which meant

thousands of signals for a job array with thousands of elements. The flood of signals would block `mbschd` from dispatching pending jobs. Now, `mbatchd` only sends one signal to `mbschd`: to switch the whole array. `mbschd` is then free to dispatch pending jobs.

Increase queue responsiveness

You can enable `DISPATCH_BY_QUEUE` to increase queue responsiveness. The scheduling decision for the specified queue will be published without waiting for the whole scheduling session to finish. The scheduling decision for the jobs in the specified queue is final and these jobs cannot be preempted within the same scheduling cycle.

Tip:

Only set this parameter for your highest priority queue (such as for an interactive queue) to ensure that this queue has the highest responsiveness.

Automatically bind LSF daemons to specific CPU cores

Bind a defined set of LSF daemons to CPU cores to distribute CPU resources among critical functions and improve scheduling performance.

You can bind LSF daemons to specific CPU cores by setting the `LSF_INTELLIGENT_CPU_BIND` parameter to `Y` in the `lsf.conf` file.

LSF binds LSF daemons to CPU cores according to the CPU binding configuration file for that host. If a CPU binding file does not exist for the host, LIM automatically detects the topology of that host and binds the LSF daemons according to the automatic binding policy.

LSF initially detects the topology of each management host and management candidate, binds the LSF daemons according to the automatic binding policy, and generates a CPU binding configuration file for that host. These configuration files are in the `LSF_ENVDIR/cpu_binding` directory and the file names are in the format of `host_name.cluster_name` and are in the `LSF_ENVDIR/cpu_binding` directory. LSF binds LSF daemons to CPU cores according to this CPU binding configuration file.

If the CPU binding configuration file no longer exists for the host, LIM automatically detects the topology of that host again and binds the LSF daemons according to the automatic binding policy.

If the CPU binding configuration file is changed, you must restart the LSF daemons by running the following commands in the following order to apply the new CPU binding settings:

1. **`bctrl restart lim`**
2. **`bctrl restart sbd`**
3. **`badmin mbdrestart`**

Note: CPU binding configuration files do not support comments. LSF removes any comments in the CPU binding configuration files after LIM restarts.

The `LSF_INTELLIGENT_CPU_BIND` parameter takes precedence over other CPU binding parameters. If you enable `LSF_INTELLIGENT_CPU_BIND`, LSF ignores the following parameters:

- `EGO_DAEMONS_CPUS`
- `ESTIMATOR_CPUS`
- `LSF_DAEMONS_CPUS`
- `MBD_QUERY_CPUS`

Note: Because this feature binds LSF daemons to eight CPU cores, the `LSF_INTELLIGENT_CPU_BIND` parameter is ignored if the management and management candidate hosts have fewer than eight CPU cores.

Use LSF Explorer to improve the performance of the bacct and bhist commands, or to retrieve additional data

Use LSF Explorer to get information for the **bacct** and **bhist** commands more efficiently, or to retrieve additional data for LSF Explorer energy accounting or IBM Spectrum Scale I/O accounting.

You can improve the performance of the **bacct** and **bhist** commands by using IBM Spectrum LSF Explorer ("LSF Explorer") to get information instead of parsing the `lsb.acct` and `lsb.events` files. Using LSF Explorer improves the performance of the **bacct** and **bhist** commands by avoiding the need for parsing large log files whenever you run these commands.

You can also integrate LSF Explorer energy accounting into LSF. LSF uses LSF Explorer to collect energy data from each host by the LSF Beats services (**lsfbeat** tool). The **bjobs** and **bhosts** commands can then query the data from LSF Explorer and display the job level or host level energy to users.

If you are using IBM Spectrum Scale as the file system with LSF, you can integrate IBM Spectrum Scale job I/O accounting into LSF. LSF uses LSF Explorer to collect IBM Spectrum Scale process-level I/O consumption and combines this LSF job information to provide job-level IBM Spectrum Scale I/O statistics. This allows you to use LSF commands to show job level IBM Spectrum Scale disk consumption or to use LSF Explorer reports to provide job I/O statistics over time.

To use LSF Explorer to retrieve data, you must have LSF Explorer, Version 10.2, or later, installed and working.

To enable this improvement, edit the `lsf.conf` file to define the `LSF_QUERY_ES_SERVERS` and `LSF_QUERY_ES_FUNCTIONS` parameters:

1. Set `LSF_QUERY_ES_SERVERS`=" [http://ip:port|http://host_name:port] ..."

If you have HTTPS enabled for Elasticsearch, then specify `LSF_QUERY_ES_SERVERS`=" [https://ip:port|https://ip:port] ..."

For the *ip* and *port* values, provide a space-separated list of IP addresses and port numbers of the LSF Explorer servers that retrieve log records.

2. Set `LSF_QUERY_ES_FUNCTIONS` with either of the following syntax:

- `LSF_QUERY_ES_FUNCTIONS`=" [acct] | [jobs] | [energy] | [gpfsio]
- `LSF_QUERY_ES_FUNCTIONS`="all"

Specify the functions that are permitted to use LSF Explorer to retrieve data:

acct

If the **acct** function is defined, the **bacct** command uses LSF Explorer to retrieve the accounting log records.

jobs

If the **jobs** function is defined, the **bhist** command uses LSF Explorer to retrieve the event log records.

energy

If the **energy** function is defined and `LSF_ENABLE_BEAT_SERVICE="energy"` is defined in the `lsf.conf` file, the **bhosts** and **bjobs** commands use LSF Explorer to retrieve energy data. For more details, refer to [LSF ENABLE BEAT SERVICE](#).

gpfsio

If the **gpfsio** function is defined, the **bjobs** and **bacct** commands use LSF Explorer to retrieve IBM Spectrum Scale job accounting I/O data.

Specifically, the **bjobs -l** command shows approximate accumulated job disk usage (I/O) data, the **bjobs -o "gpfsio"** command shows job disk usage (I/O) data, and the **bacct -l** command shows the total approximate number of read/write bytes of all storage pools, all on IBM Spectrum Scale

all

If the **all** keyword is defined, all supported functions use LSF Explorer to retrieve data.

Note:

- The **bhist** -cname, -t, and -T options always bypass LSF Explorer and get the information from the `lsb.events` file.
- Use the `bacct -f` or `bhist -f` options to manually bypass LSF Explorer and instead use the specified log files. Alternatively, use the `bacct -f -` or `bhist -f -` options to manually bypass LSF Explorer and use the `lsb.acct` or `lsb.events` log file, respectively.

Improve slot utilization by preventing bwait from running in jobs

You can define `LSB_BWAIT_IN_JOBS=N` in the `lsf.conf` file to prevent LSF from using the **bwait** command within a job. This prevents running jobs from blocking slots with the **bwait** command, since blocking slots leads to low slot utilization. Both the **bwait** command and the **mbatchd** daemon read this parameter, so ensure that you configure this parameter on all `lsf.conf` files if you have non-shared installed servers.

If the **bwait** command is allowed to run within a job, LSF posts the wait condition as an external status message, which means that running a query on the job shows the wait condition.

Achieve performance and scalability

Tune LSF for large clusters and monitor performance metrics in real time. Optimize performance in large sites by tuning queries, scheduling, and event logging.

- [Optimize performance in large sites](#)
- [Tune UNIX for large clusters](#)
- [Tune LSF for large clusters](#)

Optimize performance in large sites

As your site grows, you must tune your LSF cluster to support a large number of hosts and an increased workload.

This chapter discusses how to efficiently tune querying, scheduling, and event logging in a large cluster that scales to 6000 hosts and 500,000 pending jobs at any one time.

LSF performance enhancement features

LSF provides parameters for tuning your cluster, which you will learn about in this chapter. However, before you calculate the values to use for tuning your cluster, consider the following enhancements to the general performance of LSF daemons, job dispatching, and event replaying:

- Both scheduling and querying are much faster
- Switching and replaying the events log file, `lsb.events`, is much faster. The length of the events file no longer impacts performance
- Restarting and reconfiguring your cluster is much faster
- Job submission time is constant. It does not matter how many jobs are in the system. The submission time does not vary.
- The scalability of load updates from the server hosts to the server host has increased
- Load update intervals are scaled automatically

Tune UNIX for large clusters

The following hardware and software specifications are requirements for a large cluster that supports 5,000 hosts and 100,000 jobs at any one time.

Hardware recommendation

LSF management host:

- Four processors, one each for:
 - mbatchd
 - mbschd
 - lim
 - Operating system
- 10-GB RAM

Software requirement

To meet the performance requirements of a large cluster, increase the file descriptor limit of the operating system.

The file descriptor limit of most operating systems used to be fixed, with a limit of 1024 open files. Some operating systems, such as Linux and AIX, have removed this limit, allowing you to increase the number of file descriptors.

- [Increase the file descriptor limit](#)

Increase the file descriptor limit

Procedure

To achieve efficiency of performance in LSF, follow the instructions in your operating system documentation to increase the number of file descriptors on the LSF management host.

Tip:

To optimize your configuration, set your file descriptor limit to a value at least as high as the number of hosts in your cluster.

The following is an example configuration. The instructions for different operating systems, kernels, and shells are varied. You may have already configured the host to use the maximum number of file descriptors that are allowed by the operating system. On some operating systems, the limit is configured dynamically.

Your cluster size is 5000 hosts. Your management host is on Linux, kernel version 2.6:

- a. Log in to the LSF management host as the `root` user.
- b. Add the following line to your `/etc/rc.d/rc.local` startup script:

```
echo -n "5120" > /proc/sys/fs/file-max
```
- c. Restart the operating system to apply the changes.
- d. In the `bash` shell, instruct the operating system to use the new file limits:

```
# ulimit -n unlimited
```

Tune LSF for large clusters

To enable and sustain large clusters, you need to tune LSF for efficient querying, dispatching, and event log management.

- [Manage scheduling performance](#)
The `LSB_MAX_JOB_DISPATCH_PER_SESSION` parameter in the `lsf.conf` file, and the `MAX_SBD_CONNS` parameter in the `lsb.params` file are set automatically during `mbatchd` daemon startup, to enable the fastest possible job dispatch.
- [Limit the number of batch queries](#)
- [Improve the speed of host status updates](#)
- [Limit your user's ability to move jobs in a queue](#)
Control whether users can use `btotop` and `bbot` to move jobs to the top and bottom of queues

- [Manage the number of pending reasons](#)
Condense all the host-based pending reasons into one generic pending reason for efficient, scalable management of pending reasons.
- [Achieve efficient event switching](#)
- [Automatic load updates](#)
- [Manage I/O performance of the info directory](#)
In large clusters, the large numbers of jobs results in a large number of job files stored in the LSF_SHAREDIR/cluster_name/logdir/info directory at any time. When the total size of the job files reaches a certain point, you will notice a significant delay when performing I/O operations in the info directory due to file server directory limits dependent on the file system implementation.
- [Job ID limit](#)

Manage scheduling performance

The LSF_MAX_JOB_DISPATCH_PER_SESSION parameter in the lsf.conf file, and the MAX_SBD_CONNS parameter in the lsb.params file are set automatically during **mbatchd** daemon startup, to enable the fastest possible job dispatch.

The LSF_MAX_JOB_DISPATCH_PER_SESSION value is the maximum number of job decisions that the **mbschd** daemon can make during one job scheduling session. The default value is `LSF_MAX_JOB_DISPATCH_PER_SESSION = Min (MAX(300, Total CPUs), 3000)`.

Sets the maximum number of open file connections between the **mbatchd** and **sbatchd** daemons.

The MAX_SBD_CONNS setting is the maximum number of open file connections between the **mbatchd** and **sbatchd** daemons. The default value is `MAX_SBD_CONNS = 2 * number_of_hosts + 300`. This formula does not provide the exact number of **sbatchd** connections because it also calculates the lost and found hosts. Therefore, the calculated number of connections might be a few more than this theoretical number.

Tip: If you enabled the IBM® Spectrum LSF resource connector, there can be many LSF server nodes joining and leaving from the LSF clusters frequently. In this case, rather than use the default value for MAX_SBD_CONNS, define a more reasonable value. For example, set it as `MAX_SBD_CONNS = 2 * maximum_number_of_hosts + 300`.

The LSF_MAX_JOB_DISPATCH_PER_SESSION and MAX_SBD_CONNS settings affect the number of file descriptors. Although the system sets the default values for both parameters automatically when the **mbatchd** daemon starts, you can adjust them manually.

To decrease the load on the management host LIM, you should not to configure the management host as the first host for the LSF_SERVER_HOSTS parameter.

The values for LSF_MAX_JOB_DISPATCH_PER_SESSION and MAX_SBD_CONNS are not changed dynamically. If hosts are added dynamically, **mbatchd** does not increase their values. Once all the hosts are added, you must run the **badmin mbdrestart** command to set the correct values. If you know in advance that the number of hosts in your cluster will dynamically grow or shrink, then you should configure these parameters beforehand.

- [Enable fast job dispatch](#)
- [Enable continuous scheduling](#)
- [Use scheduler threads to evaluate resource requirement matching](#)
In large-scale clusters with large numbers of hosts, you can enable resource evaluation for hosts concurrently by enabling multithreaded resource evaluation. Set the number of threads the scheduler uses for resource requirement evaluation with the SCHEDULER_THREADS . To set an effective value for this parameter, consider the number of available CPUs on the management host, the number of hosts in the cluster, and the scheduling performance metrics.
- [Limit job dependency evaluation](#)

Related tasks

- [Updating configuration for resource connector](#)

Enable fast job dispatch

Procedure

1. Log in to the LSF management host as the `root` user.
2. Set `LSB_MAX_JOB_DISPATCH_PER_SESSION = Min(Max(300, Total CPUs), 3000)`.
3. Set `MAX_SBD_CONNS` equal to the number of hosts in the cluster plus $2 \times \text{LSB_MAX_JOB_DISPATCH_PER_SESSION}$ plus a buffer of 200.
Note: The system has automatically set this for you. If not suitable, you can manually adjust it.
4. In `lsf.conf`, set the parameter `LSB_MAX_JOB_DISPATCH_PER_SESSION` to a value greater than 300 and less than or equal to one-half the value of `MAX_SBD_CONNS`. Total File Descriptors = Max (Available FDs, `MAX_SBD_CONNS+100`)
Note: The system has automatically set this for you. If not suitable, you can still manually adjust it.
5. In `lsf.conf`, define the parameter `LSF_SERVER_HOSTS` to decrease the load on the management host LIM.
6. In the shell you used to increase the file descriptor limit, shut down the LSF batch daemons on the management host:

```
bctrl stop sbd
```
7. Run **`admin mbdrestart`** to restart the LSF batch daemons on the management host.
8. Run **`bctrl restart sbd all`** to restart every `sbatchd` in the cluster:
Note: When you shut down the batch daemons on the management host, all LSF services are temporarily unavailable, but existing jobs are not affected. When `mbatchd` is later started by `sbatchd`, its previous status is restored and job scheduling continues.

Enable continuous scheduling

Procedure

The scheduler is always running in a production cluster, so setting `JOB_SCHEDULING_INTERVAL=0` means there is no interval between job scheduling.

Use scheduler threads to evaluate resource requirement matching

In large-scale clusters with large numbers of hosts, you can enable resource evaluation for hosts concurrently by enabling multithreaded resource evaluation. Set the number of threads the scheduler uses for resource requirement evaluation with the `SCHEDULER_THREADS`. To set an effective value for this parameter, consider the number of available CPUs on the management host, the number of hosts in the cluster, and the scheduling performance metrics.

Before you begin

Configuring the `SCHEDULER_THREADS` parameter requires that you first have the `LSF_STRICT_RESREQ` parameter set to Y in the `lsf.conf` file.

About this task

This configuration is especially useful for large-scale clusters with numerous hosts. The idea is to evaluate resources for hosts concurrently. For example, there are 6000 hosts in a cluster, so the scheduler may create six threads to complete the evaluation concurrently. Each thread is in charge of 1000 hosts.

Procedure

1. Edit the `lsb.params` file.
2. Specify the value of the `SCHEDULER_THREADS` parameter to a number between 1 and the number of cores on the management host:
`SCHEDULER_THREADS=number_of_threads`

Setting this parameter to 0 means that the scheduler does not create any threads to evaluate resource requirements. This is the default behavior.

Related reference

- [LSF_STRICT_RESREQ](#)

Limit job dependency evaluation

About this task

You can set the maximum number of job dependencies **mbatchd** evaluates in one scheduling cycle. The `EVALUATE_JOB_DEPENDENCY` parameter limits the amount of time **mbatchd** spends on evaluating job dependencies in a scheduling cycle, which limits the amount of time the job dependency evaluation blocks services. Job dependency evaluation is a process that is used to check if each job's dependency condition is satisfied. When a job's dependency condition is satisfied, it sets a ready flag and allows itself to be scheduled by **mbschd**.

When `EVALUATE_JOB_DEPENDENCY` is set, a configured number of jobs are evaluated.

Limit the number of job dependencies **mbatchd** evaluates in a scheduling cycle as follows:

Procedure

1. Edit the `lsb.params` file.
2. Specify the value of the `EVALUATE_JOB_DEPENDENCY` parameter.
`EVALUATE_JOB_DEPENDENCY=integer`

Results

Starting a scheduling session triggers LSF to do job dependency evaluation. The number of jobs evaluated corresponds to the configuration and the endpoint is kept. LSF starts the job dependency evaluation from the endpoint in the next session. LSF evaluates all dependent jobs every 10 minutes regardless of the configuration for `EVALUATE_JOB_DEPENDENCY`.

Limit the number of batch queries

About this task

In large clusters, job querying can grow quickly. If your site sees a lot of high traffic job querying, you can tune LSF to limit the number of job queries that **mbatchd** can handle. This helps decrease the load on the management host.

If a job information query is sent after the limit has been reached, an error message ("*Batch system concurrent query limit exceeded*") is displayed and **mbatchd** keeps retrying, in one second intervals. If the number of job queries later drops below the limit, **mbatchd** handles the query.

Procedure

1. Define the maximum number of concurrent jobs queries to be handled by **mbatchd** in the parameter `MAX_CONCURRENT_QUERY` in `lsb.params`:

- If `mbatchd` is not using multithreading, the value of `MAX_CONCURRENT_QUERY` is always the maximum number of job queries in the cluster.
 - If `mbatchd` is using multithreading (defined by the parameter `LSB_QUERY_PORT` in `lsf.conf`), the number of job queries in the cluster can temporarily become higher than the number specified by `MAX_CONCURRENT_QUERY`. This increase in the total number of job queries is possible because the value of `MAX_CONCURRENT_QUERY` actually sets the maximum number of queries that can be handled by each child `mbatchd` that is forked by `mbatchd`. When the new child `mbatchd` starts, it handles new queries, but the old child `mbatchd` continues to run until all the old queries are finished. It is possible that the total number of job queries can be as high as `MAX_CONCURRENT_QUERY` multiplied by the number of child daemons forked by `mbatchd`.
2. To limit all batch queries (in addition to job queries), specify `LSB_QUERY_ENH=Y` in `lsf.conf`. Enabling this parameter extends multithreaded query support to all batch query requests and extends the `MAX_CONCURRENT_QUERY` parameter to limit all batch queries in addition to job queries.

Improve the speed of host status updates

LSF improves the speed of host status updates as follows:

- Fast host status discovery after cluster startup
- Multi-threaded UDP communications
- Fast response to static or dynamic host status change
- Simultaneously accepts new host registration

LSF features the following performance enhancements to achieve this improvement in speed:

- `LSB_SYNC_HOST_STAT_LIM` (in `lsb.params`) is now enabled by default (previously, this was disabled by default), so there is no need to configure it in the configuration file. This parameter improves the speed with which **mbatchd** obtains host status, and therefore the speed with which LSF reschedules rerunnable jobs: the sooner LSF knows that a host has become unavailable, the sooner LSF reschedules any rerunnable jobs executing on that host. For example, during maintenance operations, the cluster administrator might need to shut down half of the hosts at once. LSF can quickly update the host status and reschedule any rerunnable jobs that were running on the unavailable hosts.
Note: If you previously specified `LSB_SYNC_HOST_STAT_LIM=N` (to disable this parameter), change the parameter value to `Y` to improve performance.
- The default setting for `LSB_MAX_PROBE_SBD` (in `lsf.conf`) was increased from 2 to 20. This parameter specifies the maximum number of **sbatchd** instances polled by **mbatchd** in the interval `MBD_SLEEP_TIME/10`. Use this parameter in large clusters to reduce the time it takes for **mbatchd** to probe all **sbatchds**.
Note: If you previously specified a value for `LSB_MAX_PROBE_SBD` that is less than 20, remove your custom definition to use the default value of 20.
- You can set a limit with `MAX_SBD_FAIL` (in `lsb.params`) for the maximum number of retries for reaching a non-responding server batch daemon, **sbatchd**. If **mbatchd** fails to reach a host after the defined number of tries, the host is considered unavailable or unreachable.

Limit your user's ability to move jobs in a queue

Control whether users can use **btop** and **bbot** to move jobs to the top and bottom of queues

Procedure

Set `JOB_POSITION_CONTROL_BY_ADMIN=Y` in `lsb.params`.

Remember:

You must be an LSF administrator to set this parameter.

Results

When set, only the LSF administrator (including any queue administrators) can use **bbot** and **btop** to move jobs within a queue. A user attempting to use **bbot** or **btop** receives the error “User permission denied.”

Manage the number of pending reasons

Condense all the host-based pending reasons into one generic pending reason for efficient, scalable management of pending reasons.

Procedure

Set `CONDENSE_PENDING_REASONS=Y` in `lsb.params`.

If a job has no other main pending reason, **bjobs -p** or **bjobs -l** will display the following:

Individual host based reasons

If you condense host-based pending reasons, but require a full pending reason list, you can run the following command:

```
badmin diagnose <job_ID>
```

Remember:

You must be an LSF administrator or a queue administrator to run this command.

Achieve efficient event switching

About this task

Periodic switching of the event file can weaken the performance of `mbatchd`, which automatically backs up and rewrites the events file after every 1000 batch job completions. The old `lsb.events` file is moved to `lsb.events.1`, and each old `lsb.events.n` file is moved to `lsb.events.n+1`.

Procedure

Change the frequency of event switching with the following two parameters in `lsb.params`:

- `MAX_JOB_NUM` specifies the number of batch jobs to complete before `lsb.events` is backed up and moved to `lsb.events.1`. The default value is 1000.
- `MIN_SWITCH_PERIOD` controls how frequently `mbatchd` checks the number of completed batch jobs

The two parameters work together. Specify the `MIN_SWITCH_PERIOD` value in seconds.

Tip:

For large clusters, set the `MIN_SWITCH_PERIOD` to a value equal to or greater than 600. This causes `mbatchd` to fork a child process that handles event switching, thereby reducing the load on `mbatchd`. `mbatchd` terminates the child process and appends delta events to new events after the `MIN_SWITCH_PERIOD` has elapsed. If you define a value less than 600 seconds, `mbatchd` will not fork a child process for event switching.

Example

This instructs `mbatchd` to check if the events file has logged 1000 batch job completions every two hours. The two parameters can control the frequency of the events file switching as follows:

- After two hours, `mbatchd` checks the number of completed batch jobs. If 1000 completed jobs have been logged (`MAX_JOB_NUM=1000`), it starts a new event log file. The old event log file is saved as `lsb.events.n`, with subsequent

sequence number suffixes incremented by 1 each time a new log file is started. Event logging continues in the new `lsb.events` file.

- If 1000 jobs complete after five minutes, `mbatchd` does not switch the events file until the end of the two-hour period (`MIN_SWITCH_PERIOD=7200`).

Automatic load updates

Periodically, the LIM daemons exchange load information. In large clusters, let LSF automatically load the information by dynamically adjusting the period that is based on the load.

Important:

For automatic tuning of the loading interval, make sure the parameter `EXINTERVAL` in `lsf.cluster.cluster_name` file is *not* defined. Do not configure your cluster to load the information at specific intervals.

Manage I/O performance of the info directory

In large clusters, the large numbers of jobs results in a large number of job files stored in the `LSF_SHAREDIR/cluster_name/logdir/info` directory at any time. When the total size of the job files reaches a certain point, you will notice a significant delay when performing I/O operations in the info directory due to file server directory limits dependent on the file system implementation.

About this task

By dividing the total file size of the info directory among subdirectories, your cluster can process more job operations before reaching the total size limit of the job files.

Note: Job script files for jobs that are stored in the jobinfo cache are not stored in the info directory, but are stored in `lsb.jobinfo.events` file.

Procedure

1. Use `MAX_INFO_DIRS` in `lsb.params` to create subdirectories and enable `mbatchd` to distribute the job files evenly throughout the subdirectories.

`MAX_INFO_DIRS=num_subdirs`

Where `num_subdirs` specifies the number of subdirectories that you want to create under the `LSF_SHAREDIR/cluster_name/logdir/info` directory. Valid values are positive integers between 1 and 1024. By default, `MAX_INFO_DIRS` is not defined.

2. Run **`badmin reconfig`** to create and use the subdirectories.
Note: If you enabled duplicate event logging, you must run `badmin mbdrestart` instead of `badmin reconfig` to restart `mbatchd`.
3. Run **`bparams -l`** to display the value of the `MAX_INFO_DIRS` parameter.

Example

`MAX_INFO_DIRS=10`

`mbatchd` creates ten subdirectories from `LSB_SHAREDIR/cluster_name/logdir/info/0` to `LSB_SHAREDIR/cluster_name/logdir/info/9`.

Configure a job information directory

Job file I/O operations may impact cluster performance when there are millions of jobs in a LSF cluster. You can configure `LSB_JOBINFO_DIR` on high performance I/O file systems to improve cluster performance. This is separate from the `LSB_SHAREDIR` directory in `lsf.conf`. LSF will access the directory to get the job information files. If the directory does not exist, `mbatchd` will try to create it. If that fails, `mbatchd` exits.

The `LSB_JOBINFO_DIR` directory must be:

- Owned by the primary LSF administrator
- Accessible from all hosts that can potentially become the management host
- Accessible from the management host with read and write permission
- Set for 700 permission

Note: Using the `LSB_JOBINFO_DIR` parameter will require draining the whole cluster.

Job ID limit

By default, LSF assigns job IDs up to six digits. This means that no more than 999999 jobs can be in the system at once. The job ID limit is the highest job ID that LSF will ever assign, and also the maximum number of jobs in the system.

LSF assigns job IDs in sequence. When the job ID limit is reached, the count rolls over, so the next job submitted gets job ID "1". If the original job 1 remains in the system, LSF skips that number and assigns job ID "2", or the next available job ID. If you have so many jobs in the system that the low job IDs are still in use when the maximum job ID is assigned, jobs with sequential numbers could have different submission times.

Increase the maximum job ID

You cannot lower the job ID limit, but you can raise it to 10 digits. This allows longer term job accounting and analysis, and means you can have more jobs in the system, and the job ID numbers will roll over less often.

Use `MAX_JOBID` in `lsb.params` to specify any integer from 999999 to 2147483646 (for practical purposes, you can use any 10-digit integer less than this value).

Increase the job ID display length

By default, **bjobs** and **bhist** display job IDs with a maximum length of seven characters. Job IDs greater than 9999999 are truncated on the left.

Use `LSB_JOBID_DISP_LENGTH` in `lsf.conf` to increase the width of the `JOBID` column in **bjobs** and **bhist** display. When `LSB_JOBID_DISP_LENGTH=10`, the width of the `JOBID` column in **bjobs** and **bhist** increases to 10 characters.

IBM Spectrum LSF energy aware scheduling

Configure, manage, and use IBM® Spectrum LSF energy-aware scheduling features for large-scale LSF installations, where the energy requirements for operating large systems are becoming a significant factor in the overall cost of these systems.

LSF offers energy-aware scheduling features for large scale LSF installations, where the energy requirements for operating large systems are becoming a significant factor in the overall cost of these systems. On large systems with either a long lead period to full production or widely fluctuating workloads, many nodes can sit idle for significant time periods. The energy-aware scheduling features of LSF enable administrators to control the processor frequency to allow some applications to run at a decreased frequency with minor performance degradation. Decreasing the frequency can lead to overall power savings. Conversely, minimizing the frequency on unused cores can also enable maximum turbo boost to active cores, to increase application performance, and to reduce run times. Frequency control allows an organization to balance performance with power savings. It should be noted that for resizable jobs, **bjobs** can only get the energy cost of the latest resizable job's executive hosts.

LSF energy-aware scheduling includes the following features:

- Host-based policies to manage the power state of hosts.
- Ability to set the CPU frequency at the job, application, or queue level.
- Collection and reporting of power usage for an application (assuming exclusive use of nodes).
- Benchmarking application power usage and generation of relevant power coefficients.
- Prediction of performance, power usage, and runtime of applications at different CPU frequencies.
- Automatic CPU frequency selection for jobs based on predictions.
- [Managing host power states](#)
LSF energy aware scheduling host power state management enables automatic workload driven power management policies for hosts in an LSF cluster. LSF can power on hosts as jobs need them, and take appropriate power management actions as workload changes. Power management policies support the power management features of xCAT version 2.7.
- [CPU frequency management](#)
To enable CPU frequency management, set LSF_MANAGE_FREQUENCY in lsf.conf. By default, CPU frequency management is not enabled (LSF_MANAGE_FREQUENCY=N). If LSF_MANAGE_FREQUENCY=N, CPU frequency management is disabled, and lim will not load elim.frequency.
- [Automatic CPU frequency selection](#)
Automatic CPU frequency selection allows an organization to balance performance with power savings.

Managing host power states

LSF energy aware scheduling host power state management enables automatic workload driven power management policies for hosts in an LSF cluster. LSF can power on hosts as jobs need them, and take appropriate power management actions as workload changes. Power management policies support the power management features of xCAT version 2.7.

LSF administrators can set cluster-wide power management policies, and manually manage the power characteristics of specific LSF hosts. Multiple power management policies can also be configured with time windows to manage the power state for specified hosts and host groups automatically.

Cluster administrators can retrieve and monitor the power state changes of specific hosts and view power state of each host, along with the configured power management policy definitions.

System requirements

Host power management for LSF energy aware scheduling has the following requirements:

- All compute nodes have P-States and C-States enabled.
- All LSF management and management candidates must be clients of a provisioning management system, which is able to call corresponding provisioning tool command line to connect with its management node directly.
- xCAT v2.7 or higher should be ready to use for LSF server hosts management
- [Configuring host power state management](#)
Configure host power state management parameters in lsb.params and the PowerPolicy section in lsb.resources.
- [Controlling and monitoring host power state management](#)
The following commands allow for control and monitoring of host power state management.
- [Valid host statuses for power saved mode](#)
For a host to enter power saved mode, it must have one of the following statuses:
- [Disabling the power operation feature](#)
Before disabling the power operation feature, make sure all hosts are in power on status.
- [Changing lsf.shared / lsf.cluster](#)
Before making any changes to lsf.shared or lsf.cluster for resource definition, all server hosts must be in power on status. After **restart lim/mbd**, host can then be power saved by power policy or by **admin hpower**.
- [Integration with Advance Reservation](#)
System Advance Reservation (AR) takes precedence over an automated (configured) power policy. This means:

Configuring host power state management

Configure host power state management parameters in `lsb.params` and the `PowerPolicy` section in `lsb.resources`.

- [Power parameters in lsb.params](#)
The power state management parameters in `lsb.params` enable the power management feature.
- [PowerPolicy section in lsb.resources](#)
This section is used to enable power management policy. Power policies are only enabled when configured.

Power parameters in lsb.params

The power state management parameters in `lsb.params` enable the power management feature.

Suspend, Resume, Reset

To enable the power state management parameters in `lsb.params`, a valid definition includes at least one `POWER_SUSPEND_CMD` and `POWER_RESUME_CMD` pair. The configured command must have full path for execution. For example:

- `POWER_SUSPEND_CMD = $LSF_SERVERDIR/../../util/eass3/rpower_suspend.sh`
- `POWER_RESUME_CMD = $LSF_SERVERDIR/../../util/eass3/rpower_resume.sh`
- `POWER_RESET_CMD = $LSF_SERVERDIR/../../util/eass3/rpower_reset.sh`

The power parameters support the following power actions:

- Suspend (`POWER_SUSPEND_CMD`) put the host in energy saving state. Defines suspend operation command which will be called when LSF handles a host suspend power request. LSF uses the command in the format:
command host [host ...]

The command can parse all its arguments as a host list. The command must return 0 if the power control action succeeds and 1 if the power control action fails. Each line of the output has a host and its return value. For example:

```
host1 0
host2 1
```

A host can be suspended manually or by the power policy. A pending job can resume a suspended host only if it was suspended by the power policy. If the host was suspended manually (**badadmin hpower suspend**), the job cannot put the host back into working state (power resume).

- Resume (`POWER_RESUME_CMD`) put the host in working state. Defines the resume operation command which will be called when LSF handles a host resume power request. It should be an opposite operation to `POWER_SUSPEND_CMD`.
- Reset (`POWER_RESET_CMD`) resets the host. A reset is issued to the host if it fails to join the cluster within a specified time after the resume command is issued (either by manual resume command, or resume triggered by a pending job). The timeout is configured by the parameter `POWER_SUSPEND_TIMEOUT` in `lsb.params` and the default is 10 minutes.

The power parameters are applied cluster-wide, to all configured power policies and manual power operations performed by the administrator. Both `POWER_SUSPEND_CMD` and `POWER_RESUME_CMD` must be specified.

The host can only enter a power saving (suspend) state when it is idle (that is, no jobs are running; `NJOBS=0`) and the host is in “ok” state. For example:

```
POWER_SUSPEND_CMD= rpower suspend
POWER_RESUME_CMD= rpower onstandby
POWER_RESET_CMD= rpower reset
```

Configuring events switching

The parameter `POWER_STATUS_LOG_MAX` in `lsb.params` is used to configure a trigger value for events switching. The default value is 10000. This value takes effect only if `PowerPolicy` (in `lsb.resources`) is enabled.

If a finished job number is not larger than the value of `MAX_JOB_NUM`, the event switch can also be triggered by `POWER_STATUS_LOG_MAX`, which works with `MIN_SWITCH_PERIOD`.

Configuring a wait time after resume

The parameter `POWER_ON_WAIT` in `lsb.params` is used to configure a wait time (in seconds) after a host is resumed and enters ok status, before dispatching a job. This is to allow other services on the host to restart and enter a ready state. The default value is 0 and is applied globally.

PowerPolicy section in `lsb.resources`

This section is used to enable power management policy. Power policies are only enabled when configured.

A host can belong to only one PowerPolicy section. The LSF management host and management host candidates cannot be included in a PowerPolicy.

```
Begin PowerPolicy
  NAME = policy_name
  HOSTS = host_list
  TIME_WINDOW= time_window
  MIN_IDLE_TIME= minutes
  CYCLE_TIME= minutes
End PowerPolicy
```

For example:

```
Begin PowerPolicy
  NAME = policy_night
  HOSTS = hostGroup1 host3
  TIME_WINDOW= 23:00-8:00
  MIN_IDLE_TIME= 1800
  CYCLE_TIME= 60
End PowerPolicy
```

The PowerPolicy section defines the following parameters:

- `NAME=policy_name`
Mandatory. Unique name for the power management policy.

You must specify this parameter to define a power policy. LSF does not automatically assign a default power policy name.

Specify any ASCII string up to 60 characters long. You can use letters, digits, underscores (`_`), dashes (`-`), periods (`.`) in the name. The power policy name must be unique within the cluster.

- `HOSTS=host_list`
Where `host_list` is a space-separated list of the following items:

```
host name
host partition
host group
compute unit
```

Hosts specified cannot overlap among power policies.

Default is all hosts not included in another power policy (except management and management candidate hosts).

- `TIME_WINDOW=time_window`
This is the time period when this policy is active and should be applied to the hosts, the time window syntax should be the same as the rest of LSF. When leaving the `TIME_WINDOW`, hosts defined will automatically wake up. The time window is duration that power policy applies

Default is power policy is always enabled.

- `MIN_IDLE_TIME=minutes`
This parameter only takes effect if a valid `TIME_WINDOW` is configured. It defines the number of minutes a host must be idle before power operations are issued for defined hosts. The default is 0 minutes.

After a host has been idle for this period of time, it is suspended. It is applied within the `TIME_WINDOW`, which means if the time window is not reached, this parameter will not take effect. The idle time calculation is from the actual host idle time, even if it is outside the `TIME_WINDOW`. This counter gets reset when LSF restarts if:

- The host is not running a job.
- The host is in `ok`, `closed_Cu_Excl`, or `ok_Powered` state.
- The host is not part of an active system Advance Reservation.
- `CYCLE_TIME=minutes`
The minimum time in minutes between changes in power state. The counter is changed once the host is power changed. This counter is not reset when LSF restarts

This parameter only takes effect if a valid `TIME_WINDOW` is configured. It defines the minimum time in minutes between changes in power state. The default is 5 minutes. Power actions are issued regardless of recent host status changes.

To define a timeout for power suspend and resume actions, set `POWER_SUSPEND_TIMEOUT` in `lsb.params`. If the power action does not complete in the specified time, LSF treats the operation as failed. The default value is 600 seconds (10 minutes).

Controlling and monitoring host power state management

The following commands allow for control and monitoring of host power state management.

badmin hpower

The option: **hpower** for **badmin** is used to switch the power state of idle host (hosts and host groups including compute unit and host partition hosts) to enter into power saving state or working state manually. For example:

```
badmin hpower suspend | resume [-C comments] host_name [...]
```

Options:

suspend

Puts the host in energy saving state. **badmin hpower suspend** calls the script defined by **POWER_SUSPEND_CMD** in the PowerPolicy, and tags the host so that it cannot be resumed by the PowerPolicy.

resume

Puts the host in working state. The host can enter power save status when `CYCLE_TIME` is reached. If the host should not enter power save status, use the **badmin hclose** command to block the host from the power policy.

-C

Add to describe the specified power management action. Comments are displayed by **badmin hist** and **badmin hhist**.

host_name

Specify one or more host names, host groups, compute units, or host partitions. All specified hosts will be switched to energy saving state or working state. Error message will be shown if the host state is not ready for switching. (Each host is in one line with each message)

badmin hist and badmin hhist

Use **badmin hist** and **badmin hhist** to retrieve the historical information about the power state changes of hosts.

All power related events are logged for both **badmin hpower** and actions triggered by configured (automated) PowerPolicy.

Power State Action	Performed by	Success/Fail	Logged Events
Suspend	By badmin hpower	On Success	Host <host_name> suspend request from administrator <cluster_admin_name>.
			Host <host_name> suspend request done.
			Host <host_name> suspend.

Power State Action	Performed by	Success/Fail	Logged Events
		On Failure	Host <host_name> suspend request from administrator <cluster_admin_name>. Host <host_name> suspend request failed. Host <host_name> power unknown.
	By PowerPolicy	On Success	Host <host_name> suspend request from power policy <policy_name>. Host <host_name> suspend request done. Host <host_name> suspend.
		On Failure	Host <host_name> suspend request from power policy <policy_name>. Host <host_name> suspend request failed. Host <host_name> power unknown.
Resume	By badmin hpower	On Success	Host <host_name> resume request from administrator <cluster_admin_name>. Host <host_name> resume request done. Host <host_name> on.
		On Failure	Host <host_name> resume request from administrator <cluster_admin_name>. Host <host_name> resume request exit. Host <host_name> power unknown.
	By PowerPolicy	On Success	Host <host_name> resume request from power policy <policy_name>. Host <host_name> resume request done. Host <host_name> on.
		On Failure	Host <host_name> resume request from power policy <policy_name>. Host <host_name> resume request exit. Host <host_name> power unknown.

bhosts

Use **bhosts -l** to display the power state for hosts. **bhosts** only shows the power state of the host when PowerPolicy (in **lsb.resources**) is enabled. If the host status becomes unknown (power operation due to failure), the power state is shown as a dash (“-”).

Final power states:

on

The host power state is “On” (Note: power state “on” does not mean the batch host state is “ok”, which depends on whether lim and sbatchd can be connected by the management host.)

suspend

The host is suspended by policy or manually with **badmin hpower**

Intermediate power states:

The following states are displayed when mbatchd has sent a request for power operations but the execution has not returned back. If the operation command returns, LSF assumes the operation is done. The intermediate status will be changed.

restarting

The host is resetting when resume operation failed.

resuming

The host is being resumed from standby state which is triggered by either policy or job, or cluster administrator

suspending

The host is being suspended which is triggered by either policy or cluster administrator

Final host state under administrator control:

closed_Power

The host is put into power saving (suspend) state by the cluster administrator

Final host state under policy control:

ok_Power

A transitional state displayed while the host waits for **sbatchd** to resume. Lets **mbatchd** know that the host may be considered for scheduling, but it cannot immediately be used for jobs.

A host may enter this state in two ways:

1. An LSF host which is manually resumed (using **badmin hpower resume**), after it was manually suspended (using **badmin hpower suspend**).
2. When PowerPolicy is defined in lsb.resources, a member host that is suspended by the policy automatically has its power state suspended. The state of this host will be displayed as **ok_Power** (rather than **closed_Power**). This is different from suspending the host manually (by **badmin hpower suspend**) because this host may be woken by job scheduling even it was suspended by the policy.

Example bhosts:

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
host1	closed	-	4	0	0	0	0	0
host2	ok_Power	-	4	0	0	0	0	0
host3	unavail	-	4	0	0	0	0	0

Example bhosts -w:

HOST_NAME	STATUS	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV
host1	closed_Power	-	4	0	0	0	0	0
host2	ok_Power	-	4	0	0	0	0	0
host3	unavail	-	4	0	0	0	0	0

Example bhosts -l:

HOST host1

STATUS	CPUF	JL/U	MAX	NJOBS	RUN	SSUSP	USUSP	RSV	DISPATCH_WINDOW
closed_Power	1.00	-	4	4	4	0	0	-	-

CURRENT LOAD USED FOR SCHEDULING:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem	slots
Total	0.0	0.0	0.0	0%	0.0	0	0	0	31G	31G	12G	0
Reserved	0.0	0.0	0.0	0%	0.0	0	0	0	0M	0M	4096M	-

LOAD THRESHOLD USED FOR SCHEDULING:

	r15s	r1m	r15m	ut	pg	io	ls	it	tmp	swp	mem
loadSched	-	-	-	-	-	-	-	-	-	-	-
loadStop	-	-	-	-	-	-	-	-	-	-	-

POWER STATUS: ok
IDLE TIME: 2m 12s

CYCLE TIME REMAINING: 3m 1s

bjobs

When a host in energy saving state host is switched to working state by a job (that is, the job has been dispatched and waiting for the host to resume), its state is not shown as pending. Instead, it is displayed as provisioning (PROV). For example:

```
bjobs
```

JOBID	USER	STAT	QUEUE	FROM_HOST	EXEC_HOST	JOB_NAME	SUBMIT_TIME
204	root	PROV	normal	host2	host1	sleep 9999	Jun 5 15:24

The state PROV is displayed. This state shows that the job is dispatched to a suspended host, and this host is being resumed. The job remains in PROV state until LSF dispatches the job.

When a job is requires a host in energy saving state or the host is powered off, and LSF is switching the host to working state, the following event is appended by **bjobs -l**:

```
Mon Nov 5 16:40:47: Will start on 2 Hosts <host1> <host2>. Waiting for machine provisioning;
```

The message indicates which host is being provisioned and how many slots are requested.

bhist

When a job is dispatched to a standby host and provisioning the host to resume to working state is triggered, two events are saved into lsb.events and lsb.streams. For example:

```
Tue Nov 19 01:29:20: Host is being provisioned for job. Waiting for host <xxxx> to power on;
```

```
Tue Nov 19 01:30:06: Host provisioning is done;
```

bresources

Use **bresources -p** to show the configured energy aware scheduling policies. For example:

```
bresources -p
```

```
Begin PowerPolicy
  NAME = policy_night
  HOSTS = hostGroup1 host3
  TIME_WINDOW= 23:59-5:00
  MIN_IDLE_TIME= 1800
  CYCLE_TIME= 60
  APPLIED = Yes
End PowerPolicy
```

```
Begin PowerPolicy
  NAME = policy_other
  HOSTS = all
  TIME_WINDOW= all
  APPLIED = Yes
End PowerPolicy
```

In the above case, “policy_night” is defined only for hostGroup1 and host3 and applies during the hours of 23:59 and 5:00. In contrast, “policy_other” covers all other hosts not included in the “policy_night” power policy (with the exception of management and management candidate hosts) and is in effect at all hours.

Valid host statuses for power saved mode

For a host to enter power saved mode, it must have one of the following statuses:

Host Status	Automated (Configured) Power Policy	Manual Power Save Mode (badmin operation)
ok	Yes	Yes

Host Status	Automated (Configured) Power Policy	Manual Power Save Mode (badmin operation)
closed_Cu_Excl	Yes	Yes
closed_Adm		Yes
closed_Busy		Yes
closed_Lock		Yes
closed_Wind		Yes
closed_Full		Yes

Hosts in the following statuses may not enter power saved mode:

- closed_Excl
- closed_LIM
- unavailable
- unreach
- closed_EGO

Disabling the power operation feature

Before disabling the power operation feature, make sure all hosts are in power on status.

If a host is in power saved mode when you disable the power operation feature on the cluster, that host cannot be powered back on (resume) because that feature has been disabled.

Changing lsf.shared / lsf.cluster

Before making any changes to lsf.shared or lsf.cluster for resource definition, all server hosts must be in power on status. After **restart lim/mbd**, host can then be power saved by power policy or by **badmin hpower**.

Resource information persists for power saved hosts. Therefore, if resources are changed while a host is in power saved mode, the obsolete information may cause problems for mbatchd/mbschd.

Integration with Advance Reservation

System Advance Reservation (AR) takes precedence over an automated (configured) power policy. This means:

- A host in system AR does not assume the power saved mode.
- A host in power saved mode will resume when it enters system AR mode even if it breaks CYCLE_TIME.

However, manual power operations will overrule system AR. This means:

- A host in system AR can be suspended using **badmin hpower**.
- A host in manual power saved mode (using **badmin hpower**) does not resume even when it enters system AR mode.

- [Integration with provisioning systems](#)

The power parameters in lsb.params enable cluster administrators to specify the execution commands for changing the power state of hosts. The commands used for power control actions must return 0 if the power control action succeeds and 1 if the power control action fails.

Integration with provisioning systems

The power parameters in `lsb.params` enable cluster administrators to specify the execution commands for changing the power state of hosts. The commands used for power control actions must return 0 if the power control action succeeds and 1 if the power control action fails.

LSF does not maintain any information from third-party provisioning tools, and does not store any credentials or passwords for these provisioning systems. For xCAT, the LSF management host and all management candidates must be configured as clients of the provisioning system, including the SSL credentials shared with the management node. This allows LSF to issue **power** provisioning requests directly.

LSF provides the following example power action scripts for xCAT:

- `POWER_SUSPEND_CMD = $LSF_SERVERDIR/../../util/eass3/rpower_suspend.sh`
- `POWER_RESUME_CMD = $LSF_SERVERDIR/../../util/eass3/rpower_resume.sh`
- `POWER_RESET_CMD = $LSF_SERVERDIR/../../util/eass3/rpower_reset.sh`

CPU frequency management

To enable CPU frequency management, set `LSF_MANAGE_FREQUENCY` in `lsf.conf`. By default, CPU frequency management is not enabled (`LSF_MANAGE_FREQUENCY=N`). If `LSF_MANAGE_FREQUENCY=N`, CPU frequency management is disabled, and `lim` will not load `elim.frequency`.

System requirements

The following Linux kernel modules must be installed on all nodes:

- `msr`
- `ibmaem`
- `ipmi_si`
- `acpi_cpufreq`

All compute nodes have the `cpufreq-util` package installed.

Note: The linux kernel module may already be statically linked to the kernel. This can be confirmed in the file `/boot/config-2.6.32-220.el6.x86_64` where "2.6.32-220" is the kernel number used.

When an OS is installed it may already contain the kernel module in the Linux kernel, so you cannot re-probe the module when the OS starts up. Check the following:

- `msr`: `CONFIG_X86_MSR`
- `ibmaem`: `CONFIG_SENSORS_IBMAEM`
- `ipmi_si`: `CONFIG_IPMI_SI`
- `acpi_cpufreq`: `CONFIG_X86_ACPI_CPUFREQ`

If the keyword equals "y", then the module is already statically linked. If there is an "m", it means you must perform a `modprobe` when the OS starts up.

- [Configuring CPU frequency management](#)
Set `LSF_MANAGE_FREQUENCY` in `lsf.conf` to specify how CPU frequency is set for the job. `LSF_MANAGE_FREQUENCY` accepts the following values:
- [Specifying CPU frequency management for jobs](#)
Set `CPU_FREQUENCY` in `lsb.applications` or `lsb.queue`s to specify required CPU frequency in an application profile or a queue. Specify a value for the required CPU frequency. If no unit is specified, the default unit is GHz. Use MHz to specify a CPU frequency in MHz. All jobs submitted to the application or the queue will request the specified frequency.
- [Job energy usage reporting](#)
To enable job energy usage, set `LSF_COLLECT_ENERGY_USAGE=Y` in `lsf.conf`. By default, job energy usage reporting is not enabled (`LSF_COLLECT_ENERGY_USAGE=N`). If `LSF_COLLECT_ENERGY_USAGE=N`, job energy usage reporting is disabled.
- [Resource usage in job summary email](#)
With EAS features enabled, using the `bsub -o output_file` command the output file for the Job Summary information will include the following information on resource usage:

Configuring CPU frequency management

Set LSF_MANAGE_FREQUENCY in lsf.conf to specify how CPU frequency is set for the job. LSF_MANAGE_FREQUENCY accepts the following values:

HOST

Jobs require CPU frequency to be set for the entire host. Jobs that require the specified maximum CPU frequency must be submitted as exclusive jobs (bsub -x).

CORE

Jobs require CPU frequency to be set by CPU core. Jobs must be submitted with CPU affinity resource requirements.

Specifying CPU frequency management for jobs

Set CPU_FREQUENCY in lsb.applications or lsb.queues to specify required CPU frequency in an application profile or a queue. Specify a value for the required CPU frequency. If no unit is specified, the default unit is GHz. Use MHz to specify a CPU frequency in MHz. All jobs submitted to the application or the queue will request the specified frequency.

Use **bsub -freq** to submit a job with a required CPU frequency. You can specify frequency units as KHz, MHz or GHz. If no unit is specified, the default is GHz. For example, the following job requires a CPU frequency of 2.5 GHz. CPU frequency is managed by host, so the job is an exclusive job:

bsub -x -freq 2.5GHz myjob

The following job requires a CPU frequency of 2.5 GHz, but in this case, CPU frequency is managed by core, so the job is submitted with an affinity resource requirement:

bsub -R "affinity[core]" -freq 2.5GHz myjob

Job-level frequency specified with **bsub -freq** overrides the application-level frequency, and application-level frequency overrides queue-level specification.

Use **bmod -freq** to modify the CPU requirements for the job. Use **bmod -freqn** to remove job-level frequency requirements. You can only modify frequency for pending jobs. You cannot modify the CPU frequency of running jobs.

When LSF sets the specified maximum CPU frequency, it also sets the CPU governor “on demand”. The operating system will dynamically change the CPU frequency based on the minimum and maximum CPU frequency specified for the job.

Use bjobs use to display the specified maximum CPU frequency:

```
bjobs -l
```

```
Job <304>, User <user1>, Project <default>, Application <8proc>, Status <RUN>,  
Queue <normal>, Specified CPU Frequency <2.5 GHz>, Combined CPU Frequency <2.5 GHz>,  
Command <#!/bin/csh;#BSUB -q normal ;#BSUB -app '8proc';rm -rf /tmp/user1; myjob>
```

The Combined CPU Frequency is the CPU frequency setting of the job (**bsub -freq**) combined with the queue and application configuration (CPU_FREQUENCY), if any. This value is set by mbatchd when the job starts.

CPU frequency management makes use of two new dynamic string resources you must define in lsf.shared:

availcpufreqs	String	3600	()	N
currcpufreqs	String	15	()	N

and in lsf.cluster.<cluster_name>:

availcpufreqs	[default]
currcpufreqs	[default]

availcpufreqs

Logical CPU available frequency updated by elim.frequency every 3600 seconds.

currcpufreqs

Current logical CPU frequency updated by elim.frequency every 15 seconds.

Submit a job with a target CPU frequency:

- By core – target CPU frequency is set to the specified frequency
- By host – all CPUs in the host are set to the specified frequency

Use **lshosts** to display CPU frequency for a host:

```
# lshosts -l hostA
```

```
...
AVAILABLE CPU FREQUENCY( GHz ):
2.7 2.6 2.5 2.4 2.3 2.2 2.1 2.0 1.9 1.8 1.7 1.6 1.5 1.4 1.3 1.2
CURRENT CPU FREQUENCY( GHz ):
Frequency          CPUs
1.5                0, 2, 4-6
2.0                1, 3, 7-8
```

The environment variable LSB_SUB_FREQUENCY is used by **esub** to set CPU frequency.

Job energy usage reporting

To enable job energy usage, set LSF_COLLECT_ENERGY_USAGE=Y in lsf.conf. By default, job energy usage reporting is not enabled (LSF_COLLECT_ENERGY_USAGE=N). If LSF_COLLECT_ENERGY_USAGE=N, job energy usage reporting is disabled.

Jobs that require job energy usage reporting must be submitted as exclusive jobs (**bsub -x**).

Use **bacct** to display job energy consumption:

```
bacct -l
```

```
...
JOB ENERGY CONSUMPTION:
20.5kWh
```

Note: Only **blaunch** jobs will collect all energy usage for all hosts. Parallel jobs will collect energy usage for just the first host.

Resource usage in job summary email

With EAS features enabled, using the **bsub -o output_file** command the output file for the Job Summary information will include the following information on resource usage:

```
Resource usage summary:
CPU time :          0.11 sec.
Max Memory :        1 MB
Average Memory :    1.00 MB
Total Requested Memory : -
Delta Memory :      -
(Delta Memory is the difference between Total Requested Memory and Max Memory.)
Max Swap :          222 MB
Max Processes :     3
Max Threads :       4
Job Energy Consumption : 0.000447 kWh
```

The output (if any) follows:

Automatic CPU frequency selection

Automatic CPU frequency selection allows an organization to balance performance with power savings.

Note: Automatic CPU frequency selection is now deprecated and might be removed in a future version of LSF.

LSF uses a formula to predict the power consumption and the elapsed time of the job running in a specific CPU frequency. The coefficients used in the formula vary depending on hardware configuration. Before any job is scheduled to run in a cluster, the coefficients need to be determined on every compute node in each frequency.

Running at a lower CPU frequency can save energy, but machine performance may suffer and the run time will be longer. Each job may have different resource requirements. The energy consumption may be very different between a CPU-bound job and an IO-bound job. LSF's automatic CPU frequency selection feature makes it easier to choose the best frequency at which to run your jobs to maximize energy savings and minimize run time.

Each compute node runs in the nominal CPU frequency by default. When the node is idle or after it has completed a job, the compute node will switch back to nominal frequency.

- [Prerequisites](#)
- [Configuring automatic CPU frequency selection](#)
There are three major configuration steps required to enable the automatic CPU frequency selection feature of LSF:
- [Creating an energy policy tag](#)
An energy policy tag is created by submitting jobs. The job runs using the default CPU frequency. When the job is finished, LSF collects the following information and adds it to the energy policy tag:

Prerequisites

- Only iDataPlex is supported, on homogeneous nodes (same hardware, OS, CPU count, memory). Hyperthreading must be disabled on all nodes.
- No compute node may be in turbo-boost mode.
- The cpufrequtils package is installed on all compute nodes. (Use yum install or obtain an rpm package from your Linux distribution ISO.)
- unixODBC must be on the management/management candidate hosts.
- mysql-connector-odbc must be on the management/management candidate hosts.
- MySQL DB/xCat MySQL DB must be installed to save coefficient data and tag data.
- STREAM and NPB-NAS Parallel Benchmarks are required.
- [Configure MySQL database](#)
Before you can begin, you must set up your MySQL database with the required information (that is, database name, port number, the user name to use and the password, and so forth.

Configure MySQL database

Before you can begin, you must set up your MySQL database with the required information (that is, database name, port number, the user name to use and the password, and so forth.

- For xCat MySQL, open the file /etc/xcat/cfgloc and define:

```
Mysql:dbname=<user_defined_database>;host=<mgmtnode>;port=<port>\userid\pw
```

- For unixODBC, open the file /etc/unixODBC/odbc.ini and define:

```
[user_defined_database]
Description = MySQL database
Driver      = MySQL
SERVER     =
USER       = root
PASSWORD   = root
PORT       = 3306
DATABASE   = user_defined_database
```

Note: If no xCat database is configured, LSF will use the DSN (Data Sources Name) “easdb” in /etc/unixODBC/odbc.ini as the default database for energy aware scheduling features.

Configuring automatic CPU frequency selection

There are three major configuration steps required to enable the automatic CPU frequency selection feature of LSF:

Note: Automatic CPU frequency selection is now deprecated and might be removed in a future version of LSF.

- Install benchmarking programs
- Calculate coefficients data
- Submit a job using an energy policy tag name
- [Installing and configuring benchmarking programs](#)
You must install and run 7 benchmark programs (6 NPB and 1 STREAM) on all compute nodes that will calculate coefficients (or make them available in a location accessible by all compute nodes).
- [Checking compute node performance](#)
Before calculating coefficient data for each compute node it is necessary to check that the performance of each compute node in the cluster performs as predicted. This is done by running the STREAM benchmarking program.
- [Calculating coefficient data](#)
LSF provides an initialization script (initialize_eas in \$LSF_BINDIR) that calculates coefficients and must be run on all compute nodes.

Installing and configuring benchmarking programs

You must install and run 7 benchmark programs (6 NPB and 1 STREAM) on all compute nodes that will calculate coefficients (or make them available in a location accessible by all compute nodes).

About this task

- NPB (NAS Parallel Benchmarks) (<https://www.nas.nasa.gov/cgi-bin/software/start>): Developed for performance evaluation of highly parallel supercomputers. Consists of five parallel kernels and three simulated application benchmarks.
- STREAM (<http://www.cs.virginia.edu/stream/FTP/Code/>): The industry standard benchmark for measuring sustained memory bandwidth.

Note: Run each benchmarking program as root.

Note: For better performance with STREAM, we recommend using icc to compile STREAM.

Important: After installing benchmarking programs, restart the LSF cluster.

The following steps will guide you through downloading and installing these benchmarking programs:

Procedure

1. Download the **NPB-NAS** source code (Version: NPB 3.3) Parallel benchmarks (<https://www.nas.nasa.gov/cgi-bin/software/start>). The six benchmarks in NPB 3.3 are: **bt.C**, **cg.C**, **ep.D**, **lu.C**, **sp.C**, and **ua.C**.
2. Download the **STREAM** source code (<http://www.cs.virginia.edu/stream/FTP/Code/>).
3. Unpack the NPB3.3 benchmarks in the compute nodes and go to the NPB-OMP directory. For example:

```
~/benchmarks/NASA/NPB3.3/NPB3.3-OMP # ls -F
BT/  CG/  DC/  EP/  FT/  IS/  LU/  MG/
Makefile*  README*  README.install*
SP/  UA/  bin/  common/  config/  sys/
```

4. Integrate the STREAM source code with the NASA-OMP source code:
 - a. Create a directory called ST under the NPB3.3-OMP directory and copy the STREAM source code into that directory. For example:

```
~/benchmarks/NASA/NPB3.3/NPB3.3-OMP/ST # ls
HISTORY.txt LICENSE.txt Makefile READ.ME mysecond.c stream.c
stream.c.5.10 stream.f
```

- b. Modify the STREAM Makefile according to NPB3.3-OMP style. For example:

```
~/benchmarks/NASA/NPB3.3/NPB3.3-OMP/ST # cat Makefile
SHELL=/bin/sh
BENCHMARK=st
BENCHMARKU=ST

include ../config/make.def

OBSJS = stream.o

include ../sys/make.common
${PROGRAM}: ${OBSJS}
        ${CLINK} ${CLINKFLAGS} -o ${PROGRAM} ${OBSJS} ${C_LIB}
stream.o:      stream.c
        ${CCOMPILE} stream.c
clean:
        - rm -f *.o *~
        - rm -f core
        - if [ -d rii_files ]; then rm -r rii_files; fi
```

- c. Modify the NPB3.3-OMP Makefile to add the STREAM benchmark. The following is an example of the NPB3.3-OMP Makefile:

```
~/benchmarks/NASA/NPB3.3/NPB3.3-OMP # cat Makefile
SHELL=/bin/sh
CLASS=W
VERSION=
SFILE=config/suite.def

default: header
        @ sys/print_instructions

BT: bt
bt: header
        cd BT; $(MAKE) CLASS=$(CLASS) VERSION=$(VERSION)
ST: st
st: header
        cd ST; $(MAKE) CLASS=$(CLASS)
```

- d. Generate the NPB3.3-OMP definition file from the suite.template and select the benchmarks to use for LSF energy. For example:

```
~/benchmarks/NASA/NPB3.3/NPB3.3-OMP/config # cp suite.def.template
suite.def
```

- e. Change the suite.def file as follows:

```
~/benchmarks/NASA/NPB3.3/NPB3.3-OMP/config # cat suite.def
# config/suite.def
# This file is used to build several benchmarks with a single command.
# Typing "make suite" in the main directory will build all the benchmarks
# specified in this file.
# Each line of this file contains a benchmark name and the class.
# The name is one of "cg", "is", "dc", "ep", "mg", "ft", "sp",
# "bt", "lu", and "ua".
# The class is one of "S", "W", "A" through "E"
# (except that no classes C,D,E for DC and no class E for IS and UA).
# No blank lines.
# The following example builds sample sizes of all benchmarks.
sp      C
lu      C
bt      C
ep      D
cg      C
ua      C
st      U
```

Note: The last line st U is for the STREAM benchmark.

f. Generate make.def from the make.def.template and configure the compiler name.

Note: GCC and GFortran are required on each compute node to compile the benchmark data. Set the proper compiler name in the make.def file:

```
make.def:
...
CC = cc
F77 = gfortran
```

5. Compile the benchmarks:

```
~/benchmarks/NASA/NPB3.3/NPB3.3-OMP # make suite
```

The binaries are saved into the NPB3.3-OMP bin directory:

```
~/benchmarks/NASA/NPB3.3/NPB3.3-OMP # cd bin
~/benchmarks/NASA/NPB3.3/NPB3.3-OMP/bin # ls
bt.C cg.C ep.D lu.C sp.C st.U ua.C
```

Checking compute node performance

Before calculating coefficient data for each compute node it is necessary to check that the performance of each compute node in the cluster performs as predicted. This is done by running the STREAM benchmarking program.

Perform the following on all compute nodes in the cluster:

1. Set the compute nodes to run in a default frequency (The default CPU frequency can be set using the utility **initialize_eas -f**).
2. Run STREAM on each compute node 10 times.
3. Gather the performance value of the benchmark.
The output of the STREAM benchmark is the triad value (the performance value).
4. Calculate the average performance value of each compute node and compare it with the reference value.

Note: A node should not be used for energy aware scheduling if the measured performance is more than 4% lower than the reference value.

Note: The reference value is 70GB/s.

If a problem node is found after running the STREAM benchmarking program, you can:

- Check that the firmware of the problem nodes is the same as other nodes.
- Check that the threading mode (like Turbo or HT) is functioning on the problem nodes.
- Check the current CPU frequency of the problem nodes.
- Check the memory configuration of the problem nodes.

After performing the recommended checks, rerun the STREAM benchmark.

Calculating coefficient data

LSF provides an initialization script (initialize_eas in \$LSF_BINDIR) that calculates coefficients and must be run on all compute nodes.

The initialization utility:

- retrieves all supported CPU frequencies of each node and changes the CPU frequency when running the benchmark programs.
- collects the hardware counters of the 7 benchmark programs on all supported CPU frequencies.
- measures the power and elapsed time of the benchmarks.
- performs multiple liner regression analysis to determine the coefficients A, B, C, D, E and F.
- generates coefficient data and places it in the database (the table TLSF_EnergyCoefficients).
- invokes other scripts for energy initialization (as performed by the system administrator).

initialize_eas

Initialization script to generate coefficient data for automatic CPU frequency selection.

Synopsis

```
initialize_eas [ -s {rsh | ssh | xdsh} ] -n node_list_file | -a new_node_list_file [ -f default_frequency ] -c cluster_name -d  
benchmark_dir  
initialize_eas [ -s {rsh | ssh | xdsh} ] -n node_list_file [-f default_frequency]  
initialize_eas -l -c cluster_name  
initialize_eas [-h | -V]
```

Description

The script (**initialize_eas**) can be run several times with different default CPU frequencies each time to generate several coefficient data groups before starting the LSF cluster. The default CPU frequency can be set using the utility **initialize_eas -f**.

Output data can be found in the following locations:

- /etc/energy/failed_node_list
- /etc/energy/out.[hostname]
- /etc/energy/investigation/investigation.[hostname]
- /etc/energy/coefficients/out.[hostname]

Note: The initialization utility must be configured by the system administrator; it requires super user authority.

Important: Run the script as root.

Important: Run the script on the management candidate host, which must be connected to a MySQL database.

Note: Before running the script, set up the remote execution command: rsh / ssh / xdsh

Usage

- h
Provides extended help information.
- V
Displays the name of the command, release number, and lowest level of the operating system to run this release.
- s
rsh | ssh | xdsh
Specifies which remote execution command will be used to run the energy initialization commands on the remote node.
The default command is rsh.
- d
benchmark_dir
Specifies the location of the energy benchmarks.
- f
default_frequency
Specifies the default CPU frequency (GHz, MHz, or KHz). The default is GHz.
- n
node_list_file
Specifies the compute nodes that need to run the benchmarks. Each host should be on one line in the file.
- a
new_node_list_file
Specifies the new nodes that need to be added in the cluster. Each host should be on one line in the file.
- c
cluster_name
Specifies the cluster name used to generate coefficient data.

-l

load coefficient data into database.

Results

The result of initialize_eas is two new tables in the database, one for the coefficients and one for the energy policy tag:

```
CREATE TABLE IF NOT EXISTS TLSF_EnergyCoefficients (
  frequency INTEGER NOT NULL, default_frequency INTEGER NOT NULL, cluster_name VARCHAR(40)
  BINARY NOT NULL, factor_a DOUBLE NOT NULL,
  factor_b DOUBLE NOT NULL,
  factor_c DOUBLE NOT NULL,
  factor_d DOUBLE NOT NULL,
  factor_e DOUBLE NOT NULL,
  factor_f DOUBLE NOT NULL,
  KEY (frequency, cluster_name, default_frequency),
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS TLSF_EnergyPolicyTag (
  energy_tag_name VARCHAR(256) BINARY NOT NULL,
  user_name VARCHAR(256) BINARY NOT NULL,
  default_frequency INTEGER NOT NULL,
  frequency INTEGER NOT NULL,
  cluster_name VARCHAR(40) BINARY NOT NULL,
  job_id VARCHAR(1024) BINARY NOT NULL,
  predict_power DOUBLE NOT NULL,
  energy_saving_pct DOUBLE NOT NULL,
  predict_elapse_time INTEGER NOT NULL,
  _degrad_pct DOUBLE NOT NULL,
  PRIMARY KEY (energy_tag_name, user_name, frequency, default_frequency, cluster_name),
) ENGINE = InnoDB;
```

Creating an energy policy tag

An energy policy tag is created by submitting jobs. The job runs using the default CPU frequency. When the job is finished, LSF collects the following information and adds it to the energy policy tag:

- Energy usage
- Job run time
- GIPS (giga instructions per second) for each computing node.
- GBS (giga bytes per second) for each computing node.

Important: Jobs generating an energy policy tag require exclusive use of the host. Therefore, the command `bsub -x` must be used.

The energy policy tag name is specified using the `esub` command when a job is submitted for the first time. For example:

```
bsub -x -a "eas(tag1,create)" sleep 10
```

Based on the data collected from a job and the coefficient data (which is collected using Benchmarking applications) LSF generates an energy policy tag using a prediction method. Using this energy policy tag, you can create an energy policy, specifying what CPU frequency LSF should use for each job.

Two steps are involved in creating a job energy policy tag:

1. Generate energy policy tag - Run the job in the default CPU frequency. When the job is done, LSF provides the energy consumption for the default frequency and estimates the performance degradation for each supported frequency. An energy policy tag name is generated for the job. You may run the job more than once, using different default CPU frequencies to see a variety of results.
2. Automatically select CPU frequency – The same job is submitted again with the same energy policy tag name. LSF will choose the best suitable frequency for the job based on the energy policy tag, user specified energy policy and settings in the global performance threshold file.

To support energy policy tag generation and to enable the automatic select CPU frequency feature, the following parameters (in `lsf.conf`) must be defined:

- LSF_MANAGE_FREQUENCY=HOST
- LSF_COLLECT_ENERGY_USAGE=Y
- LSF_DEFAULT_FREQUENCY

For the automatic select CPU frequency feature, you must also define the `lsb.threshold` configuration file, using the energy tags.

- [Energy policy tag format](#)
A job's energy policy tag identifies the energy data for a specific job. With the energy tag, LSF can decide which frequency should be used to run the job with minimal performance degradation.
- [Generate an energy policy tag](#)
LSF provides `esub.eas` to accept the energy policy tag and the energy policy parameters.
- [Enable automatic CPU frequency selection](#)
To enable automatic CPU frequency selection, there are three requirements, after completing the configuration:

Energy policy tag format

A job's energy policy tag identifies the energy data for a specific job. With the energy tag, LSF can decide which frequency should be used to run the job with minimal performance degradation.

The energy policy tag includes energy data such as energy usage and the run time in the default CPU frequency, the estimated energy consumption, the run time in other frequencies, and the percentage of performance degradation and power.

The energy policy tag is provided by the user in the `esub` parameter; its content is generated when running the job and will be used for automatically selecting a CPU frequency. The energy policy tag is saved into a MySQL database / xCat MySQL database.

It is important for each user to have their own energy policy tag for their job, since all job data may vary depending on the industry program, parameters, environment, and input data. Even the same job with the same input data from different users could get different results, depending on the parameters and environment.

The user who submits the job should keep the energy tag name unique for his or her jobs. In order to ensure the tag is unique for all the users, LSF will add the user name of the user to the tag name specified in the `esub` parameter.

The energy tag name format is **username.tagname**

where:

- **username** - the user name who generate the energy tag
- **tagname** - the identifier set by the user for the job in `esub` parameter

Valid characters for the **tagname** identifier include a ~ z, A ~ Z, 0 ~ 9 and “_” and the maximum length of the name is 256 bytes.

Generate an energy policy tag

LSF provides `esub.eas` to accept the energy policy tag and the energy policy parameters.

The energy policy should `minimize_energy`, `minimize_time` or `create`.

```
esub.eas [username.]tagname policy
```

- **username**: User generating the energy tag.
- **tagname**: Maximum length of the tag name is 256 bytes. Valid characters include upper and lower case letters (a-z, A-Z), numbers (0-9), and underscore (_).
- **policy**: Specify `minimize_energy`, `minimize_time`, or `create`

For example:

```
bsub -a "eas([userA.]long_running_job1, create)"
```

To generate a new tag, specify “create” as the second parameter. LSF will generate related data for this energy policy tag.

Note: Users can generate tags only for themselves.

The create tag job will run under the default CPU frequency and generate a tag. If there are several jobs with the same new energy tag name, the first done job will be used to generate the energy policy tag.

LSF generates the energy policy tag for a job to identify the job run time, power usage, estimated run time with other CPU frequencies and estimated performance degradation percentage.

LSF then uses a power usage and run time estimation formula to predict the job performance degradation when running with lower CPU frequencies. The power and run time predictions are based on the hardware counters LSF collected when the job ran with the default CPU frequency.

Important: Predictions require that the job run on homogenous nodes (same CPUs, same COREs, and the same amount of memory); otherwise the prediction value will be incorrect. Also, predictions can only be performed for application that make full use of the compute node - using all of the CPU power in that node and each CPU should be at about 100% CPU usage.

Note: LSF will only create the energy tag if the job runs successfully. For JOB_INCLUDE_POSTPROC=Y, the job should run post script success return JOB_STAT_PDONE. For JOB_INCLUDE_POSTPROC=N, the job should run success return JOB_STAT_DONE.

Note: When generating an energy policy tag, do not include pre/post execution commands with the job, or predictions may not be accurate.

Note: If a job has been in the UNKNOWN state, the runtime used for the tag may not be consistent with the job’s actual RUNTIME, since the sbatchd connection with mbatchd was lost and the job was finished before sbatchd could report the job was finished to mbatchd.

Note: The minimum run time for a job to generate an energy policy tag is one (1) second since the prediction runtime unit is in seconds (any job lasting less than one second will not generate a tag). Therefore, tag generation is only suitable for long running jobs. You may not receive an accurate prediction for short running jobs (several seconds).

Enable automatic CPU frequency selection

To enable automatic CPU frequency selection, there are three requirements, after completing the configuration:

1. A global (cluster-level) performance threshold configuration file ([lsb.threshold](#)) is required, to control a minimize energy or running time policy.
2. Three parameters must be set in the lsf.conf file. LSF_MANAGE_FREQUENCY=HOST, LSF_COLLECT_ENERGY_USAGE=Y, and LSF_DEFAULT_FREQUENCY
3. Coefficient data must be generated and saved in database.

IBM Spectrum LSF multicluster capability

Learn how to use and manage the IBM Spectrum LSF multicluster capability to share resources across your LSF clusters.

- [Overview of the LSF multicluster capability](#)
Learn how multiple LSF clusters can solve problems related to ease of administration, scalability, and sites in different geographic locations. Use the LSF multicluster capability to share resources between clusters through resource leasing or job forwarding.
- [Set up IBM Spectrum LSF multicluster capability](#)
Set up IBM Spectrum LSF multicluster capability, see system requirements and post-installation configuration procedures. Learn how to ensure the security of daemon communication and configure authentication between clusters.
- [IBM Spectrum LSF multicluster capability job forwarding model](#)
In the job forwarding model, a cluster that is starving for resources sends jobs over to the cluster that has resources to spare.
- [IBM Spectrum LSF multicluster capability resource leasing model](#)
In the resource leasing model, two clusters agree that one cluster will borrow resources from the other, taking control of

the other cluster's resources.

Overview of the LSF multicluster capability

Learn how multiple LSF clusters can solve problems related to ease of administration, scalability, and sites in different geographic locations. Use the LSF multicluster capability to share resources between clusters through resource leasing or job forwarding.

Tip: To use application profiles in an LSF multicluster environment, configure the application profile on both the submission and execution clusters. The application profile option that is specified for a job will be forwarded with the job. If the application profile does not exist in the execution cluster, the job pends with a special reason.

Use the **bclusters -app** command to see the application profiles configured in a remote cluster.

- **Benefits of the LSF multicluster capability**

Within an organization, sites can have separate, independently managed LSF clusters. Having multiple LSF clusters could solve some common problems, such as ease of administration, different geographic locations, or scalability

- **Two multicluster models**

There are two different ways to share resources between clusters using the LSF multicluster capability: using the job forwarding model, or the resource leasing model. Additionally, these models can be combined, for example, `Cluster1` forwards jobs to `Cluster2` using the job forwarding model, and `Cluster2` borrows resources from `Cluster3` using the resource leasing model.

Benefits of the LSF multicluster capability

Within an organization, sites can have separate, independently managed LSF clusters. Having multiple LSF clusters could solve some common problems, such as ease of administration, different geographic locations, or scalability

When you have more than one cluster, it is desirable to allow the clusters to cooperate to reap the following benefits of global load sharing:

- Access to a diverse collection of computing resources
- Enterprise grid computing becomes a reality
- Get better performance and computing capabilities
- Use idle machines to process jobs
- Use multiple machines to process a single parallel job
- Increase user productivity
- Add resources anywhere and make them available to the entire organization
- Plan computing resources globally based on total computing demand
- Increase computing power in an economical way

A multicluster environment enables a large organization to form multiple cooperating clusters of computers so that load sharing happens not only within clusters, but also among them. A multicluster environment enables:

- Load sharing across a large numbers of hosts
- Co-scheduling among clusters: Job forwarding scheduler considers remote cluster and queue availability and load before forwarding jobs.
- Resource ownership and autonomy to be enforced
- Non-shared user accounts and file systems to be supported
- Communication limitations among the clusters to be taken into consideration in job scheduling

Two multicluster models

There are two different ways to share resources between clusters using the LSF multicluster capability: using the job forwarding model, or the resource leasing model. Additionally, these models can be combined, for example, `Cluster1`

forwards jobs to `Cluster2` using the job forwarding model, and `Cluster2` borrows resources from `Cluster3` using the resource leasing model.

Job forwarding model

In this model, the cluster that is starving for resources sends jobs over to the cluster that has resources to spare. To work together, two clusters must set up compatible send-jobs and receive-jobs queues.

With this model, scheduling of multicluster jobs is a process with two scheduling phases: the submission cluster selects a suitable remote receive-jobs queue, and forwards the job to it; then the execution cluster selects a suitable host and dispatches the job to it. This method automatically favors local hosts; a multicluster send-jobs queue always attempts to find a suitable local host before considering a receive-jobs queue in another cluster.

Resource leasing model

In this model, the cluster that is starving for resources takes resources away from the cluster that has resources to spare. To work together, the provider cluster must “export” resources to the consumer, and the consumer cluster must configure a queue to use those resources.

In this model, each cluster schedules work on a single system image, which includes both borrowed hosts and local hosts.

Note: As of Fix Pack 11, the resource leasing model is deprecated.

Choosing a model

Consider your own goals and priorities when choosing the best resource-sharing model for your site:

- The job forwarding model can make resources available to jobs from multiple clusters, this flexibility allows maximum throughput when each cluster’s resource usage fluctuates. The resource leasing model can allow one cluster exclusive control of a dedicated resource, this can be more efficient when there is a steady amount of work.
- The lease model is the most transparent to users and supports the same scheduling features as a single cluster.
- The job forwarding model has a single point of administration, while the lease model shares administration between provider and consumer clusters.

Set up IBM Spectrum LSF multicluster capability

Set up IBM Spectrum LSF multicluster capability, see system requirements and post-installation configuration procedures. Learn how to ensure the security of daemon communication and configure authentication between clusters.

- [Setup overview](#)
- [Non-uniform name spaces](#)
- [Restricted awareness of remote clusters](#)
- [Security of daemon communication](#)
- [Authentication between clusters](#)
- [Resource usage updates for MultiCluster jobs](#)
- [Global limits for job resource allocations](#)
Configure global limit scheduling to apply resource allocation limits to multiple clusters.
- [Global job IDs for forwarding and forwarded clusters using LSF multicluster capability](#)
Global job IDs allow an LSF multicluster environment to use the same job IDs between the forwarding and forwarded clusters, keeping the IDs uniform. These global job IDs are unique. To guarantee unique job IDs, starting in Fix Pack 14, LSF introduces indexes for clusters, so that each job submitted from the cluster includes an index to the ending digits of the job ID (for example, job ID 100 with an index value of 22 will have a global job ID of 10022).
- [Multicluster information cache](#)

Setup overview

- [Multiclusterc system requirements](#)

The setup procedures will guide you through configuring your system to meet each requirement. However, you might find it helpful to understand the system requirements before you begin.

- [Installation and configuration procedures for LSF multiclusterc](#)

There are six major tasks involved for installing and configuring LSF multiclusterc.

- [Install LSF multiclusterc](#)

Multiclusterc files are automatically installed by LSF's regular setup program (**lsfinstall**). Install LSF and make sure each cluster works properly as a standalone cluster before you proceed to configure multiclusterc capabilities.

- [Set common ports](#)

Participating clusters must use the same port numbers for the daemons LIM, RES, MBD, and SBD.

Multiclusterc system requirements

The setup procedures will guide you through configuring your system to meet each requirement. However, you might find it helpful to understand the system requirements before you begin.

Requirements to install a multiclusterc environment

You can use multiclusterc to link two or more LSF clusters. Then, the participating clusters can be configured to share resources.

multiclusterc files are automatically installed by LSF's regular Setup program (**lsfinstall**). Install LSF and make sure each cluster works properly as a standalone cluster before you proceed to configure multiclusterc.

Requirements for multiclusterc communication between 2 clusters

- The local management host must be configured to communicate with the remote cluster:
 - The local cluster can only communicate with other clusters if they are specified in `lsf.shared`.
 - If the `RemoteClusters` section in `lsf.cluster.cluster_name` is defined, the local cluster has a list of recognized clusters, and is only aware of those clusters.
- The local management host must be able to contact the management host of the remote cluster:
 - The valid management host list for remote clusters is used to locate the current management host on that cluster and ensure that any remote host is a valid management host for its cluster. The valid management host list is defined in `LSF_MASTER_LIST` in `lsf.conf`.
 - Participating clusters must use the same port numbers for the LSF daemons **LIM**, **RES**, **sbatchd** and **mbatchd**.

Requirements for resource sharing between 2 clusters

- The local cluster must use the same resource definitions as the remote cluster:
 - Clusters should have common definitions of host types, host models, and resources. Each cluster finds this information in `lsf.shared`.
- A host cannot belong to more than one cluster.
- The local cluster and the remote cluster must have compatible configurations, with the resource owner sharing the resource and the resource consumer seeking to use the resource.

Requirements for jobs to run across clusters

- Users must have a valid user account in each cluster.
 - By default, LSF expects that the user accounts will have the same name in each cluster. If clusters do not share a file system and common user name space, you can configure account mapping.
- LSF must be able to transfer job files and data files between clusters.
- Dynamic IP addressing is not supported across clusters. LSF hosts require a fixed IP address to communicate with a host that belongs to another cluster.

Installation and configuration procedures for LSF multicluster

There are six major tasks involved for installing and configuring LSF multicluster.

Procedure

1. Plan the cluster.
2. Establish communication between clusters.
3. Additional tasks that might be required to establish communication between clusters.
4. Test communication between clusters.
5. Establish resource sharing.
6. Optional tasks.

Plan the cluster (required)

Procedure

1. Read the [Overview of the LSF multicluster capability](#) to learn about how a multicluster environment can be useful to you.
2. Decide which clusters will participate. Read the [Setup overview](#) to learn about the issues that could prevent clusters from working together.
3. Decide which resources you want to share.
4. Decide how you will share the resources among clusters. Read about the various configuration options available in the [IBM Spectrum LSF multicluster capability job forwarding model](#) and the [IBM Spectrum LSF multicluster capability resource leasing model](#) and the configuration options common for each model.

Establish communication between clusters (required)

Procedure

1. For resource sharing to work between clusters, the clusters should have common definitions of host types, host models, and resources. Configure this information in `lsf.shared`.
2. To establish communication, clusters must be aware of other clusters and know how to contact other clusters. Add each cluster name and its management host name and management host candidate names to the Cluster section of `lsf.shared`.

Additional tasks that might be required to establish communication between clusters

Procedure

1. By default, LSF assumes a uniform user name space within a cluster and between clusters.
2. With multicluster, LSF daemons can use non-privileged ports. By default, LSF daemons in a multicluster environment use privileged port authentication.
3. When using multicluster, the external client to server authentication method that is used must be the same for all clusters. The client to server external authentication method is defined in `LSF_AUTH` in `lsf.conf`.

Test communication between clusters (required)

Procedure

1. Restart each cluster using the `lsadmin` and `badmin` commands:


```
% bctrld restart lim all
% badmin mbdrestart
```

2. To verify that multicluster is enabled, run **lsclusters** and **bclusters**:

```
% lsclusters
CLUSTER_NAME    STATUS    MASTER_HOST    ADMIN    HOSTS    SERVERS
cluster1        ok        hostA          admin1    1        1
cluster2        ok        hostD          admin2    3        3

% bcusters
[Remote Batch Information]
No local queue sending/receiving jobs from remote clusters
```

Establish resource sharing (required)

Procedure

1. Run a simple test of resource sharing (optional).
2. Configure resource-sharing policies between clusters.

Optional tasks

Procedure

1. By default, all the clusters in a multicluster environment are aware of all the other clusters. This makes it possible for clusters to share resources or information. You can restrict awareness of remote clusters at the cluster level.
2. With multicluster, LSF daemons can use non-privileged ports (by default, LSF daemons in a multicluster environment use privileged port authentication). You can also choose the method of daemon authentication.
3. When a local cluster requests load or host information from a remote cluster, the information is cached. If the local cluster is required to display the same information again, LSF displays the cached information, unless the cache has expired. The expiry period for cached information is configurable.
4. The default configuration of LSF is that clusters share information about the resources used by other clusters, and the information is updated every five minutes by the execution or provider cluster. You can disable the feature or modify how often multicluster resource usage is updated.
5. To learn about optional features related to each configuration model, read about the various configuration options available in the multicluster job forwarding model and the multicluster resource leasing model.

Install LSF multicluster

Multicluster files are automatically installed by LSF's regular setup program (**lsfinstall**). Install LSF and make sure each cluster works properly as a standalone cluster before you proceed to configure multicluster capabilities.

Set common ports

Participating clusters must use the same port numbers for the daemons LIM, RES, MBD, and SBD.

By default, all clusters have identical settings, as shown:

```
LSF_LIM_PORT=7869
LSF_RES_PORT=6878
LSB_MBD_PORT=6881
LSB_SBD_PORT=6882
```

LSF_LIM_PORT change

The default for LSF_LIM_PORT changed in LSF version 7.0 to accommodate IBM® EGO default port configuration. On EGO, default ports start with **lim** at 7869, and are numbered consecutively for the EGO **pem**, **vemkd**, and **egosc**

daemons.

This is different from previous LSF releases where the default LSF_LIM_PORT was 6879. LSF **res**, **sbatchd**, and **mbatchd** continue to use the default pre-version 7.0 ports 6878, 6881, and 6882.

Upgrade installation preserves existing port settings for **lim**, **res**, **sbatchd**, and **mbatchd**. EGO **pem**, **vermkd**, and **egosc** use default EGO ports starting at 7870, if they do not conflict with existing **lim**, **res**, **sbatchd**, and **mbatchd** ports.

Troubleshooting

To check your port numbers, check the LSF_TOP/conf/lsf.conf file in each cluster. (LSF_TOP is the LSF installation directory. On UNIX, this is defined in the install.config file). Make sure you have identical settings in each cluster for the following parameters:

- LSF_LIM_PORT
- LSF_RES_PORT
- LSB_MBD_PORT
- LSB_SBD_PORT

Set common resource definitions

For resource sharing to work between clusters, the clusters should have common definitions of host types, host models, and resources. Each cluster finds this information in lsf.shared, so the best way to configure multicluster is to make sure lsf.shared is identical for each cluster. If you do not have a shared file system, replicate lsf.shared across all clusters.

If it is impossible to have the same lsf.shared file for all clusters, LSF still allows for different host types, host models or resources in the lsf.shared file. In this case, the general rules for MC resource configuration are:

1. Local cluster information overrides remote cluster information (host type, host model, or resource attributes and order of specification in configuration files).
2. The local cluster ignores remote cluster configuration if the remote type/host model/resource does not exist in local cluster.

Define participating clusters and valid management hosts

To enable multicluster, define all participating clusters in the Cluster section of the LSF_TOP/conf/lsf.shared file.

1. For ClusterName, specify the name of each participating cluster. On UNIX, each cluster name is defined by LSF_CLUSTER_NAME in the install.config file.
2. For Servers, specify the management host and optionally candidate management hosts for the cluster. A cluster will not participate in multicluster resource sharing unless its current management host is listed here.

Example

```
Begin Cluster
ClusterName Servers
Cluster1      (hostA hostB)
Cluster2      (hostD)
End Cluster
```

In this example, hostA should be the management host of Cluster1 with hostB as the backup, and hostD should be the management host of Cluster2. If the management host fails in Cluster1, MultiCluster will still work because the backup management host is also listed here. However, if the management host fails in Cluster2, MultiCluster will not recognize any other host as the management host, so Cluster2 will no longer participate in MultiCluster resource sharing.

Non-uniform name spaces

By default, LSF assumes a uniform user name space within a cluster and between clusters.

User account mapping

To support the execution of batch jobs across non-uniform user name spaces between clusters, LSF allows user account mapping.

File transfer

By default, LSF uses **lsrnp** for file transfer (**bsub -f** option),

Tip:

The lsrnp utility depends on a uniform user ID in different clusters.

Account mapping between clusters

By default, LSF assumes a uniform user name space within a cluster and between clusters. To support the execution of batch jobs across non-uniform user name spaces between clusters, LSF allows user account mapping.

For a job submitted by one user account in one cluster to run under a different user account on a host that belongs to a remote cluster, both the local and remote clusters must have the account mapping properly configured. System-level account mapping is configured by the LSF administrator, while user-level account mapping can be configured by LSF users.

System-level account mapping

You must be an LSF administrator to configure system level account mapping.

System-level account mapping is defined in the `UserMap` section of `lsb.users`. The submission cluster proposes a set of user mappings (defined using the keyword `export`) and the execution cluster accepts a set of user mappings (defined using the keyword `import`). For a user's job to run, the mapping must be both proposed and accepted.

Example

lsb.users on cluster1:

```
Begin UserMap
LOCAL      REMOTE      DIRECTION
user1      user2@cluster2    export
user3      (user4@cluster2 user6@cluster2)    export
End UserMap
```

lsb.users on cluster2:

```
Begin UserMap
LOCAL      REMOTE      DIRECTION
user2      user1@cluster1    import
(user6 user8)    user3@cluster1    import
End UserMap
```

Cluster1 configures user1 to run jobs as user2 in cluster2, and user3 to run jobs as user4 or user6 in cluster2.

Cluster2 configures user1 from cluster1 to run jobs as user2, and user3 from cluster1 to run jobs as user6 or user8.

Only mappings configured in both clusters work. The common account mappings are for user1 to run jobs as user2, and for user3 to run jobs as user6. Therefore, these mappings work, but the mappings of user3 to users 4 and 8 are only half-done and so do not work.

- [User-level account mapping](#)

User-level account mapping

Procedure

To set up your own account mapping, set up a `.lsfhosts` file in your home directory with Owner Read-Write permissions only. Do not give other users and groups permissions on this file.

Tip:

Account mapping can specify cluster names in place of host names.

Example #1

You have two accounts: `user1` on `cluster1`, and `user2` on `cluster2`. To run jobs in either cluster, configure `.lsfhosts` as shown.

On each host in `cluster1`:

```
% cat ~user1/.lsfhosts
cluster2 user2
```

On each host in `cluster2`:

```
% cat ~user2/.lsfhosts
cluster1 user1
```

Example #2

You have the account `user1` on `cluster1`, and want to run jobs on `cluster2` under the `lsfquest` account. Configure `.lsfhosts` as shown.

On each host in `cluster1`:

```
% cat ~user1/.lsfhosts
cluster2 lsfquest send
```

On each host in `cluster2`:

```
% cat ~lsfquest/.lsfhosts
cluster1 user1 recv
```

Example #3

You have a uniform account name (`user2`) on all hosts in `cluster2`, and a uniform account name (`user1`) on all hosts in `cluster1` except `hostX`. On `hostX`, you have the account name `user99`.

To use both clusters transparently, configure `.lsfhosts` in your home directories on different hosts as shown.

On `hostX` in `cluster1`:

```
% cat ~user99/.lsfhosts
cluster1    user1
hostX       user99
cluster2    user2
```

On every other host in `cluster1`:

```
% cat ~user1/.lsfhosts
cluster2    user2
hostX       user99
```

On each host in `cluster2`:

```
% cat ~user2/.lsfhosts
cluster1    user1
hostX       user99
```

Restricted awareness of remote clusters

By default, all the clusters in a MultiCluster environment are aware of all the other clusters. This makes it possible for clusters to share resources or information when you configure MultiCluster links between them.

You can restrict awareness of remote clusters at the cluster level, by listing which of the other clusters in the MultiCluster environment are allowed to interact with the local cluster. In this case, the local cluster cannot display information about unrecognized clusters and does not participate in MultiCluster resource sharing with unrecognized clusters.

How it works

By default, the local cluster can obtain information about all other clusters specified in `lsf.shared`. The default behavior of RES is to accept requests from all the clusters in `lsf.shared`.

If the `RemoteClusters` section in `lsf.cluster.cluster_name` is defined, the local cluster has a list of recognized clusters, and is only aware of those clusters. The local cluster is not aware of the other clusters in the MultiCluster environment:

- The cluster does not forward jobs to unrecognized clusters, even if a local queue is configured to do so.
- The cluster does not borrow resources from unrecognized clusters, even if the remote cluster has exported the resources.
- The cluster does not export resources to unrecognized clusters, even if the local resource export section is configured to do so.
- The cluster does not receive jobs from unrecognized clusters, even if a local queue is configured to do so.
- The cluster cannot view information about unrecognized clusters.

However, remote clusters might still be aware of this cluster:

- Unrecognized clusters can view information about this cluster.
- Unrecognized clusters can send MultiCluster jobs to this cluster (they will be rejected, even if a local queue is configured to accept them).
- Unrecognized clusters can export resources to this cluster (this cluster will not use the resources, even if a local queue is configured to import them).

Example

This example illustrates how the `RemoteClusters` list works.

The MultiCluster environment consists of 4 clusters with a common `lsf.shared`:

```
CLUSTERS
cluster1
cluster2
cluster3
cluster4
```

In addition, `cluster2` is configured with a `RemoteClusters` list in `lsf.cluster.cluster_name`:

```
Begin RemoteClusters
CLUSTERNAME
cluster3
cluster4
End RemoteClusters
```

Because of the `RemoteClusters` list, local applications in `cluster2` are aware of `cluster3` and `cluster4`, but not `cluster1`. For example, if you view information or configure queues using the keyword `all`, LSF will behave as if you specified the list of recognized clusters instead of all clusters in `lsf.shared`.

- [Add or modify RemoteClusters list](#)

Add or modify RemoteClusters list

About this task

You must have cluster administrator privileges in the local cluster to perform this task.

Procedure

1. Open `lsf.cluster.cluster_name` of the local cluster.
2. If it does not already exist, create the `RemoteClusters` section as shown:

```
Begin RemoteClusters
CLUSTERNAME
...
End RemoteClusters
```

3. Edit the `RemoteClusters` section. Under the heading `CLUSTERNAME`, specify the names of the remote clusters that you want the local cluster recognize.
These clusters must also be listed in `lsf.shared`, so the `RemoteClusters` list is always a subset of the clusters list in `lsf.shared`.

Security of daemon communication

About this task

LSF daemons in a MultiCluster environment use privileged port authentication by default. LSF `mbatchd` and `lim` daemons can be configured to communicate over non-privileged ports. If disabling the privileged port authentication makes you concerned about the security of daemon authentication, you can use an **eaauth** program to enable any method of authentication for secure communication between clusters.

Configuring an **eaauth** or setting `LSF_MC_NON_PRIVILEGED_PORTS` to `N` disables privileged port authentication.

Note:

Windows does not use privileged ports for authentication.

Requirements

- All clusters must be configured to use non-privileged ports for LSF daemon communication.
- If you use a firewall, it must accept incoming communication from non-privileged source ports if the destination ports are the LIM port configured `LSF_LIM_PORT` in `lsf.conf` and **mbatchd** port configured in `LSB_MBD_PORT` in `lsf.conf`.
- If you use a firewall, it must allow outgoing communication from non-privileged source ports to non-privileged destination ports.

Procedure

1. To make LSF daemons use non-privileged ports, edit `lsf.conf` in every cluster as shown:
`LSF_MC_NON_PRIVILEGED_PORTS=Y`
2. To make the changes take effect, restart the management host LIM and MBD in every cluster, and the LIM on all management host candidates. For example, if a cluster's management host is `hostA` and management host candidate is `hostB`, run the following commands in that cluster:

```
bctrld restart lim hostA
bctrld restart lim hostB
badmin mbdrestart
```

Authentication between clusters

About this task

Because this is configured for individual clusters, not globally, different cluster pairs can use different systems of authentication. You use a different **eaauth** program for each different authentication mechanism.

Procedure

- For extra security, you can use any method of external authentication between any two clusters in the MultiCluster grid.
- If no common external authentication method has been configured, two clusters communicate with the default security, which is privileged port authentication.

eauth executables

About this task

Contact IBM for more information about the **eauth** programs that IBM distributes to allow LSF to work with different security mechanisms. If you already have an **eauth** that works with LSF for daemon authentication within the cluster, use a copy of it.

If different clusters use different methods of authentication, set up multiple **eauth** programs.

Procedure

1. Copy the corresponding **eauth** program to LSF_SERVERDIR.
2. Name the **eauth** program `eauth.method_name`.
If you happen to use the same **eauth** program for daemon authentication within the cluster, you should have two copies, one named **eauth** (used by LSF) and one named `eauth.method_name` (used by MultiCluster).

MultiCluster configuration

Procedure

1. Edit the `lsf.cluster.cluster_name RemoteClusters` section.
If the cluster does not already include a RemoteClusters list, you must add it now. To maintain the existing compatibility, specify all remote clusters in the list, even if the preferred method of authentication is the default method.
2. If necessary, add the AUTH column to the RemoteClusters section.
3. For each remote cluster, specify the preferred authentication method. Set AUTH to `method_name` (using the same method name that identifies the corresponding **eauth** program). For default behavior, specify a dash (-).
4. To make the changes take effect in a working cluster, run the following commands:

```
bctrld restart lim management_host  
bctrld restart lim management_candidate_host  
badmin mbdrestart
```

Repeat the steps for each cluster that will use external authentication, making sure that the configurations of paired-up clusters match.

Configuration example

In this example, Cluster1 and Cluster2 use Kerberos authentication with each other, but not with Cluster3. It does not matter how Cluster3 is configured, because without a common authentication method between clusters no communication occurs.

RECV_FROM set to Y indicates the local cluster accepts parallel jobs that originate in a remote cluster.

EQUIV set to Y changes the default behavior of LSF commands and utilities and causes them to automatically return load (`lsload(1)`), host (`lshosts(1)`), or placement (`lsplace(1)`) information about the remote cluster as well as the local cluster, even when you don't specify a cluster name.

Cluster1

```
lsf.cluster.cluster1:
```

```
Begin RemoteClusters  
CLUSTERNAME EQUIV CACHE_INTERVAL RECV_FROM AUTH  
cluster2      Y      60             Y      KRB  
cluster3      N      30             N      -  
End RemoteClusters
```

LSF_SERVERDIR in Cluster1 includes an **eauth** executable named `eauth.KRB`.

Cluster2

lsf.cluster.cluster2:

Begin RemoteClusters

CLUSTERNAME	EQUIV	CACHE_INTERVAL	RECV_FROM	AUTH
cluster1	Y	60	Y	KRB
cluster3	N	30	N	-

End RemoteClusters

LSF_SERVERDIR in Cluster2 includes an **eauth** executable named eauth.KRB.

Resource usage updates for MultiCluster jobs

Upon installation, the default configuration of LSF is that clusters share information about the resources used by other clusters, and the information is updated every 5 minutes by the execution or provider cluster. You can disable the feature or modify how often MultiCluster resource usage is updated. Depending on load, updating the information very frequently can affect the performance of LSF.

Configure resource usage updates for MultiCluster jobs

To change the timing of resource usage updating between clusters, set MC_RUSAGE_UPDATE_INTERVAL in lsb.params in the execution or provider cluster. Specify how often to update the information to the submission or consumer cluster, in seconds.

To disable LSF resource usage updating between clusters, specify zero:

MC_RUSAGE_UPDATE_INTERVAL=0

Global limits for job resource allocations

Configure global limit scheduling to apply resource allocation limits to multiple clusters.

Before you begin

Batch features for the LSF multicluster capability, job forward mode, or lease mode are not required to use global limits. Therefore, you can use global limit scheduling if you have multiple LSF clusters without using the LSF multicluster capability.

However, job group limits, which are needed to limit the forwarded jobs, requires the use of the job forward mode for the LSF multicluster capability.

About this task

When using global limit scheduling, job group limits are still applied when the jobs are forwarded to another cluster.

Global limits support both the compete and evenly distribution policies.

- [Configure global limits](#)
Enable global limits by specifying the GLOBAL_LIMITS parameter in the lsb.params file. Define global limits in the Limit section of the lsb.globalpolicies file.
- [View the global limits](#)
The **blimits -gl** command option displays global limits.

Configure global limits

Enable global limits by specifying the GLOBAL_LIMITS parameter in the lsb.params file. Define global limits in the Limit section of the lsb.globalpolicies file.

Before you begin

To use global limits, you must ensure that the LSB_GPD_PORT and LSB_GPD_CLUSTER parameters are defined correctly in the lsf.conf file for the **gpolicyd** daemon.

Procedure

1. Edit the lsb.params file and define the GLOBAL_LIMITS parameter to enable global limit scheduling among all clusters.

```
GLOBAL_LIMITS=Y
```

2. Edit the lsb.globalpolicies file and specify the global resource allocation limits in the Limit section. Specify global resource allocations the same way you would specify local resource allocation limits in the Limit sections of the lsb.resources file.

For example, to configure global limits using the horizontal format,

```
Begin Limit
NAME           = limit1
USERS          = ugroup1 ~user1
JOBS           = 12
End Limit
```

View the global limits

The **blimits -gl** command option displays global limits.

Procedure

1. Run the **blimits -gl** command option to display the current usage of global limits. LSF calls the **gpolicyd** daemon to get the global limits usage information.
2. Run the **blimits -gl -c** command option to display the global limits configuration.

```
$ blimits -gl -c
Begin Limit
NAME           = limit1
USERS          = ugroup1 ~user1
JOBS           = 12
End Limit
```

Global job IDs for forwarding and forwarded clusters using LSF multicluster capability

Global job IDs allow an LSF multicluster environment to use the same job IDs between the forwarding and forwarded clusters, keeping the IDs uniform. These global job IDs are unique. To guarantee unique job IDs, starting in Fix Pack 14, LSF introduces indexes for clusters, so that each job submitted from the cluster includes an index to the ending digits of the job ID (for example, job ID 100 with an index value of 22 will have a global job ID of 10022).

About this task

To configure global job IDs for your LSF multicluster environment, add an index column to the `lsf.shared` configuration file, then as a best practice, increase both the `MAX_JOBID` value in the `lsb.params` file (to 99999999) and the `LSB_JOBID_DISP_LENGTH` value in the `lsf.conf` file (to 8).

Procedure

1. Log on to any host in the cluster as the LSF administrator.
2. Edit the [lsf.shared](#) configuration file:
 - a. Add new column called **Index** to the **Cluster** section and assign indexes for each cluster. The index can be a number from 1-99, and must be unique to the other clusters within the same cluster group.
For example:

```
Begin Cluster
ClusterName Servers      Index
cluster1    (hostA hostB)  1
cluster2    (hostD)    2
End Cluster
```

Tip: Typically, the `lsf.shared` file is centrally located so that any changes can be accessed by all clusters; however, if you do not have a shared system, then ensure that you update the `lsf.shared` file on each cluster.

With the **Index** column set, the cluster reading the configuration generates global job IDs ending with the index. Also, when jobs are forwarded to the cluster, the cluster will try to match the job ID with the index configured. If the index does not match the job ID, the job will be rejected from the cluster. If the index matches the job ID, the job will then be accepted as the same job ID given from the forwarding cluster. If the job ID has already been used, the job ID will be changed to end in 00.

- b. Save the changes to the `lsf.shared` file.
 - c. Run **lsadmin reconfig** to reconfigure LIM.
 - d. Run **badmin reconfig** to reconfigure the **mbatchd** daemon.
3. To maximize the number of available global job IDs that LSF can assign, as a best practice, increase both the `MAX_JOBID` value in the `lsb.params` file (to 99999999) and the `LSB_JOBID_DISP_LENGTH` value in the `lsf.conf` file (to 8).
 4. To view the new index column, run the **lscluster cluster_name** command to see all configuration information for your cluster (including the new column).
Here is example output from running **lscluster hostA**:

CLUSTER_NAME	STATUS	MASTER_HOST	ADMIN	HOSTS	SERVERS	INDEX
cluster1	ok	hostA	jsmith	1	1	1

Related reference

- [MAX_JOBID](#)
- [lsb.params](#)
- [LSB_JOBID_DISP_LENGTH](#)
- [lsf.conf](#)

Multicluster information cache

When a local cluster LIM requests load or host information from a remote cluster using commands such as **lload** or **lshosts**, the information is cached. If the local cluster is required to display the same information again, LSF displays the cached information, unless the cache has expired.

The expiry period for cached LIM information is configurable, so you can view more up-to-date information if you don't mind connecting to the remote cluster more often.

It is more efficient to get information from a local cluster than from a remote cluster. Caching remote cluster information locally minimizes excessive communication between clusters.

Cache thresholds

The cache threshold is the maximum time that remote cluster information can remain in the local cache.

There are two cache thresholds, one for load information, and one for host information. The threshold for host information is always double the threshold for load information.

By default, cached load information expires after 60 seconds and cached host information expires after 120 seconds.

How it works

When a local cluster requests load or host information from a remote cluster, the information is cached by the local management host LIM.

When the local cluster is required to display the same information again, LSF evaluates the age of the information in the cache.

- If the information has been stored in the local cluster for longer than the specified time, LSF contacts the remote cluster again, updates the cache, and displays current information.
- If the age of the cached information is less than the threshold time, LSF displays the cached information.

Configure cache threshold

Set `CACHE_INTERVAL` in the `RemoteClusters` section of `lsf.cluster.cluster_name`, and specify the number of seconds to cache load information.

IBM Spectrum LSF multicluster capability job forwarding model

In the job forwarding model, a cluster that is starving for resources sends jobs over to the cluster that has resources to spare.

- [Job forwarding model overview](#)
- [Job scheduling under the job forwarding model](#)
- [Queue scheduling parameters under job forwarding model](#)
- [Advance reservations across clusters](#)
- [Special considerations under job forwarding model](#)
- [Queues for the LSF multicluster capability](#)

By default, clusters do not share resources, even if the LSF multicluster capability is installed. To enable job forwarding, enable queues for the LSF multicluster capability in both the submission and execution clusters.

- [Remote-only queues](#)
- [Request a specific cluster](#)
- [Remote cluster equivalency](#)
- [Remote Resources](#)
- [Remote queue workload job-forwarding scheduler](#)
- [Pre-exec retry threshold](#)
- [Retry threshold and suspend notification](#)
- [Pending MultiCluster job limit](#)
- [Update pending reason for MultiCluster jobs](#)
- [Remote timeout limit](#)
- [Enable job priority in MultiCluster job forward mode](#)
- [Enhance fair share calculation to include the job forwarding mode](#)

You can enhance the calculations for the fair share policies to account for the forwarding mode in the LSF multicluster capability, which charges the user priority for each job forwarded, providing a fair opportunity for all users to forward their jobs. This policy charges the priority of the user whose job is successfully forwarded to the remote execution cluster, and restores the charge for the user whose job is returned from the remote execution cluster.

Job forwarding model overview

In this model, the cluster that is starving for resources sends jobs over to the cluster that has resources to spare. Job status, pending reason, and resource usage are returned to the submission cluster. When the job is done, the exit code returns to the submission cluster.

Tracking

bhosts

By default, **bhosts** shows information about hosts and resources that are available to the local cluster and information about jobs that are scheduled by the local cluster.

bjobs

The **bjobs** command shows all jobs associated with hosts in the cluster, including MultiCluster jobs. Jobs from remote clusters can be identified by the FROM_HOST column, which shows the remote cluster name and the submission or consumer cluster job ID in the format *host_name@remote_cluster_name:remote_job_ID*.

If the MultiCluster job is running under the job forwarding model, the QUEUE column shows a local queue, but if the MultiCluster job is running under the resource leasing model, the name of the remote queue is shown in the format *queue_name@remote_cluster_name*.

You can use the local job ID and cluster name (for example, `bjobs 123@submission_cluster_name`) to see the job IDs for the submission, execution and lease clusters. For job arrays, the query syntax is `bjobs "submission_job_id[index]"@submission_cluster_name`.

Use **-w** or **-l** to prevent the MultiCluster information from being truncated.

Use **-fwd** from the submission cluster to filter output to display forwarded jobs, including the forwarded time and the name of the cluster to which the job was forwarded. **-fwd** can be used with other options to further filter the results. For example, **bjobs -fwd -r** displays only forwarded running jobs.

```
% bjobs -fwd
JOBID USER   STAT QUEUE   EXEC_HOST JOB_NAME   CLUSTER   FORWARD_TIME
123   lsufuser  RUN  queue1  hostC     sleep 1234 cluster3  Nov 29 14:08
```

The **-fwd** option cannot be used together with the following **bjobs** options: **-A**, **-d**, **-sla**, **-ss**.

When used with **-x**, the **-fwd** option shows forwarded job information only for exceptions configured in `lsb.queues` of the submission cluster. To see exceptions on the execution cluster, use **bjobs -m execution_cluster -x**.

When **bjobs -m** is followed by cluster name, a user message is displayed. The message is only about the host and host group. There is no cluster reference (**bjobs -m <cluster name> -fwd** does not display the job running on the cluster).

bclusters

Displays remote resource provider and consumer information, resource flow information, and connection status between the local and remote cluster.

Use **-app** to view available application profiles in remote clusters.

Information related to the job forwarding model is displayed under the heading Job Forwarding Information.

- **LOCAL_QUEUE**: Name of a local MultiCluster send-jobs or receive-jobs queue.
- **JOB_FLOW**: Indicates direction of job flow.
 - **send**
The local queue is a MultiCluster send-jobs queue (SNDJOBS_TO is defined in the local queue).
 - **recv**
The local queue is a MultiCluster receive-jobs queue (RCVJOBS_FROM is defined in the local queue).
- **REMOTE**: For send-jobs queues, shows the name of the receive-jobs queue in a remote cluster. For receive-jobs queues, always “-”.
- **CLUSTER**: For send-jobs queues, shows the name of the remote cluster containing the receive-jobs queue. For receive-jobs queues, shows the name of the remote cluster that can send jobs to the local queue.
- **STATUS**: Indicates the connection status between the local queue and remote queue.
 - **ok**

The two clusters can exchange information and the system is properly configured.

- disc
Communication between the two clusters has not been established. This could occur because there are no jobs waiting to be dispatched, or because the remote management host cannot be located.
- reject
The remote queue rejects jobs from the send-jobs queue. The local queue and remote queue are connected and the clusters communicate, but the queue-level configuration is not correct. For example, the send-jobs queue in the submission cluster points to a receive-jobs queue that does not exist in the remote cluster.

If the job is rejected, it returns to the submission cluster.

For example, consider the following application profile configurations:

- On the submission cluster (Cluster1) in the lsb.applications file:

```
Begin Application
NAME           = fluent
DESCRIPTION    = FLUENT Version 6.2
CPULIMIT      = 180/bp860-10      # 3 hours of host hostA
FILELIMIT     = 20000
DATALIMIT     = 20000             # jobs data segment limit
CORELIMIT     = 20000
PROCLIMIT     = 5                 # job processor limit
PRE_EXEC      = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
POST_EXEC     = /usr/local/lsf/misc/testq_post |grep -v "Hi"
REQUEUE_EXIT_VALUES = 55 34 78
End Application
```

```
Begin Application
NAME           = catia
DESCRIPTION    = CATIA V5
CPULIMIT      = 24:0/bp860-10    # 24 hours of host hostA
FILELIMIT     = 20000
DATALIMIT     = 20000             # jobs data segment limit
CORELIMIT     = 20000
PROCLIMIT     = 5                 # job processor limit
PRE_EXEC      = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
POST_EXEC     = /usr/local/lsf/misc/testq_post |grep -v "Hi"
REQUEUE_EXIT_VALUES = 55 34 78
End Application
```

```
Begin Application
NAME           = djob
DESCRIPTION    = distributed jobs
FILELIMIT     = 20000
DATALIMIT     = 2000000          # jobs data segment limit
RTASK_GONE_ACTION="KILLJOB TASKEXIT IGNORE TASKCRASH"
DJOB_ENV_SCRIPT = /lsf/djobs/proj_1/djob_env
DJOB_RU_INTERVAL = 300
DJOB_HB_INTERVAL = 30
DJOB_COMMFAIL_ACTION="KILL_TASKS"
End Application
```

- On the execution cluster (Cluster2) in the lsb.applications file:

```
Begin Application
NAME           = dyna
DESCRIPTION    = ANSYS LS-DYNA
CPULIMIT      = 8:0/amd64dcore   # 8 hours of host model SunIPC
FILELIMIT     = 20000
DATALIMIT     = 20000             # jobs data segment limit
CORELIMIT     = 20000
PROCLIMIT     = 5                 # job processor limit
PRE_EXEC      = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
POST_EXEC     = /usr/local/lsf/misc/testq_post |grep -v "Hi"
REQUEUE_EXIT_VALUES = 55 255 78
End Application
```

```

Begin Application
NAME           = default
DESCRIPTION    = global defaults
CORELIMIT     = 0                # No core files
STACKLIMIT    = 200000          # Give large default
RERUNNABLE    = Y                #
RES_REQ       = order[mem:ut]   # change the default ordering method
End Application

```

Verify that MultiCluster is enabled:

```

lsclusters

```

CLUSTER_NAME	STATUS	MASTER_HOST	ADMIN	HOSTS	SERVERS
cluster1	ok	management_c1	admin	1	1
cluster2	ok	management_c2	admin	2	2

View available applications on remote clusters from the submission cluster (Cluster1):

```

bclusters -app

```

REMOTE_CLUSTER	APP_NAME	DESCRIPTION
cluster2	dyna	ANSYS LS-DYNA
cluster2	default	global defaults

View available applications on remote clusters from the execution cluster (Cluster2):

```

bclusters -app

```

REMOTE_CLUSTER	APP_NAME	DESCRIPTION
cluster1	catia	CATIA V5
cluster1	fluent	FLUENT Version 6.2
cluster1	djob	distributed jobs

Job scheduling under the job forwarding model

About this task

With this model, scheduling of MultiCluster jobs is a process with two scheduling phases:

Procedure

- the submission cluster selects a suitable remote receive-jobs queue, and forwards the job to it
 - the execution cluster selects a suitable host and dispatches the job to it.
- If a suitable host is not found immediately, the job remains pending in the execution cluster, and is evaluated again the next scheduling cycle.

This method automatically favors local hosts; a MultiCluster send-jobs queue always attempts to find a suitable local host before considering a receive-jobs queue in another cluster.

Phase I, local scheduling phase (all jobs)

Procedure

1. The send-jobs queue receives the job submission request from a user.
2. The send-jobs queue parameters affect whether or not the job is accepted. For example, a job that requires 100 MB memory will be rejected if queue-level parameters specify a memory limit of only 50 MB.
3. If the job is accepted, it becomes pending in the send-jobs queue with a job ID assigned by the submission cluster.
4. During the next scheduling cycle, the send-jobs queue attempts to place the job on a host in the submission cluster. If a suitable host is found, the job is dispatched locally.
5. If the job cannot be placed locally (local hosts may not satisfy its resource requirements, or all the local hosts could be busy), the send-jobs queue attempts to forward the job to another cluster.

Phase II, job forwarding phase (MultiCluster submission queues only)

Procedure

1. The send-jobs queue has a list of remote receive-jobs queues that it can forward jobs to. If a job cannot be placed locally, the send-jobs queue evaluates each receive-jobs queue. All queues that will accept more MultiCluster jobs are candidates. To find out how many additional MultiCluster jobs a queue can accept, subtract the number of MultiCluster jobs already pending in the queue from the queue's pending MultiCluster job threshold (IMPT_JOBKLG).
2. To enable LSF MultiCluster job forwarding, uncomment `schmod_mc` in `lsb.modules`.
3. If information available to the submission cluster indicates that the first queue is suitable, LSF forwards the job to that queue.
 - a. By default, only queue capacity is considered and the first queue evaluated is the one that has room to accept the most new MultiCluster jobs.
 - b. When `MC_PLUGIN_REMOTE_RESOURCE=Y` is set, boolean resource requirements and available remote resources are considered.
Tip: When `MC_PLUGIN_REMOTE_RESOURCE` is defined, only the following resource requirements (boolean only) are supported: `-R "type==type_name"`, `-R "same[type]"`, and `-R "defined(resource_name)"`
 - c. When `MC_PLUGIN_SCHEDULE_ENHANCE` is defined, remote resources are considered as for `MC_PLUGIN_REMOTE_RESOURCE=Y`, and the scheduler is enhanced to consider remote queue preemptable jobs, queue priority, and queue workload, based on the settings selected.
 - d. If `TASKLIMIT` is defined in the remote cluster (in `lsb.applications` for the application profile or `lsb.queues` for the receive queue), the `TASKLIMIT` settings are considered.
 - If the remote application's `TASKLIMIT` in the remote cluster cannot satisfy the job's processor requirements for an application profile, the job is not forwarded to that cluster.
 - If the remote application's `TASKLIMIT` cannot satisfy the receive queue's `TASKLIMIT` in the remote cluster, the job is not forwarded to that remote queue.
 - If the receive queue's `TASKLIMIT` in the remote cluster cannot satisfy the job's processor requirements, the job is not forwarded to that remote queue.
4. If the first queue is not suitable, LSF considers the next queue.
5. If LSF cannot forward the job to any of the receive-jobs queues, the job remains pending in the send-jobs cluster and is evaluated again during the next scheduling cycle.

Phase III, remote scheduling phase (MultiCluster jobs only)

Procedure

1. The receive-jobs queue receives the MultiCluster job submission.
2. The receive-jobs queue parameters affect whether or not the job is accepted. For example, a job that requires 100 MB memory will be rejected if queue-level parameters specify a memory limit of only 50 MB.
3. If the job is rejected, it returns to the submission cluster.
4. If the job is accepted, it becomes pending in the receive-jobs queue with a new job ID assigned by the execution cluster.
5. During the next scheduling cycle, the receive-jobs queue attempts to place the job on a host in the execution cluster. If a suitable host is found, the job is dispatched. If a suitable host is not found, the job remains pending in the receive-jobs cluster, and is evaluated again the next scheduling cycle.
6. If the job is dispatched to the execution host but cannot start after the time interval `MAX_RSCHED_TIME` (`lsb.params`), it returns to the submission cluster to be rescheduled. However, if the job repeatedly returns to the submission cluster because it could not be started in a remote cluster, after `LSB_MC_INITFAIL_RETRY` tries to start the job (`lsf.conf`), LSF suspends the job (PSUSP) in the submission cluster, unless a forwarded job could not start in remote cluster due to insufficient resource.

Queue scheduling parameters under job forwarding model

Forcing consistent scheduling behavior

If the queue policies of the send-jobs queue are the same as the queue policies of the receive-jobs queue, the user should see identical behavior, whether the job is scheduled locally or remotely.

Queue policies differ

The job-level (user-specified) requirements and queue-level parameters (set by the administrator) are used to schedule and run the job.

If a job runs in the submission cluster, the send-jobs queue parameters apply. If a job becomes a MultiCluster job and runs in another cluster, the receive-jobs queue parameters apply.

Since the receive-jobs queue policies replace the send-jobs queue policies, LSF users might notice that identical jobs are subject to different scheduling policies, depending on whether or not the job becomes a MultiCluster job.

Send-jobs queue parameters that affect MultiCluster jobs

- If the job requirements conflict with the send-jobs queue parameters, the job is rejected by the send-jobs queue.
- In general, queue-level parameters at the submission side don't affect the scheduling of MultiCluster jobs once the jobs have been forwarded to the execution queue.

Receive-jobs queue parameters that affect MultiCluster jobs

In general, queue-level policies set on the execution side are the only parameters that affect MultiCluster jobs:

- If the job requirements conflict with the receive-jobs queue parameters, the job is rejected by the receive-jobs queue and returns to the submission cluster.
- Runtime queue level parameters (terminate when, job starter, load threshold, exclusive, etc): the receive-jobs queue settings are enforced, the send-jobs queue settings are ignored.
- Resource requirements: the receive-jobs queue settings are enforced, the send-jobs queue settings are ignored.
- Resource limits: the execution cluster settings are enforced, the submission cluster settings are ignored.
- Job slot limits (hjob limit, ujob limit, qjob limit): the execution cluster settings are enforced, the submission cluster settings are ignored.

Advance reservations across clusters

Users can create and use advance reservation for the MultiCluster job forwarding model. To enable this feature, you must upgrade all clusters to LSF 10.1.0 or later.

Advance reservation

The user from the submission cluster negotiates an advance reservation with the administrator of the execution cluster. The administrator creates the reservation in the execution cluster.

The reservation information is visible from the submission cluster. To submit a job and use the reserved resources, users specify the reservation at the time of job submission.

A job that specifies a reservation can only start on the reserved resources during the time of the reservation, even if other resources are available. Therefore, this type of job does not follow the normal scheduling process. Instead, the job is immediately forwarded to the execution cluster and is held in PEND until it can start. These jobs are not affected by the remote timeout limit (MAX_RSCHED_TIME in lsb.queues) since the system cannot automatically reschedule the job to any other cluster.

Missed reservations

If the execution cluster cannot accept the job because the reservation is expired or deleted, the job will be in the submission cluster in the PSUSP state.

The pending reason is:

Specified reservation has expired or has been deleted.

The job should be modified or killed by the owner.

If the execution cluster accepts the job and then the reservation expires or is deleted while job is pending, the job will be in the execution cluster detached from the reservation and scheduled as a normal job.

Broken connections

If cluster connectivity is interrupted, all remote reservations are forgotten.

During this time, submission clusters will not be able to see remote reservations; jobs submitted with remote reservation and not yet forwarded will PEND; and new jobs will not be able to use the reservation. Reservation information will not be available until cluster connectivity is re-established and the clusters have a chance to synchronize on reservation. At that time (given that reservation is still available), jobs will be forwarded, new jobs can be submitted with specified reservation, and users will be able to see the remote reservation.

Modify a reservation

After an advance reservation is made, you can use **brsvmod** to modify the reservation.

Advance reservations only can be modified with **brsvmod** in the local cluster. A modified remote reservation is visible from the submission cluster. The jobs attached to the remote reservation are treated as the local jobs when the advance reservation is modified in the remote cluster.

Delete a reservation

After an advance reservation is made, you can use **brsvdel** to delete the reservation from the execution cluster.

brsvdel reservation_ID

If you try to delete the reservation from the submission cluster, you will see an error.

Submit jobs to a reservation in a remote cluster

Submit the job and specify the remote advance reservation as shown:

bsub -U reservation_name@cluster_name

In this example, we assume the default queue is configured to forward jobs to the remote cluster.

Extend a reservation

bmod -t allows the job to keep running after the reservation expires.

The command **bmod** does not apply to pending jobs or jobs that are already forwarded to the remote cluster. However it can be used on the execution cluster. For that, it behaves as if it is a local job.

Special considerations under job forwarding model

Chunk jobs

Note: Job chunking is now deprecated and might be removed in a future version of LSF.

Job chunking is done after a suitable host is found for the job. Multiclust jobs can be chunked, but they are forwarded to the remote execution cluster one at a time, and chunked in the execution cluster. Therefore, the **CHUNK_JOB_SIZE** parameter in the submission queue is ignored by multiclust jobs that are forwarded to a remote cluster for execution.

If multicluster jobs are chunked, and one job in the chunk starts to run, both clusters display the **WAIT** status for the remaining jobs. However, the execution cluster sees these jobs in the **PEND** state, while the submission cluster sees these jobs in the **RUN** state. This affects scheduling calculations for fair share and limits on both clusters.

Fair share

If fair share scheduling is enabled, resource usage information is a factor used in the calculation of dynamic user priority. Multicluster jobs count towards a user's fair share priority in the execution cluster, and do not affect fair share calculations in the submission cluster.

There is no requirement that both clusters use fair share or have the same fair share policies. However, if you submit a job and specify a local user group for fair share purposes (**bsub -G**), your job cannot run remotely unless you also belong to a user group of the same name in the execution cluster.

Parallel jobs

A parallel job can be forwarded to another cluster, but the job cannot start unless the execution cluster has enough hosts and resources to run the entire job. A parallel job cannot span clusters.

Resizable jobs

Resizable jobs across multicluster clusters is not supported. This implies the following behavior:

- For the forwarding model, once job is forwarded to remote cluster, job is not automatically resizable.
- You cannot run **bresize** commands to shrink allocations from submission clusters in either lease model or job forwarding model

Only **bresize release** is supported in the job forwarding model from the execution cluster:

- The submission cluster logs all events related to **bresize** release in the submission cluster `lsb.events` file.
- The submission cluster logs `JOB_RESIZE` events into the `lsb.acct` file after the allocation is changed.
- Users should be able to view allocation changes from submission cluster through **bjobs**, **bhist**, **bacct**, **busers**, **bqueues**, and so on.

Job re-queuing

If job re-queue is enabled, LSF re-queues jobs that finish with exit codes that indicate job failure.

User-specified job re-queues

bre-queue in the submission cluster causes the job to be re-queued in the send-jobs queue.

bre-queue in the execution cluster causes the job to be re-queued in the receive-jobs queue.

Automatic job re-queues

1. If job re-queue (`REQUEUE_EXIT_VALUES` in `lsb.queues`) is enabled in the receive-jobs queue, and the job's exit code matches, the execution cluster re-queues the job (it does not return to the submission cluster). Exclusive job re-queue works properly.
2. If the execution cluster does not re-queue the job, the job returns to the send-jobs cluster, and gets a second chance to be re-queued. If job re-queue is enabled in the send-jobs queue, and the job's exit code matches, the submission cluster re-queues the job.
3. Exclusive job re-queue values configured in the send-jobs queue always cause the job to be re-queued, but for multicluster jobs the exclusive feature does not work; these jobs could be dispatched to the same remote execution host as before.

Automatic retry limits

The pre-execution command retry limit (`MAX_PREEEXEC_RETRY`, `LOCAL_MAX_PREEEXEC_RETRY`, and `REMOTE_MAX_PREEEXEC_RETRY`), job re-queue limit (`MAX_JOB_REQUEUE`), and job preemption retry limit (`MAX_JOB_PREEMPT`) configured in `lsb.params`, `lsb.queues`, and `lsb.applications` on the execution cluster are applied. If the forwarded job re-queue limit exceeds the limit on the execution cluster, the job exits and returns to the submission cluster and remains pending for rescheduling.

Remote pending job re-queue

Use **bre-queue -p** to re-queue specified remote pending jobs.

Use **bre-queue -a** to re-queue remote pending jobs and all local jobs (including **DONE** and **EXIT** jobs in local cluster).

Only job owners, LSF cluster administrators, or LSF group administrators can re-queue a job.

The only difference between **-p** and **-a** is the job dispatch order. Running **bre-queue -p** in the submission cluster re-queues a remote job to the top of the queue, so that the re-queued job is dispatched first no matter which position it is in the pending job list. **bre-queue -a** puts the remote job to the end of queue just as in the local cluster.

Job rerun

If job rerun is enabled, LSF automatically restarts running jobs that are interrupted due to failure of the execution host.

If queue-level job rerun (RERUNNABLE in lsb.queues) is enabled in both send-jobs and receive-jobs queues, only the receive-jobs queue reruns the job.

1. If job rerun is enabled in the receive-jobs queue, the execution cluster reruns the job. While the job is pending in the execution cluster, the job status is returned to the submission cluster.
2. If the receive-jobs queue does not enable job rerun, the job returns to the submission cluster and gets a second chance to be rerun. If job rerun is enabled at the user level, or is enabled in the send-jobs queue, the submission cluster reruns the job.

Sorting forwarded jobs by their submission time

Enabling the parameter MC_SORT_BY_SUBMIT_TIME in lsb.params allows forwarded jobs on the execution cluster to be sorted and run based on their original submission time (instead of their forwarded time), in a multicluster environment.

- [Job migration](#)
- [Checkpoint a multicluster job](#)
- [Absolute priority scheduling](#)
- [Strict resource requirement select string syntax](#)
- [Compute unit requirement strings](#)

Job migration

Procedure

As long as a MultiCluster job is rerunnable (**bsub -r** or RERUNNABLE=yes in the send-jobs queue) and is not checkpointable, you can migrate it to another host, but you cannot specify which host. Migrated jobs return to the submission cluster to be dispatched with a new job ID.

For more information on job migration, see Administering IBM Platform LSF.

User-specified job migration

Procedure

To migrate a job manually, run **bmig** in either the submission or execution cluster, using the appropriate job ID. You cannot use **bmig -m** to specify a host.

Tip:

Operating in the execution cluster is more efficient than sending the **bmig** command through the submission cluster.

Automatic job migration

Procedure

1. To enable automatic job migration, set the migration threshold (MIG in lsb.queues) in the receive-jobs queue.
2. You can also set a migration threshold at the host level on the execution host (MIG in lsb.hosts).
The lowest migration threshold applies to the job.

Tip:

Automatic job migration configured in the send-jobs queue does not affect MultiCluster jobs.

Migration of checkpointable jobs

Procedure

Checkpointable MultiCluster jobs cannot be migrated to another host.

The migration action stops and checkpoints the job, then schedules the job on the same host again.

Checkpoint a multicluster job

Before you begin

Checkpointing of a multicluster job is only supported when the send-jobs queue is configured to forward jobs to a single remote receive-jobs queue, without ever using local hosts:

Procedure

The checkpoint-able multicluster jobs resume on the same host.

Configuration

About this task

Checkpointing multicluster jobs

Procedure

To enable checkpointing of multicluster jobs, define a checkpoint directory:

- a. in both the send-jobs and receive-jobs queues (CHKPNT in lsb.queues)
- b. or in an application profile (CHKPNT_DIR, CHKPNT_PERIOD, CHKPNT_INITPERIOD, CHKPNT_METHOD in lsb.applications) of both submission cluster and execution cluster.

LSF uses the directory specified in the execution cluster and ignores the directory specified in the submission cluster.

LSF writes the checkpoint file in a subdirectory named with the submission cluster name and submission cluster job ID. This allows LSF to checkpoint multiple jobs to the same checkpoint directory. For example, the submission cluster is **ClusterA**, the submission job ID is 789, and the send-jobs queue enables checkpointing. The job is forwarded to **ClusterB**, the execution job ID is 123, and the receive-jobs queue specifies a checkpoint directory called XYZ_dir. LSF will save the checkpoint file in the XYZ_dir/clusterA/789/ directory.

Tip: You cannot use **bsub -k** to make a multicluster job checkpoint-able.

Checkpoint a job

Procedure

To checkpoint and stop a multicluster job, run **bmig** in the execution cluster and specify the local job ID.

Tip: You cannot run **bmig** from the submission cluster. You cannot use **bmig -m** to specify a host.

Force a check-pointed job

Procedure

Use **brun** to force any pending job to be dispatched immediately to a specific host, regardless of user limits and fair share priorities. This is the only way to resume a job that has been check-pointed on a different host. By default, these jobs attempt to restart from the last checkpoint.

Tip: Use **brun -b** if you want to make checkpoint-able jobs start over from the beginning (for example, this might be necessary if the new host does not have access to the old checkpoint directory).

Example

About this task

In this example, users in a remote cluster submit work to a data center using a send-jobs queue that is configured to forward jobs to only one receive-jobs queue. You are the administrator of the data center and you need to shut down a host for maintenance. The host is busy running checkpoint-able multicluster jobs.

Procedure

1. Before you perform maintenance on a host in the execution cluster, take these steps:
 - a. Run **badmin hclose** to close the host and prevent additional jobs from starting on the host.
 - b. Run **bmig** and specify the execution cluster job IDs of the checkpoint-able multicluster jobs running on the host. For example, if jobs from a remote cluster use job IDs 123 and 456 in the local cluster, type the following command to checkpoint and stop the jobs:

bmig 123 456

You cannot use **bmig -m** to specify a host.
 - c. Allow the checkpoint process to complete. The jobs are requeued to the submission cluster. From there, they will be forwarded to the same receive-jobs queue again, and scheduled on the same host. However, if the host is closed, they will not start.
 - d. Shut down LSF daemons on the host.
2. After you perform maintenance on a host, take these steps:
 - a. Start LSF daemons on the host.
 - b. Use **badmin hopen** to open the host. The multicluster jobs resume automatically.

Absolute priority scheduling

When absolute priority scheduling (APS) is enabled in the submission queue:

- The APS value at the submission cluster:
 - The APS value will affect the job forwarding order for new incoming jobs, but not for jobs that have already been forwarded (that is, the job is still pending at the execution cluster)
 - The APS value does not affect the job order at the remote cluster. Job order is determined by the local policies at the remote cluster.
 - **bmod -aps** does not apply to the send-jobs queue
 - **bjobs -aps** shows the job order and APS value at the local cluster
- The APS value at the execution cluster:
 - The APS value at receiving queue will affect remote job execution at the execution cluster
 - The APS value at the execution cluster will not be sent back to the submission cluster

Strict resource requirement select string syntax

Resource requirements are checked before jobs are forwarded to the remote cluster. If the selection string is valid, the job is forwarded.

When strict resource requirement checking configuration does not match between the submission and remote clusters, jobs may be rejected by the remote cluster.

Compute unit requirement strings

When a job is submitted with compute unit resource requirements, any requirements apply only to the execution cluster. Only the syntax of the resource requirement string is checked on the submission side, and if the `cu[]` string is valid, the job is forwarded.

When compute unit requirements cannot be satisfied in the remote cluster (such as a non-existent compute unit type) jobs may be rejected by the remote cluster. Hosts running LSF 7 Update 4 or earlier cannot satisfy compute unit resource requirements.

Queues for the LSF multicluster capability

By default, clusters do not share resources, even if the LSF multicluster capability is installed. To enable job forwarding, enable queues for the LSF multicluster capability in both the submission and execution clusters.

How it works

Send-jobs queue

A send-jobs queue can forward jobs to a specified remote queue. By default, LSF attempts to run jobs in the local cluster first. LSF only attempts to place a job remotely if it cannot place the job locally.

Receive-jobs queue

A receive-jobs queue accepts jobs from queues in a specified remote cluster. Although send-jobs queues only forward jobs to specific queues in the remote cluster, receive-jobs queues can accept work from any and all queues in the remote cluster.

Multiple queue pairs

- You can configure multiple send-jobs and receive-jobs queues in one cluster.
- A queue can forward jobs to as many queues in as many clusters as you want, and can also receive jobs from as many other clusters as you want.
- A receive-jobs queue can also borrow resources using the resource leasing method, but a send-jobs queue using the job forwarding method cannot also share resources using the resource leasing method.
- [Enable queues for the LSF multicluster capability](#)
To enable queues for the LSF multicluster capability, configure a send-jobs queue in the submission cluster and a receive-jobs queue in the execution cluster.

Enable queues for the LSF multicluster capability

To enable queues for the LSF multicluster capability, configure a send-jobs queue in the submission cluster and a receive-jobs queue in the execution cluster.

About this task

To set up a pair of queues for the LSF multicluster capability, do the following:

Procedure

1. In the submission cluster, configure a send-jobs queue that forwards work to the execution queue.
2. In the execution cluster, configure a receive-jobs queue that accepts work from the cluster that contains the send-jobs queue.

Send-jobs queues

Procedure

1. To configure a send-jobs queue, define the SNDJOBS_TO parameter in the lsb.queues queue definition. Specify a space-separated list of queue names in the format *queue_name@cluster_name*.
If the send-jobs queue does not have the SNDJOBS_TO parameter configured, it cannot forward jobs for the LSF multicluster capability. The job remains pending in the submission cluster and is evaluated again during the next scheduling cycle.
2. Make sure the HOSTS parameter in the lsb.queues file specifies only local hosts (or the special keyword none).
If the HOSTS parameter specifies any remote hosts, the SNDJOBS_TO parameter is ignored and the queue behaves as a receive-jobs queue under the resource leasing method.
3. Optional. To restrict users or user groups from forwarding jobs to remote clusters in this queue, define the FWD_USERS parameter in the lsb.queues queue definition.
Specify a space-separated list of user names or user groups in the following format:

```
FWD_USERS=all [~user_name ...] [~user_group ...] | [user_name ...] [user_group [~user_group ...] ...]
```

Receive-jobs queues

Procedure

To configure a receive-jobs queue, define the RCVJOBS_FROM parameter in the lsb.queues queue definition. Specify a space-separated list of cluster names.

Use the keyword `allclusters` to specify any remote cluster.

Examples

```
Begin Queue
QUEUE_NAME=send
PRIORITY=30
NICE=20
SNDJOBS_TO = rcv@allclusters
HOSTS = none
FWD_USERS=all ~ugroup1
End Queue
```

The following queue is both a send-jobs and receive-jobs queue, and links with multiple remote clusters. If `queue1` cannot place a job in the local cluster, it can forward the job to `queue2` in `cluster2`, or to `queue3` in `cluster3`. If any queues in clusters 2 or 3 are configured to send MultiCluster jobs to `queue1`, `queue1` accepts them.

```
Begin Queue
QUEUE_NAME=queue1
SNDJOBS_TO=queue2@cluster2 queue3@cluster3
RCVJOBS_FROM=cluster2 cluster3
PRIORITY=30
NICE=20
End Queue
```

Remote-only queues

By default, LSF tries to place jobs in the local cluster. If your local cluster is occupied, it may take a long time before your jobs can run locally. You might want to force some jobs to run on a remote cluster instead of the local cluster. Submit these jobs to a remote-only queue. A remote-only queue forwards all jobs to a remote cluster without attempting to schedule the job locally.

- [Configure a remote-only queue](#)

Configure a remote-only queue

About this task

To make a queue that only runs jobs in remote clusters, take the following steps:

Procedure

1. Edit the lsb.queues queue definition for the send-jobs queue.
 - a. Define SNDJOBS_TO. This specifies that the queue can forward jobs to specified remote execution queues.
 - b. Set HOSTS to none. This specifies that the queue uses no local hosts.
2. Edit the lsb.queues queue definition for each receive-jobs queue.
 - a. Define RCVJOBS_FROM. This specifies that the receive-jobs queue accepts jobs from the specified submission cluster.

Example

In cluster1:

```
Begin Queue
QUEUE_NAME = queue1
HOSTS = none
SNDJOBS_TO = queue2@cluster2
MAX_RSCHED_TIME = infinit
DESCRIPTION = A remote-only queue that sends jobs to cluster2.
End Queue
```

In cluster2:

```
Begin Queue
QUEUE_NAME = queue2
RCVJOBS_FROM = cluster1
DESCRIPTION = A queue that receives jobs from cluster1.
End Queue
```

Queue1 in cluster1 forwards all jobs to queue2 in cluster2.

Disable timeout in remote-only queues

About this task

A remote-only send-jobs queue that sends to only one receive-jobs queue.

Procedure

Set MAX_RSCHED_TIME=infinit to maintain FCFS job order of MultiCluster jobs in the execution queue.

Otherwise, jobs that time out are rescheduled to the same execution queue, but they lose priority and position because they are treated as a new job submission.

In general, the timeout is helpful because it allows LSF to automatically shift a pending MultiCluster job to a better queue.

Submit a job to run in a remote cluster

About this task

Jobs can be submitted to run only in a remote cluster.

Procedure

Use **bsub -q** and specify a remote-only MultiCluster queue.

This is not compatible with **bsub -m**. When your job is forwarded to a remote queue, you cannot specify the execution host by name.

Example:

`queue1` is a remote-only MultiCluster queue.

```
% bsub -q queue1 myjob
Job <101> is submitted to queue <queue1>.
```

This job will be dispatched to a remote cluster.

Force a pending job to run

Use **brun -m** to force a pending or finished job to run or be forwarded to a specified cluster. The exact behavior of **brun** on a pending job depends on where the job is pending, and which hosts or clusters are specified in the **brun** command.

Important:

Only administrators can use the **brun** command. You can only run **brun** from the submission cluster.

You must specify one or more host names or a cluster name when you force a job to run.

If multiple hosts are specified, the first available host is selected and the remainder ignored. Specified hosts cannot belong to more than one cluster.

You can only specify one cluster name. The job is forced to be forwarded to the specified cluster.

You cannot specify host names and cluster names together in the same **brun** command.

A job pending in an execution cluster forced to run in a different cluster is returned to the submission cluster, and then forwarded once again.

If a job is submitted with a cluster name and the job is forwarded to a remote cluster, you cannot use **brun -m** again to switch the job to another execution cluster. For example:

```
bsub -m cluster1 -q test1 sleep 1000
```

The job is pending on cluster1. Running **brun** again to forward the job to cluster2 is rejected:

```
brun -m cluster2 1803
Failed to run the job: Hosts requested do not belong to the cluster
```

For example:

```
brun -m "host12 host27"
```

In this example, if host12 is available the job is sent to the cluster containing host12 and tries to run. If unsuccessful, the job pends in the cluster containing host12. If host12 is not available, the job is sent to the cluster containing host27 where it runs or pends.

Force a job to run on a specific host

Local host specified

Job runs locally. For example:

```
brun -m hostA 246
Job <246> is being forced to run or forwarded.
```

```
bjobs 246
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
246    user1  RUN   normal  hostD      hostA      *eep 10000 Jan  3 12:15
```

```
bhist -l 246
Job <246>, User <user1>, Project <default>, Command <sleep 10000>
Mon Jan  3 12:15:22: Submitted from host <hostD>, to Queue <normal>,
CWD <$HOME/envs>, Requested Resources <type == any>;
Mon Jan  3 12:16:13: Job is forced to run or forwarded by user or administrator <user1>;
Mon Jan  3 12:16:13: Dispatched to <hostA>;
Mon Jan  3 12:16:41: Starting (Pid 10467);
Mon Jan  3 12:16:59: Running with execution home </home/user1>,
Execution CWD </home/user1/envs>, Execution Pid <10467>;
```

Host in execution cluster specified

Job is forwarded to execution cluster containing specified host, and runs.

For example:

```
brun -m hostB 244
Job <244> is being forced to run or forwarded.
```

```
bjobs 244
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
244    user1  RUN   normal  hostD      hostB      *eep 10000 Jan  3 12:15
```

```
bhist -l 244
Job <244>, User <user1>, Project <default>, Command <sleep 10000>
Mon Jan  3 12:15:22: Submitted from host <hostD>, to Queue <normal>,
CWD <$HOME/envs>, Requested Resources <type == any>;
Mon Jan  3 12:19:18: Job is forced to run or forwarded by user or administrator <user1>;
Mon Jan  3 12:19:18: Forwarded job to cluster cluster2;
Mon Jan  3 12:19:18: Remote job control initiated;
Mon Jan  3 12:19:18: Dispatched to <hostB>;
Mon Jan  3 12:19:18: Remote job control completed;
Mon Jan  3 12:19:19: Starting (Pid 28804);
Mon Jan  3 12:19:19: Running with execution home </home/user1>,
Execution CWD </home/user1/envs>, Execution Pid <28804>;
```

Host in same execution cluster specified

Job runs on the specified host in the same execution cluster. For example:

```
brun -m hostB 237
Job <237> is being forced to run or forwarded.
```

```
bjobs 237
JOBID  USER  STAT  QUEUE  FROM_HOST  EXEC_HOST  JOB_NAME  SUBMIT_TIME
237    user1  RUN   normal  hostD      hostB      *eep 10000 Jan  3 12:14
```

```
bhist -l 237
Job <237>, User <user1>, Project <default>, Command <sleep 10000>
Mon Jan  3 12:14:48: Submitted from host <hostD>, to Queue <normal>,
CWD <$HOME/envs>, Requested Resources <type == any>;
Mon Jan  3 12:14:53: Forwarded job to cluster cluster2;
Mon Jan  3 12:22:08: Job is forced to run or forwarded by user or administrator <user1>;
Mon Jan  3 12:22:08: Remote job control initiated;
Mon Jan  3 12:22:08: Dispatched to <hostB>;
Mon Jan  3 12:22:09: Remote job control completed;
Mon Jan  3 12:22:09: Starting (Pid 0);
Mon Jan  3 12:22:09: Starting (Pid 29073);
Mon Jan  3 12:22:09: Running with execution home </home/user1>,
Execution CWD </home/user1/envs>, Execution Pid <29073>;
```

Host in submission cluster specified

Job runs on the specified host in the submission cluster. For example:

```
brun -m hostA 238
Job <238> is being forced to run or forwarded.
```

```

bjobs 237
JOBID    USER    STAT  QUEUE    FROM_HOST    EXEC_HOST    JOB_NAME    SUBMIT_TIME
238      user1    RUN   normal    hostB        hostA        *eep 10000  Oct  5 11:00

bhist -l 237

Job <237>, User <user1>, Project <default>, Command <sleep 10000>
Wed Oct  5 11:00:16: Submitted from host <hostB>, to Queue <normal>,
  CWD </usr/local/xl/conf>,
    Requested Resources <type == any>;
Wed Oct  5 11:00:18: Forwarded job to cluster ecl;
Wed Oct  5 11:00:46: Job is forced to run or forwarded by user or administrator <user1>;
Wed Oct  5 11:00:46: Pending: Job has returned from remote cluster;
Wed Oct  5 11:00:46: Dispatched to <hostA>;
Wed Oct  5 11:00:46: Starting (Pid 15686);
Wed Oct  5 11:00:47: Running with execution home </home/user1>,
  Execution CWD </usr/local/xl/conf>,
    Execution Pid <15686>;

Summary of time in seconds spent in various states by Wed Oct  5 11:01:06
  PEND    PSUSP    RUN    USUSP    SSUSP    UNKWN    TOTAL
   30         0    20         0         0         0    50

```

Force a job to run in a specific cluster

Host in different execution cluster specified

Job returns to submission cluster, is forwarded to execution cluster containing specified host, and runs.

```

brun -m ec2-hostA 3111
Job <3111> is being forced to run or forwarded.

```

```

bjobs 3111
JOBID    USER    STAT  QUEUE    FROM_HOST    EXEC_HOST    JOB_NAME    SUBMIT_TIME
3111      user1    RUN   queue1    sub-management  ec2-hostA    sleep 1000  Feb 23 11:21

bhist -l 3111

Job <3111>, User <user1>, Project <default>, Command <sleep 1000>
Wed Feb 23 11:21:00: Submitted from host <sub-management>, to Queue <queue1>,
  CWD </usr/local/xl/conf>;
Wed Feb 23 11:21:03: Forwarded job to cluster cluster1;
Wed Feb 23 11:21:58: Job is forced to run or forwarded by user or administrator <user1>;
Wed Feb 23 11:21:58: Pending: Job has returned from remote cluster;
Wed Feb 23 11:21:58: Forwarded job to cluster cluster2;
Wed Feb 23 11:21:58: Remote job run control initiated;
Wed Feb 23 11:21:59: Dispatched to <ec2-hostA>;
Wed Feb 23 11:21:59: Remote job run control completed;
Wed Feb 23 11:21:59: Starting (Pid 3257);
Wed Feb 23 11:21:59: Running with execution home </home/user1>,
  Execution CWD </usr/local/xl/conf >, Execution Pid <3257>;

Summary of time in seconds spent in various states by Wed Feb 23 11:24:59
  PEND    PSUSP    RUN    USUSP    SSUSP    UNKWN    TOTAL
   59         0   180         0         0         0   239

```

Job already forwarded to execution

Job has already been forwarded to an execution cluster, and you specify a different execution cluster. The job returns to submission cluster, and is forced to be forwarded to the specified execution cluster. The job is not forced to run in the new execution cluster. After the job is forwarded, the execution cluster schedules the job according to local policies. For example:

```

brun -m cluster2 244
Job <244> is being forced to run or forwarded.

```

```

bjobs 244
JOBID    USER    STAT  QUEUE    FROM_HOST    EXEC_HOST    JOB_NAME    SUBMIT_TIME
244      user1    RUN   normal    hostD        hostB        *eep 10000  Jan  3 12:15

bhist -l 244

```

```

Job <244>, User <user1>, Project <default>, Command <sleep 10000>
Mon Jan 3 12:15:22: Submitted from host <hostD>, to Queue <normal>,
  CWD <$HOME/envs>, Requested Resources <type == any>;
Mon Jan 3 12:15:25: Forwarded job to cluster cluster1;
Mon Jan 3 12:19:18: Job is forced to run or forwarded by user or administrator <user1>;
Mon Jan 3 12:19:18: Pending: Job has returned from remote cluster;
Mon Jan 3 12:19:18: Forwarded job to cluster cluster2;
Mon Jan 3 12:19:18: Dispatched to <hostB>;
Mon Jan 3 12:19:19: Starting (Pid 28804);
Mon Jan 3 12:19:19: Running with execution home </home/user1>,
  Execution CWD </home/user1/envs>, Execution Pid <28804>;

```

Job pending in execution cluster

Job is forwarded to the specified execution cluster, but the job is not forced to run. After the job is forwarded, the execution cluster schedules the job according to local policies.

For example:

```

brun -m cluster2 244
Job <244> is being forced to run or forwarded.

```

```

bhist -l 244

Job <244>, User <user1>, Project <default>, Command <sleep 10000>
Mon Jan 3 12:15:22: Submitted from host <hostD>, to Queue <normal>,
  CWD <$HOME/envs>, Requested Resources <type == any>;
Mon Jan 3 12:19:18: Job is forced to run or forwarded by user or administrator <user1>;
Mon Jan 3 12:19:18: Forwarded job to cluster cluster2;
Mon Jan 3 12:19:18: Remote job control initiated;
Mon Jan 3 12:19:18: Dispatched to <hostB>;
Mon Jan 3 12:19:18: Remote job control completed;
Mon Jan 3 12:19:19: Starting (Pid 28804);
Mon Jan 3 12:19:19: Running with execution home </home/user1>,
  Execution CWD </home/user1/envs>, Execution Pid <28804>;

```

Request a specific cluster

You can specify cluster names when submitting jobs. If no cluster name is specified, a list of remote cluster names are presented.

The `-clusters` option has four keywords:

- **all**: Specifies both local cluster and all remote clusters in the `SNDJOBS_TO` parameter of the target queue in `lsb.queues`. For example:

```
bsub -clusters all -q <send queue>
```

LSF will go through the `SNDJOBS_TO` parameter in `lsb.queues` to check whether asked clusters (except for the local cluster) are members of `SNDJOBS_TO`. If any cluster except the local cluster does not exist in `SNDJOBS_TO`, the job is rejected with an error message.

- **others**: Sends the job to all clusters except for the clusters you specify. For example:

```
bsub -clusters "c1+3 c2+1 others+2"
```

- **~**: Must be used with **all** to indicate the rest of the clusters, excluding the specified clusters.
- **+**: When followed by a positive integer, specifies job level preference for requested clusters. For example:

```
bsub -clusters "c1+2 c2+1"
```

Refer to the IBM Platform LSF Command Reference for details on command syntax.

If the local cluster name is `local_c1`, and `SNDJOBS_TO=q1@rmt_c1 q2@rmt_c2 q3@rmt_c3`, then the requested cluster should be `local_c1` and `rmt_c3`. For example:

```
bsub -clusters "all ~rmt_c1 ~rmt_c2"
```

-clusters local_cluster restricts the job for dispatch to local hosts. To run a job on remote clusters only, use:

```
bsub -clusters "all ~local_cluster"
```

A job that only specifies remote clusters will not run on local hosts.

If there are multiple default queues, then when **bsub -clusters remote_clusters** is issued, the job is sent to the queue whose `SNDJOBS_TO` contains the requested clusters. For example:

```
bsub -clusters "c2" , DEFAULT_QUEUE=q1 q2, q1: SNDJOBS_TO=recvQ1@c1 recvQ2@c3, q2:  
SNDJOBS_TO=recvQ1@c1 recvQ2@c2
```

The job is sent to q2.

Remote cluster equivalency

By default, if no cluster name is specified, LSF utilities such as **lsload** return information about the local cluster.

If you configure a remote cluster to be equivalent to the local cluster, LSF displays information about the remote cluster as well. For example, **lsload** with no options lists hosts in the local cluster and hosts in the equivalent remote clusters.

The following commands automatically display information about hosts in a remote cluster if equivalency is configured:

- **lshosts**
- **lsload**
- **lsplace**
- **lsrun**

Performance limitation

Expect performance in a cluster to decrease as the number of equivalent clusters increases, because you must wait while LSF retrieves information from each remote cluster in turn. Defining all clusters in a large MultiCluster system as equivalent can cause a performance bottleneck as the management host LIM polls all clusters synchronously.

Transparency for users

To make resources in remote clusters as transparent as possible to the user, configure a remote cluster to be equivalent to the local cluster. The users see information about the local and equivalent clusters without having to supply a cluster name to the command.

Hosts in equivalent clusters are all identified by the keyword `remoteHost` instead of the actual host name. For example, **bjobs -p -l** will show `remoteHost@cluster_name` instead of `host_name@cluster_name`.

Simplify MultiCluster administration

If you have many clusters configured to use MultiCluster, create one cluster for administrative purposes, and configure every other cluster to be equivalent to it. This allows you to view the status of all clusters at once, and makes administration of LSF easier.

Configuration

To specify equivalent clusters, set `EQUIV` in the `RemoteClusters` section of `lsf.cluster.cluster_name` to `Y` for the equivalent clusters.

Remote Resources

You can allow the submission forward policy to consider remote resource availability before forwarding jobs. This allows jobs to be forwarded more successfully, but may result in the submission cluster only running local jobs.

Configure remote resource availability

About this task

Submission cluster scheduler considers whether remote resources exist, and only forwards jobs to a queue with free slots or space in the MultiCluster pending job threshold (IMPT_JOBKLG).

To enable a submission forward policy and consider remote resource availability, define `MC_PLUGIN_REMOTE_RESOURCE=y` in `lsf.conf`.

When `MC_PLUGIN_REMOTE_RESOURCE` is defined, the following resource requirements are supported: `-R "type==type_name"`, `-R "same[type]"` and `-R "defined(boolean resource_name)"`.

Note: Both remote resource availability and remote queue workload are considered by the scheduler when the parameter `MC_PLUGIN_SCHEDULE_ENHANCE` is defined.

The submission cluster scheduler considers whether remote resources exist, and only forwards jobs to a queue with free slots or space in the MultiCluster pending job threshold (IMPT_JOBKLG or IMPT_SLOTKLG). Host type and user-defined boolean resources for a host are automatically passed from the execution cluster to the submission cluster.

In some cases, you may want to forward jobs based on numeric or string resources on the execution cluster. For example, different versions of the same application may be installed on different nodes, which are naturally represented as string or numeric resources. Numeric and string resources are only passed back if they are defined in `MC_RESOURCE_MATCHING_CRITERIA` in `lsb.params`. The remote execution cluster makes the submission cluster aware of what resources (and their values) are listed so that the submission cluster can make better job forward decision. Mapping information for the remote receive queue to the execution cluster is also sent back to the submission cluster so that the submission cluster knows which remote queue can access which execution cluster.

Based on MC forwarding concepts, the job is forwarded through a channel between the send queue and the receive queue. One send queue can correspond to multiple receive queues belonging to one or more remote clusters. The forward scheduling finds the best destination queue among several candidate receive queues. Job forward scheduling is usually based on the ownership of resources visible to receive queues instead of whole remote cluster.

Remote job modification

In LSF MC forwarding mode, you can modify almost all attributes for pending jobs (in `PEND` and `PSUSP` status) and some attributes for started jobs (which include jobs in `RUN`, `SSUSP`, `USUSP` and `WAIT` status) from the submission cluster with the **bmod** command.

After `LSB_MOD_ALL_JOBS` in `lsf.conf` is set to `Y`, you can modify the following attributes of a running job:

- `cpu limit` (`[-c cpu_limit[/host_spec] | -cn]`)
- `Memory limit` (`[-M mem_limit | -Mn]`)
- `Rerunnable attribute` (`[-r | -rn]`)
- `Run limit` (`[-W [hour:]minute[/host_name | /host_model] | -Wn]`)
- `Swap limit` (`[-v swap_limit | -vn]`)
- `Standard output/error` (`[-o out_file | -on] [-oo out_file | -oon] [-e err_file | -en] [-eo err_file | -en]`)

Remote queue workload job-forwarding scheduler

Enhanced scheduler decisions can be customized to consider characteristics of remote queues before forwarding a job. Remote queue attributes such as queue priority, number of preemptable jobs, and queue workload are sent to the submission

scheduler. The decisions made by the scheduler, based on this information, depend on the setting of MC_PLUGIN_SCHEDULE_ENHANCE in lsb.params.

Queue workload and configuration is considered in conjunction with remote resource availability (MC_PLUGIN_REMOTE_RESOURCE=Y is automatically set in lsf.conf).

Tip:

Defining MC_PLUGIN_SCHEDULE_ENHANCE as a valid value, the submission scheduler supports the same remote resources as MC_PLUGIN_REMOTE_RESOURCE: -R "type==*type_name*", and -R "same[type]"

Remote queue counter collection

The submission cluster receives up-to-date information about each queue in remote clusters. This information is considered during job forwarding decisions.

Queue information is collected by the submission cluster when MC_PLUGIN_SCHEDULE_ENHANCE (on the submission cluster) is set to a valid value. Information is sent by each execution cluster when MC_PLUGIN_UPDATE_INTERVAL (on the execution cluster) is defined, and the submission cluster is collecting queue information.

Some jobs may be forwarded between counter update intervals. The submission scheduler increments locally stored counter information as jobs are forwarded, and reconciles incoming counter updates to account for all jobs.

The following counter information is collected for each queue:

- Queue ID
- Queue priority
- Total slots: The total number of slots (on all hosts) jobs are dispatched to from this queue. This includes slots on hosts with the status ok, and with the status closed due to running jobs.
- Available slots: The free slots, or slots (out of the total slots) which do not currently have a job running.
- Running slots: The number of slots currently running jobs from the queue.
- Pending slots: The number of slots required by jobs pending on the queue.
- Preemptable available slots: The number of slots the queue can access through preemption.
- Preemptable slots
- Preemptable queue counters (1...*n*):
 - Preemptable queue ID
 - Preemptable queue priority
 - Preemptable available slots

Note:

After a MultiCluster connection is established, counters take the time set in MC_PLUGIN_UPDATE_INTERVAL to update. Scheduling decisions made before this first interval has passed do not accurately account for remote queue workload.

The parameter MC_PLUGIN_SCHEDULE_ENHANCE was introduced in LSF Version 7 Update 6. All clusters within a MultiCluster configuration must be running a version of LSF containing this parameter to enable the enhanced scheduler.

Remote queue selection

The information considered by the job-forwarding scheduler when accounting for workload and remote resources depends on the setting of MC_PLUGIN_SCHEDULE_ENHANCE in lsb.params. Valid settings for this parameter are:

- RESOURCE_ONLY

Jobs are forwarded to the remote queue with the requested resources and the largest (available slots)-(pending slots).

- COUNT_PREEMPTABLE

Jobs are forwarded as with RESOURCE_ONLY, but if no appropriate queues have free slots, the best queue is selected based on the largest (preemptable available slots)-(pending slots).

- COUNT_PREEMPTABLE with HIGH_QUEUE_PRIORITY

Jobs are forwarded as with COUNT_PREEMPTABLE, but jobs are forwarded to the highest priority remote queue.

- COUNT_PREEMPTABLE with PREEMPTABLE_QUEUE_PRIORITY

Jobs are forwarded as with COUNT_PREEMPTABLE, but queue selection is based on which queues can preempt lowest priority queue jobs.

- COUNT_PREEMPTABLE with PENDING_WHEN_NOSLOTS

Jobs are forwarded as with COUNT_PREEMPTABLE, but if no queues have free slots even after preemption, submitted jobs pend.

- COUNT_PREEMPTABLE with HIGH_QUEUE_PRIORITY and PREEMPTABLE_QUEUE_PRIORITY

If no appropriate queues have free slots, the best queue is selected based on:

- queues that can preempt lowest priority queue jobs
- the number of preemptable jobs
- the pending job workload

- COUNT_PREEMPTABLE with HIGH_QUEUE_PRIORITY and PENDING_WHEN_NOSLOTS

If no appropriate queues have free slots, queues with free slots after jobs are preempted are considered.

If no queues have free slots even after preemption, submitted jobs pend.

- COUNT_PREEMPTABLE with PREEMPTABLE_QUEUE_PRIORITY and PENDING_WHEN_NOSLOTS

If no appropriate queues have free slots, queues are considered based on:

- the most free slots after preempting lowest priority queue jobs and preemptable jobs

If no queues have free slots even after preemption, submitted jobs pend.

- COUNT_PREEMPTABLE with HIGH_QUEUE_PRIORITY and PREEMPTABLE_QUEUE_PRIORITY and PENDING_WHEN_NOSLOTS

If no appropriate queues have free slots, queues are considered based on:

- the most free slots after preempting lowest priority queue jobs and preemptable jobs

If no queues have free slots even after preemption, submitted jobs pend.

- DYN_CLUSTER_WEIGHTING

Sets a policy to select the best receiving queue for forwarded jobs. LSF considers queue preference, the queue with the least actual available slots, and the pending ratio in selecting the receiving queue.

In the queue filtering phase, LSF performs an additional check against the IMPT_SLOTBLKG limit in lsb.queues. If a receive queue reaches its IMPT_SLOTBLKG limit, that queue is removed from the candidate queue list.

In the candidate queues ordering phase, LSF orders the candidate receive queues based on whether some queues can meet the job's slot requirements. If some queues can meet the job's slot requirements, the queue that has the highest preference is selected; if multiple queues have the same preference, the queue that has the least number of available job slots is selected as the receive queue. If no queues can meet the job's slot requirements, the queue with the lowest pending ratio is selected; if multiple queues have the same pending ratio, the queue with the highest preference is selected as the receive queue.

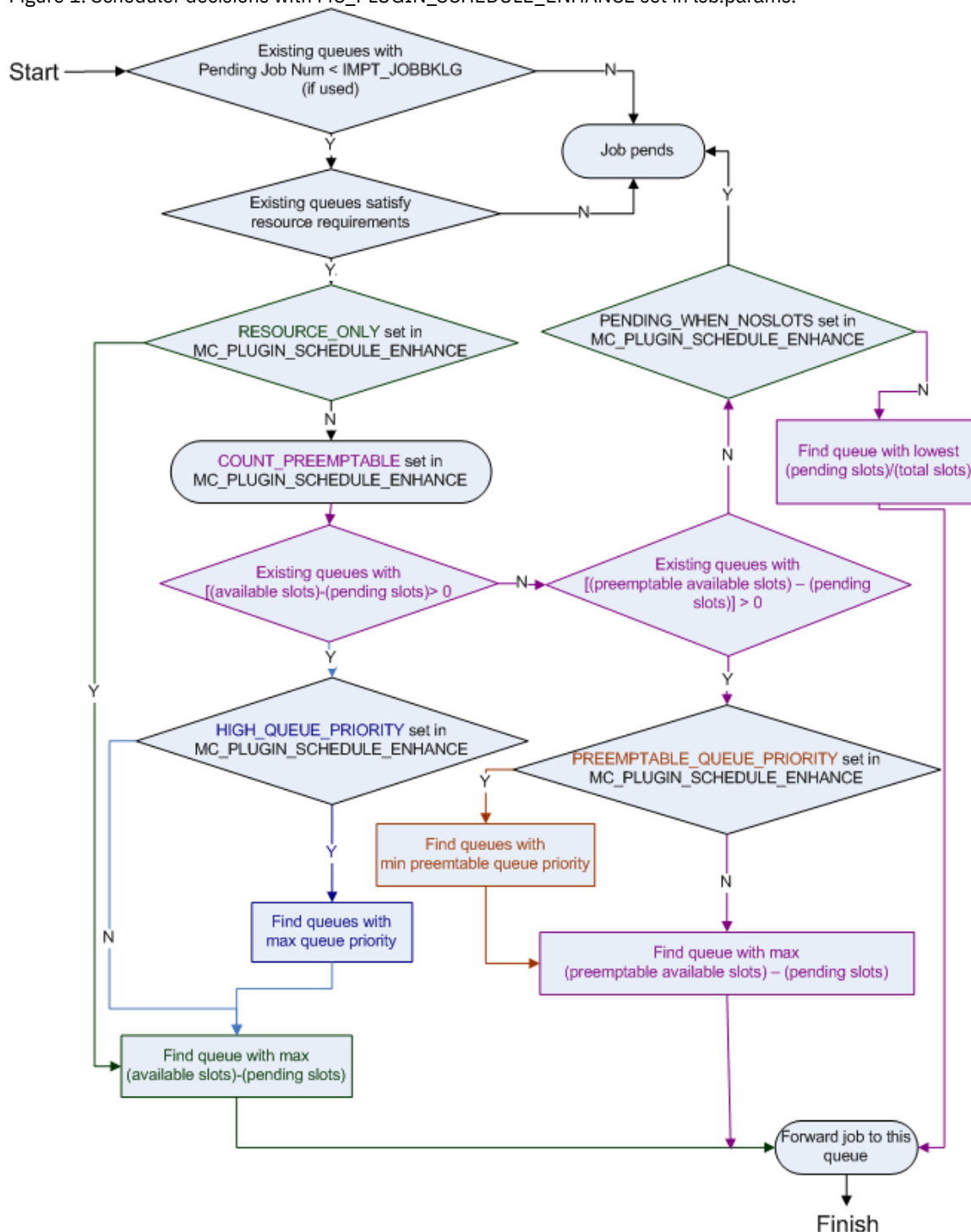
Note: DYN_CLUSTER_WEIGHTING cannot be combined with any other option specified in MC_PLUGIN_SCHEDULE_ENHANCE.

The figure shown illustrates the scheduler decision-making process for valid settings of MC_PLUGIN_SCHEDULE_ENHANCE.

Note:

When the scheduler looks for maximum values, such as for (available slots)-(pending slots), these values can be negative so long as they are within the pending job limit for a receive-jobs queue set by IMPT_JOBBLKG in lsb.queues.

Figure 1. Scheduler decisions with MC_PLUGIN_SCHEDULE_ENHANCE set in lsb.params.



- [Enable queue preference](#)
- [Enable job slot limit](#)

Limitations

Advance reservation

When an advance reservation is active on a remote cluster, slots within the advance reservation are excluded from the number of available slots. Inactive advance reservations do not affect the number of available slots since the slots may still be available for backfill jobs.

Same boolean resource within hostgroups

Hosts in a hostgroup configured without the required same boolean resources can cause ineffectual job-forwarding decisions from the scheduler.

For example, a job may be forwarded to a queue accessing a hostgroup with many slots available, only some of which have the boolean resource required. If there are not enough slots to run the job it will return to the submission cluster, which may continue forwarding the same job back to the same queue.

Same host type within hostgroups

A remote queue hostgroup satisfies host type requirements when any one of the hosts available is the host type requested by a job. As for boolean resources, the submission cluster assumes all slots within a hostgroup are of the same host type. Other hostgroup configurations can result in unexpected job-forwarding decisions.

Configure remote resource and preemptable job scheduling

About this task

Submission cluster scheduler considers whether remote resources exist, and only forwards jobs to a queue with free slots or space in the MultiCluster pending job threshold (IMPT_JOBCLKG).

If no appropriate queues with free slots or space for new pending jobs are found, the best queue is selected based on the number of preemptable jobs and the pending job workload.

Procedure

1. In the submission cluster define `MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE` in `lsb.params`.
2. In the execution cluster set `MC_PLUGIN_UPDATE_INTERVAL` in `lsb.params` to a non-zero value.
3. To make the changes take effect in both the submission and execution clusters run the following command:

```
badmin reconfig
```

Configure remote resource and free slot scheduling

About this task

Submission cluster scheduler considers whether remote resources exist, and only forwards jobs to a queue with free slots or space in the MultiCluster pending job threshold (IMPT_JOBCLKG). If no appropriate queues with free slots or space for new pending jobs are found, the best queue is selected based on which queues can preempt lower priority jobs.

If no queues have free slots even after preemption, jobs pend on the submission cluster.

Procedure

1. In the submission cluster define `MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE PENDING_WHEN_NOSLOTS` in `lsb.params`.
2. In the execution cluster set `MC_PLUGIN_UPDATE_INTERVAL` in `lsb.params` to a non-zero value.
3. To make the changes take effect in both the submission and execution clusters run the following command:

```
badmin reconfig
```

Configure remote resource, preemptable job, and queue priority free slot scheduling

About this task

All scheduler options are configured.

Submission cluster scheduler considers whether remote resources exist, and only forwards jobs to a queue with free slots or space in the MultiCluster pending job threshold (IMPT_JOBKLG).

If no appropriate queues with free slots or space for new pending jobs are found, the best queue is selected based on the number of free slots after preempting low priority jobs and preemptable jobs.

If no queues have free slots even after preemption, jobs pend on the submission cluster.

Procedure

1. In the submission cluster define `MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE HIGH_QUEUE_PRIORITY PREEMPTABLE_QUEUE_PRIORITY PENDING_WHEN_NOSLOTS` in `lsb.params`.
2. In the execution cluster set `MC_PLUGIN_UPDATE_INTERVAL` in `lsb.params` to a non-zero value.
3. To make the changes take effect in both the submission and execution clusters run the following command:

```
badmin reconfig
```

Examples

MultiCluster job forwarding is enabled from a send-queue on Cluster1 to the receive-queues HighPriority@Cluster2 and HighPriority@Cluster3. Both clusters have lower priority queues from running local jobs, and the high priority queues can preempt jobs from the lower priority queues. The scheduler on Cluster1 has the following information about the remote clusters:

Example 1: `MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE`:

Cluster2 (100 total slots)

- queue=HighPriority, priority=60, running slots=20, pending slots=20
- queue=LowPriority, priority=20, running slots=50, pending slots=0

Cluster3 (100 total slots)

- queue=HighPriority, priority=70, running slots=30, pending slots=5
- queue=LowPriority, priority=20, running slots=60, pending slots=0

Cluster2 has a total of 70 running slots out of 100 total slots, with 20 pending slots. The number of (available slots) -(pending slots) for Cluster2 is 10. Cluster3 has a total of 90 running slots out of 100 total slots, with 5 pending slots. The number of (available slots) -(pending slots) for Cluster3 is 5. Thus a job forwarded from Cluster1 is sent to HighPriority@Cluster2.

Example 2: `MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE PREEMPTABLE_QUEUE_PRIORITY`:

Cluster2 (100 total slots)

- queue=HighPriority, priority=50, running slots=20, pending slots=20
- queue=LowPriority, priority=30, running slots=80, pending slots=0

Cluster3 (100 total slots)

- queue=HighPriority, priority=50, running slots=30, pending slots=15
- queue=LowPriority, priority=20, running slots=70, pending slots=0

In both Cluster1 and Cluster2, running jobs occupy all 100 slots. LowPriority@Cluster2 has a queue priority of 30, while LowPriority@Cluster3 has a queue priority of 20. Thus a job forwarded from Cluster1 is sent to HighPriority@Cluster3, where slots can be preempted from the lowest priority queue.

Example 3: `MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE HIGH_QUEUE_PRIORITY PREEMPTABLE_QUEUE_PRIORITY`:

Cluster2 (100 total slots)

- queue=HighPriority, priority=60, running slots=20, pending slots=20
- queue=LowPriority, priority=20, running slots=50, pending slots=0

Cluster3 (100 total slots)

- queue=HighPriority, priority=70, running slots=30, pending slots=5
- queue=LowPriority, priority=20, running slots=60, pending slots=0

Cluster2 has a total of 70 running slots out of 100 total slots, with 20 pending slots. The number of (available slots) -(pending slots) for Cluster2 is 10. Cluster3 has a total of 90 running slots out of 100 total slots, with 5 pending slots. The number of (available slots) -(pending slots) for Cluster3 is 5.

Although (available slots)-(pending slots) is higher for Cluster2, Cluster3 contains a higher priority queue. Thus a job forwarded from Cluster1 is sent to HighPriority@Cluster3.

Example 4: MC_PLUGIN_SCHEDULE_ENHANCE=COUNT_PREEMPTABLE_HIGH_QUEUE_PRIORITY
PREEMPTABLE_QUEUE_PRIORITY:

Cluster2 (100 total slots)

- queue=HighPriority, priority=60, running slots=20, pending slots=20
- queue=LowPriority, priority=20, running slots=80, pending slots=0

Cluster3 (100 total slots)

- queue=HighPriority, priority=60, running slots=30, pending slots=5
- queue=LowPriority, priority=20, running slots=70, pending slots=0

In both Cluster1 and Cluster2, running jobs occupy all 100 slots. In this case (preemptable available slots)-(pending slots) is considered. For HighPriority@Cluster2 this number is (80-20)=60; for HighPriority@Cluster3 this number is (70-5)=65. Both queues have the same priority, thus a job forwarded from Cluster1 is sent to HighPriority@Cluster3.

Enable queue preference

Set MC_PLUGIN_SCHEDULE_ENHANCE=DYN_CLUSTER_WEIGHTING in lsb.params on the submission cluster to select the best receiving queue for the forwarded the job. For queue preference weighting to take effect, add preference values to the queues that are specified in SNDJOBS_TO. LSF considers queue preference, the queue with the least actual available slots, and the pending ratio in selecting the receiving queue.

- [Configure queue preference](#)

Configure queue preference

About this task

MultiCluster job forwarding model only.

To set up queue preference, do the following:

Procedure

1. Enable queue preference weighting.
Define MC_PLUGIN_SCHEDULE_ENHANCE=DYN_CLUSTER_WEIGHTING in lsb.params on the submission cluster.

Use **bparams -a** or **bparams -l** to display the configuration of MC_PLUGIN_SCHEDULE_ENHANCE.

2. On the execution cluster, define MC_PLUGIN_UPDATE_INTERVAL to a positive integer in lsb.params.
3. Edit lsb.queues and add the preference values to the queues that are specified in SNDJOBS_TO.
Specify a preference value as an integer between 0 and 2147483647. The larger the number, the higher the preference for that queue. Higher values increase the possibility of forwarding a job to the receiving queue. A single plus sign '+' indicates a queue preference of 1.

For example:

```
Begin Queue
...
SNDJOBS_TO = recvq2@cluster3+ recvq1@cluster4+3 recvq1@cluster5+6
...
End Queue
```

In this example, the preference for queue recvq2@cluster3 is 1, recvq1@cluster4 is 3 and recvq1@cluster5 is 6.

4. Use **bqueues -l** to display the queue preference in the receiving queues.

Enable job slot limit

Configure IMPT_SLOTBKLG in lsb.queues to specify how many slots the forwarded pending jobs can occupy on the receiving queue.

- [Configure pending job slot limit](#)

Configure pending job slot limit

About this task

MultiCluster job forwarding model only.

To set up a pending job slot limit on a receiving queue, do the following:

Procedure

1. Edit lsb.queues on the execution cluster, and define IMPT_SLOTBKLG on any receiving queue.
IMPT_SLOTBKLG specifies the MultiCluster pending job slot limit for a receive-jobs queue. In the submission cluster, if the total of requested job slots and the number of imported pending slots in the receiving queue is greater than IMPT_SLOTBKLG, the queue stops accepting jobs from remote clusters, and the job is not forwarded to the receiving queue.

Specify an integer between 0 and 2147483646 for the number of slots.

Set IMPT_SLOTBKLG to 0 to forbid any job being forwarded to the receiving queue, or use the keyword infinit to make the queue accept an unlimited number of pending MultiCluster job slots.

For example:

```
Begin Queues
...
RCVJOBS_FROM = cluster1 cluster2
IMPT_SLOTBKLG = 100
...
End Queues
```

2. Use **bqueues -l** to display the value of IMPT_SLOTBKLG.

Pre-exec retry threshold

When a job has a pre-execution command, LSF runs the job's pre-execution command first. By default, LSF retries the pre-execution command five times.

With a threshold configured, LSF returns the entire job to the submission cluster if the pre-execution command fails to run after a certain number of attempts. The submission cluster can then reschedule the job.

Configure pre-exec retries

To limit the number of times the local cluster attempts to run the pre-execution command, set `LOCAL_MAX_PREEEXEC_RETRY` in `lsb.params` and specify the maximum number of attempts. Configure `MAX_PREEEXEC_RETRY` or `REMOTE_MAX_PREEEXEC_RETRY` to limit pre-execution retry attempts on the remote cluster.

The pre-execution command retry limit configured in `lsb.params`, `lsb.queues`, and `lsb.applications` on the execution cluster is applied.

Retry threshold and suspend notification

If a job is forwarded to a remote cluster and then fails to start, it returns to the submission queue and LSF retries the job. After a certain number of failed retry attempts, LSF suspends the job (PSUSP). The job remains in that state until the job owner or administrator takes action to resume, modify, or remove the job.

By default, LSF tries to start a job up to 6 times (the threshold is 5 retry attempts). The retry threshold is configurable.

You can also configure LSF to send email to the job owner when the job is suspended. This allows the job owner to investigate the problem promptly. By default, LSF does not alert users when a job has reached its retry threshold.

Configure retries

Set `LSB_MC_INITFAIL_RETRY` in `lsf.conf` and specify the maximum number of retry attempts. For example, to attempt to start a job no more than 3 times in total, specify 2 retry attempts:

```
LSB_MC_INITFAIL_RETRY = 2
```

Configure mail notification

To make LSF email the user when a job is suspended after reaching the retry threshold, set `LSB_MC_INITFAIL_MAIL` in `lsf.conf` to `y`:

```
LSB_MC_INITFAIL_MAIL = y
```

By default, LSF does not notify the user.

Pending MultiCluster job limit

The pending MultiCluster job limit determines the maximum number of MultiCluster jobs that can be pending in the queue. The queue rejects jobs from remote clusters when this limit is reached. It does not matter how many MultiCluster jobs are running in the queue, or how many local jobs are running or pending.

By default, the limit is 50 pending MultiCluster jobs.

Configure a pending MultiCluster job limit

Edit `IMPT_JOBKLG` in `lsb.queues`, and specify the maximum number of MultiCluster jobs from remote clusters that can be pending in the queue. This prevents jobs from being over-committed to an execution cluster with limited resources.

If you specify the keyword `infinite`, the queue will accept an infinite number of jobs.

Considerations

When you set the limit, consider the following:

- Make sure there are enough pending jobs in the queue for LSF to dispatch, in order to make full use of the execution servers. If you use advance reservation, set the limit higher to allow for the pending jobs that are waiting to use a reservation.
- Make sure the queue does not fill up with so many MultiCluster jobs that LSF cannot dispatch them all in the near future.

Therefore, estimate your expected job flow and set the limit 50% or 100% higher than the estimate.

Example

Assume that locally submitted jobs do not occupy all the available resources, so you estimate that each processor can schedule and execute 2 MultiCluster jobs per scheduling session. To make full use of the job slots, and make sure the queue never runs out of jobs to dispatch, set the limit at 3 or 4 jobs per processor: if this queue has 20 processors, set the limit to allow 60 or 80 MultiCluster jobs pending. You expect to run about 40 of them immediately, and the remainder only wait for one scheduling cycle.

Update pending reason for MultiCluster jobs

By default, the pending reasons for MultiCluster jobs are updated every 5 minutes by the execution cluster, but the maximum amount of data transferred between clusters is 512 KB. If LSF cannot update the pending reasons for all jobs at once, it will update the additional jobs during the next cycles.

You can disable the feature or modify how often the pending reasons are updated and how much data can be transferred at one time. Depending on load, updating the information very frequently or sending an unlimited amount of information can affect the performance of LSF.

- [Configure the pending reason updating interval](#)
- [Configure the pending reason update package size](#)

Configure the pending reason updating interval

About this task

Change the timing of pending reason updating between clusters.

Procedure

1. Set `MC_PENDING_REASON_UPDATE_INTERVAL` in `lsb.params` in the execution cluster.
2. Specify how often to update the information in the submission cluster, in seconds.
To disable pending reason updating between clusters, specify zero:

```
MC_PENDING_REASON_UPDATE_INTERVAL=0
```

Configure the pending reason update package size

About this task

Change the package size of each pending reason update.

Procedure

1. Set MC_PENDING_REASON_PKG_SIZE in lsb.params in the execution cluster.
2. Specify the maximum package size, in KB.
To disable the limit and allow any amount of data in one package, specify zero:

```
MC_PENDING_REASON_PKG_SIZE=0
```

This parameter has no effect if pending reason updating is disabled (MC_PENDING_REASON_UPDATE_INTERVAL=0).

Remote timeout limit

The remote timeout limit is set in the submission cluster and determines how long a MultiCluster job stays pending in the execution cluster. After the allowed time, the job returns to the submission cluster to be rescheduled.

The remote timeout limit in seconds is:

```
MAX_RSCHED_TIME(lsb.queues) * MBD_SLEEP_TIME(lsb.params)
```

In a default installation, MBD_SLEEP_TIME is 20 seconds and the multiplying factor for MultiCluster is 20, so the timeout limit is normally 400 seconds.

Problem with remote-only queues

By default, LSF queues dispatch jobs in FCFS order. However, there is one case in which the default behavior can be a problem. This is when a send-jobs queue sends to only one remote queue, and never uses local hosts.

In this case, jobs that time out in the receive-jobs cluster can only be re-dispatched to the same receive-jobs queue. When this happens, the receive-jobs queue takes the re-dispatched job as a new submission, gives it a new job ID, and gives it lowest priority in FCFS ordering. In this way, the highest-priority MultiCluster job times out and then becomes the lowest-priority job. Also, since local jobs don't time out, the MultiCluster jobs get a lower priority than local jobs that have been pending for less time.

To make sure that jobs are always dispatched in the original order, you can disable remote timeout for the send-jobs queue.

Disable timeout

To disable remote timeout, edit MAX_RSCHED_TIME in lsb.queues in the submission cluster, and specify the keyword INFINIT. This increases the remote timeout limit to infinity.

Even if the limit is set to infinity, jobs time out if a remote execution cluster gets reconfigured. However, all the pending jobs time out at once, so when the queue attempts to send them again, the original priority is maintained.

Enable job priority in MultiCluster job forward mode

User-assigned job priority is supported in MultiCluster job forwarding mode. Use **bsub -sp** to specify a user-assigned job priority that orders all jobs from all users in a queue. Jobs submitted with **-sp** are passed to the execution cluster with the

specified priority.

- [Specify a job priority \(bsub -sp\)](#)
- [Configure maximum job priority](#)

Specify a job priority (bsub -sp)

Valid values for priority specified by **bsub -sp** are any integers between 1 and MAX_USER_PRIORITY (configured in lsb.params, displayed by **bparams -l**). Job priorities that are not valid are rejected. LSF and queue administrators can specify priorities beyond MAX_USER_PRIORITY.

Job owners can change the priority of their own jobs. LSF and queue administrators can change the priority of all jobs in a queue.

Job order is the first consideration to determine job eligibility for dispatch. Jobs are still subject to all scheduling policies regardless of job priority. Jobs are scheduled based on priority and order in the queue, in the following order:

1. Queue priority
2. Job priority
3. First-come first-served order in the queue

Administrators can configure user-assigned job priority with automatic job priority escalation to automatically increase the priority of jobs that have been pending for a specified period of time (JOB_PRIORITY_OVER_TIME in lsb.params). For example, JOB_PRIORITY_OVER_TIME=1/1 increases the job priority of a pending job by 1 every minute.

For job priority escalation to take effect on pending jobs in the execution cluster, JOB_PRIORITY_OVER_TIME must be defined in lsb.params in the execution cluster. MAX_USER_PRIORITY must be defined in lsb.params in the execution cluster for the job priority specified in submission cluster to be propagated to execution cluster.

Note:

The **btop** and **bbot** commands only move jobs within the queue relative to other jobs with the same priority. These commands do not change job priority.

Job priority scaling

If both clusters have different values for MAX_USER_PRIORITY, the execution cluster will scale up or down when the job is forwarded to the execution cluster.

Job priority scaling is based on the following formula:

$$\text{job_priority} = \text{submission_job_priority} * \text{MAX_USER_PRIORITY_exec_cluster} / \text{MAX_USER_PRIORITY_sub_cluster}$$

Where:

- *submission_job_priority* is the job priority in the submission cluster
- *job_priority* is the calculated job priority in submission cluster based on the submission job priority, the MAX_USER_PRIORITY of the execution cluster and the MAX_USER_PRIORITY of the submission cluster

The scaled job priority cannot drop below 1.

The **bjobs -l** command from the submission cluster displays the initial job priority and the dynamic job priority of submission cluster. It does not display dynamic job priority for execution cluster.

For example, if MAX_USER_PRIORITY in the submission cluster is 10 and MAX_USER_PRIORITY in the execution cluster is 20, a job submitted with a job priority of 4 will have a job priority of 8 in the execution cluster.

Configure maximum job priority

About this task

User-assigned job priorities are supported in MultiCluster job forwarding mode. Jobs submitted with **-sp** are passed to the execution cluster with the specified priority.

To set up maximum user-assigned job priority, do the following:

Procedure

1. Configure maximum user-assigned job priority in both the submission cluster and the execution cluster. Edit `lsb.params` and define `MAX_USER_PRIORITY`. This configuration applies to all queues in the cluster.

MAX_USER_PRIORITY=integer

Specifies the maximum priority a user can assign to a job. Specify a positive integer. Larger values represent higher priority. 1 is the lowest priority. For example, `MAX_USER_PRIORITY=100` specifies that 100 is the maximum job priority that a user can specify. LSF and queue administrators can assign a job priority higher than the `MAX_USER_PRIORITY` value for jobs they own.

2. Use **bparams -l** to display the value of `MAX_USER_PRIORITY`.

Enhance fair share calculation to include the job forwarding mode

You can enhance the calculations for the fair share policies to account for the forwarding mode in the LSF multicluster capability, which charges the user priority for each job forwarded, providing a fair opportunity for all users to forward their jobs. This policy charges the priority of the user whose job is successfully forwarded to the remote execution cluster, and restores the charge for the user whose job is returned from the remote execution cluster.

See more information on global fair share scheduling in [Global fair share scheduling](#).

Enabling and configuring fair share scheduling for the LSF multicluster capability job forwarding model

Include forwarded jobs for the LSF multicluster capability in the fair share calculations by defining the following parameters:

- `FWD_JOB_FACTOR=number` in the `lsb.params` or `lsb.queue` files.
Specify the weighting factor to indicate the relative importance of the number of forwarded job slots in the user priority calculation. Set this parameter cluster-wide in the `lsb.params` file, or for an individual queue in the `lsb.queue` file. If defined in both files, the queue value takes precedence. To treat remote jobs and local jobs as the same, set `FWD_JOB_FACTOR` to the same value as `RUN_JOB_FACTOR` unless you want a different weighting factor for forwarded jobs.
- `LSF_MC_FORWARD_fair share_CHARGE_DURATION=seconds` in the `lsf.conf` file.
When accounting for forwarded jobs in the fair share calculations, job usage might be counted twice if global fair share is used because job usage is counted on the submission cluster, then counted again when the job is running on a remote cluster. To avoid this problem, specify the duration of time after which LSF removes the forwarded jobs from the user priority calculation for fair share scheduling by specifying the `LSF_MC_FORWARD_fair share_CHARGE_DURATION` parameter in the `lsf.conf` file. If you enabled global fair share and intend to use the new forwarded job slots factor, set the value of `LSF_MC_FORWARD_fair share_CHARGE_DURATION` to double the value of the `SYNC_INTERVAL` parameter in the `lsb.globalpolicies` file (approximately 5 minutes) to avoid double-counting the job usage for forwarded jobs. If global fair share is not enabled, this parameter is not needed.

IBM Spectrum LSF multicluster capability resource leasing model

In the resource leasing model, two clusters agree that one cluster will borrow resources from the other, taking control of the other cluster's resources.

Note: This capability is deprecated and might not be available or might be replaced by alternate functionality in future releases.

- [Lease model overview](#)
- [Using the lease model](#)
- [Special considerations under resource leasing model](#)
- [Resource export](#)
- [Create an export policy](#)
- [Export workstations](#)
- [Export special hosts](#)
- [Export other resources](#)
- [Export shared resources](#)
- [Shared lease](#)
- [Borrow resources](#)
- [Parallel jobs and the lease model](#)

Lease model overview

About this task

Two clusters agree that one cluster will borrow resources from the other, taking control of the resources. Both clusters must change their configuration to make this possible, and the arrangement, called a “lease”, does not expire, although it might change due to changes in the cluster configuration.

With this model, scheduling of jobs is always done by a single cluster. When a queue is configured to run jobs on borrowed hosts, LSF schedules jobs as if the borrowed hosts actually belonged to the cluster.

Procedure

1. Setup:
 - A resource provider cluster “exports” hosts, and specifies the clusters that will use the resources on these hosts.
 - A resource consumer cluster configures a queue with a host list that includes the borrowed hosts.
2. To establish a lease:
 - a. Configure two clusters properly (the provider cluster must export the resources, and the consumer cluster must have a queue that requests remote resources).
 - b. Start up the clusters.
 - c. In the consumer cluster, submit jobs to the queue that requests remote resourceAt this point, a lease is established that gives the consumer cluster control of the remote resources.
 - If the provider did not export the resources requested by the consumer, there is no lease. The provider continues to use its own resources as usual, and the consumer cannot use any resources from the provider.
 - If the consumer did not request the resources exported to it, there is no lease. However, when entire hosts are exported the provider cannot use resources that it has exported, so neither cluster can use the resources; they will be wasted.
3. Changes to the lease:

- The lease does not expire. To modify or cancel the lease, you should change the export policy in the provider cluster.
- If you export a group of workstations allowing LSF to automatically select the hosts for you, these hosts do not change until the lease is modified. However, if the original lease could not include the requested number of hosts, LSF can automatically update the lease to add hosts that become available later on.
- If the configuration changes and some resources are no longer exported, jobs from the consumer cluster that have already started to run using those resources will be killed and requeued automatically.

If LSF selects the hosts to export, and the new export policy allows some of the same hosts to be exported again, then LSF tries to re-export the hosts that already have jobs from the consumer cluster running on them (in this case, the jobs continue running without interruption). If LSF has to kill some jobs from the consumer cluster to remove some hosts from the lease, it selects the hosts according to job run time, so it kills the most recently started jobs.

Using the lease model

Submit jobs

LSF will automatically schedule jobs on the available resources, so jobs submitted to a queue that uses borrowed hosts can automatically use the borrowed resources.

bsub

To submit a job and request a particular host borrowed from another cluster, use the format *host_name@cluster_name* to specify the host. For example, to run a job on *hostA* in *cluster4*:

```
bsub -q myqueue -m hostA@cluster4 myjob
```

This will not work when you first start up the MultiCluster grid; the remote host names are not recognized until the lease has been established.

bmod

The **bmod** syntax also allows you to specify borrowed hosts in the same format *host_name@cluster_name*.

Administration

badmin

The administrator of the consumer cluster can open and close borrowed hosts using **badmin**. Use the format *host_name@cluster_name* to specify the borrowed host. This action only affects scheduling on the job slots that belong to that consumer cluster. For example, if slots on a host are shared among multiple consumers, one consumer can close the host, but the others will not be affected or be aware of any change.

You must be the administrator of the provider cluster to shut down or start up a host. This action will affect the consumer cluster as well.

Host groups or host partitions

When you define a host group in *lsb.hosts*, or a host partition, you can use the keyword *allremote* to indicate all borrowed hosts available to the cluster. You cannot define a host group that includes borrowed hosts specified by host name or cluster name.

Compute units

Compute units defined in *lsb.hosts* can use wild cards to include the names of borrowed hosts available to the cluster. You cannot define a host group that includes borrowed hosts specified by host name or cluster name directly. Hosts running LSF 7 Update 4 or earlier cannot satisfy compute unit resource requirements, and thus cannot be included in compute units.

Automatic retry limits

The pre-execution command retry limit (*MAX_PREEEXEC_RETRY* and *REMOTE_MAX_PREEEXEC_RETRY*), job requeue limit (*MAX_JOB_REQUEUE*), and job preemption retry limit (*MAX_JOB_PREEMPT*) configured in *lsb.params*, *lsb.queues*, and *lsb.applications* apply to jobs running on remote leased hosts as if they are running on local hosts

bhosts

By default, **bhosts** only shows information about hosts and resources that are available to the local cluster and information about jobs that are scheduled by the local cluster. Therefore, borrowed resources are included in the summary, but exported resources are not normally included (the exception is reclaimed resources, which are shown during the times that they are available to the local cluster).

For borrowed resources, the host name is displayed in the format *host_name@cluster_name*. The number of job slots shown is the number available to the consumer cluster, the JL/U and host status shown is determined by the consumer cluster, and the status shown is relative to the consumer cluster. For example, the consumer might see closed or closed_Full status, while the provider sees ok status.

- Cluster1 has borrowed one job slot on hostA. It shows the borrowed host is closed because that job slot is in use by a running job.

```
bhosts
HOST_NAME      STATUS  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV
hostA@cluster2 closed   -    1    1     1     0     0     0
```

- Cluster2 has kept 3 job slots on hostA for its own use. It shows the host is open, because all the available slots are free.

```
bhosts
HOST_NAME      STATUS  JL/U  MAX  NJOBS  RUN  SSUSP  USUSP  RSV
hostA          ok     -    3    0     0     0     0     0
```

bhosts -e

This option displays information about the exported resources. The provider cluster does not display JL/U or host status; this status information is determined by the consumer cluster and does not affect the provider.

bhosts -e -s

This option displays information about exported shared resources.

bjobs

The **bjobs** command shows all jobs associated with hosts in the cluster, including MultiCluster jobs. Jobs from remote clusters can be identified by the FROM_HOST column, which shows the remote cluster name and the submission or consumer cluster job ID in the format *host_name@remote_cluster_name:remote_job_ID*.

If the MultiCluster job is running under the job forwarding model, the QUEUE column shows a local queue, but if the MultiCluster job is running under the resource leasing model, the name of the remote queue is shown in the format *queue_name@remote_cluster_name*.

You can use the local job ID and cluster name (for example, `bjobs 123@submission_cluster_name`) to see the job IDs for the submission, execution and lease clusters. For job arrays, the query syntax is `bjobs "submission_job_id[index]"@submission_cluster_name`.

Use **-w** or **-l** to prevent the MultiCluster information from being truncated.

bclusters

For the resource leasing model, **bclusters** shows information about each lease.

- Status
 - ok means that the resources are leased and the resources that belong to the provider are being used by the consumer.
 - conn indicates that a connection has been established but the lease has not yet started; probably because the consumer has not yet attempted to use the shared resources. The conn status remains until jobs are submitted, at which point the status changes to ok. If this status persists in a production environment, it could mean that the consumer cluster is not properly configured.
 - disc indicates that there is no connection between the two clusters.
- Resource flow
 - For resources exported to another cluster, the resource flow direction is "EXPORT", and the remote cluster specified is the consumer of the resources.
 - For resources borrowed from another cluster, the resource flow direction is IMPORT, and the remote cluster specified is the resource provider.

bmgroup

The **bmgroup** command displays compute units, host groups, host names, and administrators for each group or unit. For the resource leasing model, host groups with leased-in hosts are displayed by default as a list of hosts in the form `host_name@cluster_name` (for example, `hosta@cluster1`) in the `HOSTS` column.. If `LSB_BMGROUP_ALLREMOTE_EXPAND=N` is configured in `lsf.conf` or as an environment variable, leased-in hosts are represented by a single keyword `allremote` instead of being displayed as a list. For example, if `cluster_1` has defined a host group called `management_hosts` that contains only `host_A`, and a host group called `remote_hosts` with leased-in hosts as members, and `cluster_2` contains `host_B` and `host_C` that are both being leased in by `cluster_1`:

By default, the `remote_hosts` host group contains leased-in hosts and displays `allremote` in the `HOSTS` column.

GROUP_NAME	HOSTS
<code>management_hosts</code>	<code>host_A</code>
<code>remote_hosts</code>	<code>allremote</code>

If `LSB_BMGROUP_ALLREMOTE_EXPAND=Y` is configured in `lsf.conf`, the `HOSTS` column displays a list of leased-in hosts:

GROUP_NAME	HOSTS
<code>management_hosts</code>	<code>host_A</code>
<code>remote_hosts</code>	<code>host_B@cluster_2 host_C@cluster_2</code>

Special considerations under resource leasing model

Resizable jobs

Resizable jobs across MultiCluster clusters is not supported. This implies the following behavior:

- For the lease model, the initial allocation for the job may contain lease hosts. But once the job allocation includes a leased host, LSF does not generate a pending allocation request. LSF does not allocate any leased hosts to pending allocation requests.
- You cannot run **bresize** commands to shrink allocations from submission clusters in either the lease model or job forwarding model

Checkpointing

Checkpointing is not supported if a job runs on a leased host.

Resource export

lsb.resources file

The `lsb.resources` file contains MultiCluster configuration information for the lease model, including the export policies which describe the hosts and resources that are exported, and the clusters that can use them.

You must reconfigure the cluster to make the configuration take effect.

Resources that can be exported

Job slots

To export resources, you must always export job slots on hosts, so that the consumer cluster can start jobs on the borrowed hosts.

Additional host-based resources

By default, all the jobs on a host compete for its resources. To help share resources fairly when a host's job slots are divided among multiple clusters, you can export quantities of memory and swap space, also for the use of the consumer cluster.

Shared resources

By default, shared resources are not exported. You can create a separate policy to export these resources.

Who can use exported resources

The export policy defines the consumers of exported resources. By default, resources that are exported can be used by the provider; this applies to job slots on a host and also to resources like memory.

With resource reclaim, exported job slots can be reclaimed by the provider if the consumer is not using them to run jobs. In this way, the provider can share in the use of the exported job slots.

Create an export policy

An export policy defined in `lsb.resources` is enclosed by the lines:

```
Begin HostExport
...
End HostExport
```

In each policy, you must specify which hosts to export, how many job slots, and distribution of resources. Optionally, you can specify quantities of memory and swap space.

Tip:

To export hosts of HostExport Type==DLINUX, specifying swap space is mandatory.

Configure as many different export policies as you need.

Each export policy corresponds to a separate lease agreement.

Export policy examples

This simple export policy exports a single job slot on a single host to a single consumer cluster:

```
Begin HostExport
PER_HOST=HostA
SLOTS=1
DISTRIBUTION=([Cluster5, 1])
End HostExport
```

This simple policy exports all the resources on a single Linux host to a single consumer cluster:

```
Begin HostExport
RES_SELECT=type==LINUX
NHOSTS=1
DISTRIBUTION=([Cluster5, 1])
End HostExport
```

Export hosts

To export resources such as job slots or other resources, you must specify which hosts the resources are located on. There are two ways to specify which hosts you want to export: you can list host names, or you can specify resource requirements and let LSF find hosts that match those resource requirements. The method you use to specify the exported hosts determines the method that LSF uses to share the hosts among competing consumer clusters.

Export a large number of hosts

If you have a group of similar hosts, you can share a portion of these hosts with other clusters. To choose this method, let LSF automatically select the hosts to export. The group of hosts can be shared among multiple consumer clusters,

but each host is leased to only one consumer cluster, and all the job slots on the host are exported to the consumer.

Share a large computer

You can share a powerful multiprocessor host among multiple clusters. To choose this method, export one or more hosts by name and specify the number of job slots to export. The exported job slots on each host are divided among multiple consumer clusters.

Distribute exported resources

An export policy exports specific resources. The distribution statement in `lsb.resources` partitions these resources, assigning a certain amount exclusively to each consumer cluster. Clusters that are not named in the distribution list do not get to use any of the resources exported by the policy.

The simplest distribution policy assigns all of the exported resources to a single consumer cluster:

```
DISTRIBUTION=([Cluster5, 1])
```

Distribution list syntax

The syntax for the distribution list is a series of share assignments. Enclose each share assignment in square brackets, as shown, and use a space to separate multiple share assignments. Enclose the full list in parentheses:

```
DISTRIBUTION=([share_assignment]...)
```

Share assignment syntax

The share assignment determines what fraction of the total resources is assigned to each cluster.

The syntax of each share assignment is the cluster name, a comma, and the number of shares.

`[cluster_name, number_shares]`

- *cluster_name*

Specify the name of a cluster allowed to use the exported resources.

- *number_shares*

Specify a positive integer representing the number of shares of exported resources assigned to the cluster.

The number of shares assigned to a cluster is only meaningful when you compare it to the number assigned to other clusters, or to the total number. The total number of shares is just the sum of all the shares assigned in each share assignment.

Examples

- In this example, resources are leased to 3 clusters in an even 1:1:1 ratio. Each cluster gets 1/3 of the resources.

```
DISTRIBUTION=([C1, 1] [C2, 1] [C3, 1])
```

- In this example, resources are leased to 3 clusters in an uneven ratio. There are 5 shares assigned in total, so C1 gets 2/5 of the resources, C2 gets the same, and C3 gets 1/5 of the resources.

```
DISTRIBUTION=([C1, 2] [C2, 2] [C3, 1])
```

Export workstations

About this task

These steps describe the way to share part of a large farm of identical hosts. This is most useful for reallocating resources among different departments, to meet a temporary need for more processing power.

Procedure

1. Create the new policy.
 2. Specify the hosts that are affected by the policy. Each host is entirely exported; the provider cluster does not save any job slots on the exported hosts for its own use.
 3. Specify the distribution policy. This determines which clusters share in the use of the exported job slots.
 4. Optional. Share additional resources (any combination of memory, swap space, or shared resources).
- [Distribution policy for automatically selected hosts](#)

Allow LSF to select the hosts you want to export

Procedure

1. Specify both RES_SELECT and NHOSTS in lsb.resources.
2. For RES_SELECT, specify the selection criteria using the same syntax as the “select” part of the resource requirement string (normally used in the LSF **bsub** command).
For details about resource selection syntax, see Administering IBM Platform LSF. For this parameter, if you do not specify the required host type, the default is “type==any”.
3. For NHOSTS, specify a maximum number of hosts to export.

```
Begin HostExport
RES_SELECT=type==LINUX
NHOSTS=4
```

In this example, we want to export 4 Linux hosts. If the cluster has 5 Linux hosts available, 4 are exported, and the last one is not exported. If the cluster has only 3 Linux hosts available at this time, then only 3 hosts are exported, but LSF can update the lease automatically if another host becomes available to export later on.

4. Use **lshosts** to view the host types that are available in your cluster.

Distribution policy for automatically selected hosts

For syntax of the distribution policy, see “Distributing exported resources”.

When you export hosts by specifying the resource selection statement, multiple hosts are divided among multiple consumer clusters, but each host is entirely exported to a single consumer cluster. All the job slots on a host are exported to the consumer cluster, along with all its other host-based resources including swap space and memory.

Example

```
Begin HostExport
RES_SELECT=type==LINUX
NHOSTS=2
DISTRIBUTION=([C1, 1] [C2, 1])
End HostExport
```

In this example, 2 hosts that match the resource requirements are selected, suppose they are HostA and HostB, and each has 2 job slots. All job slots on each host are exported. Resources are shared evenly among 2 clusters, each cluster gets 1/2 of the resources.

Since the hosts are automatically selected, the hosts are distributed to only one consumer cluster, so the first host, HostA, goes to Cluster1, and the second host, HostB, goes to Cluster2. Assume each host has 2 job slots for use by the consumer cluster. Cluster1 gets 2 job slots on HostA, and Cluster2 gets 2 job slots on HostB.

In this example there is an even distribution policy, but it is still possible for one consumer cluster to get more resources than the other, if the exported hosts are not all identical.

Export special hosts

About this task

These steps describe the way to share a large multiprocessor host among multiple clusters. This is most useful for allowing separate departments to share the cost and use of a very powerful host. It might also be used to allow multiple clusters occasional access to a host that has some unique feature.

Procedure

1. Create the new policy.
2. Specify the hosts that are affected by the policy.
3. Specify how many job slots you want to export from each host. Optionally, reduce the number of job slots available to the local cluster by the same amount.
4. Specify the distribution policy. This determines which clusters share in the use of the exported job slots.
5. Optional. Share additional resources (any combination of memory, swap space, or shared resources).

- [Distribution policy for named hosts](#)

Name the hosts you want to export

Procedure

Specify the name of a host in the PER_HOST parameter in lsb.resources:

```
Begin HostExport
PER_HOST=HostA
```

If you specify multiple hosts, this policy will apply to all the hosts you specify:

```
Begin HostExport
PER_HOST=HostA HostB HostC
```

Control job slots

Procedure

Use the SLOTS parameter to specify the number of job slots to export from each host.

By default, the provider can still run the usual number of jobs at all times. The additional jobs that the consumer clusters are allowed to start might overload the host. If you are concerned with keeping the host's performance consistent, reduce the job slot configuration in the local cluster to compensate for the number of slots exported to remote clusters.

Example

For example, this policy exports 4 job slots on each host:

```
Begin HostExport
PER_HOST=HostA HostB
SLOTS=4
```

- Default configuration of lsb.hosts in the provider cluster:

HOST_NAME	MXJ
HostA	6
HostB	8

- How you can update lsb.hosts to compensate for the exported job slots:

HOST_NAME	MXJ
HostA	2
HostB	4

Distribution policy for named hosts

For syntax of the distribution policy, see “Distributing exported resources”.

When you export hosts by specifying host names, the job slots on each host are divided among multiple consumer clusters, so each cluster gets a part of each host.

Example

```
Begin HostExport
PER_HOST=HostA HostB
SLOTS=2
DISTRIBUTION=([C1, 1] [C2, 1])
End HostExport
```

In this example, 2 job slots are exported from HostA and HostB. Resources are shared evenly among 2 clusters, so each cluster is entitled to 1/2 of the resources.

Because the hosts are specified by name, the distribution policy is applied at the job slot level. The first job slot on HostA goes to Cluster1, and the second job slot on HostA goes to Cluster2. Similarly, one job slot on HostB goes to Cluster1, and the other job slot on HostB goes to Cluster2. Each consumer cluster can start 2 jobs, one on HostA, and one on HostB.

The provider cluster can always use the number of job slots that are configured in the provider cluster (no matter how many slots are exported). You might want to adjust the configuration of the provider cluster after exporting hosts and reduce the number of job slots (MXJ in lsb.hosts); otherwise, you might notice a difference in performance because of the extra jobs that can be started by the consumer clusters.

Export other resources

Once you have exported a host, you can export memory and swap space in addition to job slots.

By default, the consumer cluster borrows a job slot but is not guaranteed that there will be free memory or swap space, because all jobs on the host compete for the host's resources. If these resources are exported, each consumer cluster schedules work as if only the exported amount is available (the exported amount acts as a limit for the consumer cluster), and the provider cluster can no longer use the amount that has been exported.

- The distribution policies that apply to job slots also apply to other resources.
- If the provider cluster doesn't have the amount that is specified in the export policy, it will export as much as it has.

Tip:

To export hosts of HostExport Type==DLINUX, exporting swap space is mandatory. If you do not specify swap space, the hosts of this host type are filtered because the resource is seen as unavailable

Export memory

To export memory, set MEM in lsb.resources host export policy, and specify the number of MB per host:

- exporting 100 MB on each host:

```
RES_SELECT=type==LINUX
NHOSTS=3
MEM=100
```

Export swap space

To export swap space, set SWP in lsb.resources host export policy, and specify the number of MB per host:

- exporting 100 MB on each host:

```
PER_HOST=HostA HostB HostC
SWP=100
```

Export shared resources

In addition to job slots and some other built-in resources, it is possible to export numeric shared resources. The resource definitions in `lsf.shared` must be the same in both clusters.

Export policies for shared resources are defined in `lsb.resources`, after export policies for hosts. The configuration is different—shared resources are not exported per host.

When you export a shared resource to a consumer cluster, you must already have a host export policy that exports hosts to the same consumer cluster, and the shared resource must be available on one or more of those exported hosts. Otherwise, the export policy does not have any effect.

Configure shared resource export

In `lsb.resources`, configure a resource export policy for each resource as shown:

```
Begin SharedResourceExport
NAME          = AppX
NINSTANCES    = 10
DISTRIBUTION  = ([C1, 30] [C2, 70])
End SharedResourceExport
```

In each policy, you specify one shared numeric resource, the maximum number of these you want to export, and distribution, using the same syntax as a host export policy.

If some quantity of the resource is available, but not the full amount you configured, LSF exports as many instances of the resource as are available to the exported hosts.

Shared lease

Optional.

You can export resources from a cluster and enable shared lease, which allows the provider cluster to share in the use of the exported resources. This type of lease dynamically balances the job slots according to the load in each cluster.

Only job slots will be shared. If you export memory, swap space, and shared resources, they become available to the consumer cluster exclusively.

About shared lease

By default, exported resources are for the exclusive use of the consumer, they cannot be used by the provider. If they are not being used by the consumer, they are wasted.

There is a way to lease job slots to a cluster part-time. With shared lease, both provider and consumer clusters can have the opportunity to take any idle job slots. The benefit of the shared lease is that the provider cluster has a chance to share in the use of its exported resources, so the average resource usage is increased.

Shared lease is not compatible with advance reservation.

If you enable shared leasing, each host can only be exported to a single consumer cluster. Therefore, when shared leasing is enabled, you can export a group of workstations to multiple consumers using `RES_SELECT` syntax, but you cannot share a powerful multiprocessor host among multiple consumer clusters using `PER_HOST` syntax unless the distribution policy specifies just one cluster.

How it works

By default, a lease is exclusive, which means a fixed amount of exported resources is always dedicated exclusively to a consumer cluster. However, if you configure leases to be shared, the job slots exported by each export policy can also become available to the provider cluster.

Reclaimable resources are job slots that are exported with shared leasing enabled. The reclaim process is managed separately for each lease, so the set of job slots exported by one resource export policy to one consumer cluster is managed as a group.

When the provider cluster is started, the job slots are allocated to the provider cluster, except for one that is reserved for the consumer cluster, to allow a lease to be made. Therefore, all but one slot is initially available to the provider cluster, and one slot could be available to the consumer. The lease is made when the consumer schedules a job to run on the single job slot that is initially available to it.

To make job slots available to a different cluster, LSF automatically modifies the lease contract. The lease will go through a temporary “inactive” phase each time. When a lease is updated, the slots controlled by the corresponding export policy are distributed as follows: the slots that are being used to run jobs remain under the control of the cluster that is using them, but the slots that are idle are all made available to just one cluster.

To determine which cluster will reclaim the idle slots each time, LSF considers the number of idle job slots in each cluster:

```
idle_slots_provider = available_slots_provider - used_slots_provider
idle_slots_consumer = available_slots_consumer - used_slots_consumer
```

The action depends on the relative quantity of idle slots in each cluster.

- If the consumer has more idle slots:

```
idle_slots_consumer > idle_slots_provider
```

then the provider reclaims idle slots from the consumer, and all the idle slots go to the provider cluster.

- If the provider has more idle slots:

```
idle_slots_provider > idle_slots_consumer
```

then the reverse happens, and all the idle slots go to the consumer cluster.

- However, if each cluster has an equal number of idle slots:

```
idle_slots_consumer = idle_slots_provider
```

then the lease does not get updated.

LSF evaluates the status at regular intervals, specified by MC_RECLAIM_DELAY in lsb.params.

The calculations are performed separately for each set of reclaimable resources, so if a provider cluster has multiple resource export policies, some leases could be reconfigured in favor of the provider while others get reconfigured in favor of the consumer.

- [Enable shared leasing](#)

Enable shared leasing

Procedure

Set TYPE=shared in the resource export policy (lsb.resources HostExport section).

Remember that each resource export policy using PER_HOST syntax must specify just one cluster in the distribution policy, if the lease is shared.

```
Begin HostExport
PER_HOST=HostA
SLOTS=4
TYPE=shared
DISTRIBUTION=([C1, 1])
End HostExport
```

In this example, `HostA` is exported with shared leasing enabled, so the lease can be reconfigured at regular intervals, allowing LSF to give any idle job slots to the cluster that needs them the most.

Configure reclaim interval

About this task

Optionally set the reclaim interval.

Procedure

Set `MC_RECLAIM_DELAY` in `lsb.params` and specify how often to reconfigure a shared lease, in minutes. The interval is the same for every lease in the cluster.

The default interval is 10 minutes.

Borrow resources

Default queues

When you add new hosts to a single LSF cluster, you might need to update your queues to start sending work to the new hosts. This is often not necessary, because queues with the default configuration can use all hosts in the local cluster.

However, when a MultiCluster provider cluster exports resources to a consumer cluster, the default queue configuration does not allow the consumer cluster to use those resources. You must update your queue configuration to start using the borrowed resources.

Queues that use borrowed hosts

By default, LSF queues only use hosts that belong to the submission cluster. Queues can use borrowed resources when they are configured to use borrowed hosts (and the provider cluster's export policy must be compatible).

Queues for parallel jobs

If your clusters do not have a shared file system, then parallel jobs that require a common file space could fail if they span multiple clusters. One way to prevent this is to submit these jobs to a queue that uses hosts all from one cluster (for example, configure the queue to use local hosts or borrowed hosts, but not both).

Configure a queue to use borrowed resources

To configure a queue to use borrowed resources, edit `lsb.queues` `HOSTS` parameter and specify the hosts you want to borrow from one or more other clusters.

- The keyword `all` does not include borrowed hosts, only hosts that belong to the consumer cluster.
- The keyword `allremote` specifies the group of borrowed hosts belonging to all provider clusters.
- The keyword `others` does not include borrowed hosts, only hosts that belong to the consumer cluster.
- The keyword `none` is not compatible with the resource leasing model.
- You can specify a borrowed host in the format `host_name@cluster_name`. Make sure you configure this correctly, LSF does not validate names of borrowed hosts when you reconfigure the cluster.
- You can specify a host group that includes borrowed resources.
- You can specify all the hosts borrowed from another cluster in the format `all@cluster_name`.

`all` and `allremote`

- Queues configured with the keyword `all` can use all available resources that belong to the consumer cluster. You can specify additional clusters or hosts to use selected borrowed resources also.

```
HOSTS = all all@cluster2 hostB@cluster4
```

- Queues configured with the keyword `allremote` can use all available borrowed resources, from all other clusters. You can also specify additional host names to use selected resources that belong to the consumer cluster.

```
HOSTS = hostB hostC allremote
```

- Queues configured with both keywords can use all available resources whether the hosts are borrowed or belong to the consumer cluster.

```
HOSTS = all allremote
```

Preference

You can specify preference levels for borrowed resources, as well as for local resources. If your clusters do not have a common file system, the extra overhead of file transfer between clusters can affect performance, if a job involves large files. In this case, you should give preference to local hosts.

```
HOSTS = all+1 allremote
```

Parallel jobs and the lease model

About this task

To run parallel jobs (specifying multiple processors with `bsub -n`) across clusters, you must configure the RemoteClusters list in each cluster. By default, this list is not configured. For more information on running parallel jobs, see *Administering IBM Platform LSF*.

Procedure

1. If you do not already have a RemoteClusters list, create the RemoteClusters list and include the names of all remote clusters (the same list as `lsf.shared`). This enables proper communication among all clusters, and enables cross-cluster parallel jobs for all clusters.
2. If you have a RemoteClusters list, and you do not want to run parallel jobs on resources from all provider clusters, configure the `RECV_FROM` column in `lsf.cluster.cluster_name`. Specify “N” to exclude a remote cluster (LSF will not start parallel jobs on resources that belong to the remote cluster). Specify “Y” to enable resource-sharing for parallel jobs. This is the default.

Submitting jobs using JSDL

The Job Submission Description Language (JSDL) provides a convenient format for describing job requirements. You can save a set of job requirements in a JSDL XML file, and then reuse that file as needed to submit jobs to LSF.

For detailed information about JSDL, see the “Job Submission Description Language (JSDL) Specification” at <http://www.gridforum.org/documents/GFD.56.pdf>.

- [Using JSDL files with LSF](#)
LSF complies with the JSDL specification by supporting most valid JSDL elements and POSIX extensions. The LSF extension schema allows you to use LSF features not included in the JSDL standard schema.
- [Collect resource values using `elim.jsdl`](#)
To support the use of JSDL files at job submission, LSF collects the following load indices:

Using JSDL files with LSF

LSF complies with the JSDL specification by supporting most valid JSDL elements and POSIX extensions. The LSF extension schema allows you to use LSF features not included in the JSDL standard schema.

The following sections describe how LSF supports the use of JSDL files for job submission.

Where to find the JSDL schema files

The JSDL schema (jsdl.xsd), the POSIX extension (jsdl-posix.xsd), and the LSF extension (jsdl-lsf.xsd) are located in the LSF_LIBDIR directory.

Supported JSDL and POSIX extension elements

The following table maps the supported JSDL standard and POSIX extension elements to LSF submission options.

Note: For information about how to specify JSDL element types such as range values, see the "Job Submission Description Language (JSDL) Specification" at <http://www.gridforum.org/documents/GFD.56.pdf>.

Table 1. Supported JSDL and POSIX extension elements

Element	bsub Option	Description	Example
Job Structure Elements			
JobDefinition	Not applicable	Root element of the JSDL document. Contains the mandatory child element JobDescription .	<pre><JobDefinition> <JobDescription> ... </JobDescription> </JobDefinition></pre>
JobDescription	-P	High-level container element that holds more specific description elements.	
Job Identity Elements			
JobName	-J	String used to name the job.	<pre><jsdl:JobName>myjob</jsdl:JobName></pre>
JobProject	-P	String that specifies the project to which the job belongs.	<pre><jsdl:JobProject>myproject</jsdl:JobProject></pre>
Application Elements			
Application	Not applicable	High-level container element that holds more specific application definition elements.	
ApplicationName	-app	String that defines the name of an application profile defined in lsb.applications.	<pre><jsdl:Application> <jsdl:ApplicationName>ApplicationX</jsdl:ApplicationName> </jsdl:Application></pre>

Element	bsub Option	Description	Example
ApplicationVersion	-app	String that defines the version of the application defined in lsb.applications.	<pre> <jSDL:Application> <jSDL:ApplicationName> ApplicationX </jSDL:ApplicationName> <jSDL:ApplicationVersion> 5.5 </jSDL:ApplicationVersion> ... </jSDL:Application> </pre>
Resource Elements			
CandidateHosts	-m	Complex type element that specifies the set of named hosts that can be selected to run the job.	<pre> <jSDL:CandidateHosts> <jSDL:HostName>host1 </jSDL:HostName> <jSDL:HostName>host2 </jSDL:HostName> </jSDL:CandidateHosts> </pre>
HostName	-m	Contains a single name of a host or host group. See the previous example (CandidateHosts).	
ExclusiveExecution	-x	Boolean that designates whether the job must have exclusive access to the resources it uses.	<pre> <jSDL:ExclusiveExecution> true </jSDL:ExclusiveExecution> </pre>
OperatingSystemName	-R	A token type that contains the operating system name. LSF uses the external resource osname .	<pre> <jSDL:OperatingSystemName> LINUX </jSDL:OperatingSystemName> </pre>
OperatingSystemVersion	-R	A token type that contains the operating system version. LSF uses the external resource osver .	<pre> <jSDL:OperatingSystemVersion> 5.7 </jSDL:OperatingSystemVersion> </pre>
CPUArchitectureName	-R	Token that specifies the CPU architecture required by the job in the execution environment. LSF uses the external resource cpuarch .	<pre> <jSDL:CPUArchitectureName> sparc </jSDL:CPUArchitectureName> </pre>

Element	bsub Option	Description	Example
IndividualCPUSpeed	-R	Range value that specifies the speed of each CPU required by the job in the execution environment, in Hertz (Hz). LSF uses the external resource cpuspeed .	<pre> <jsd1:IndividualCPUSpeed> <jsd1:LowerBoundedRange> 1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualCPUSpeed> </pre>
IndividualCPUCount	-n	Range value that specifies the number of CPUs for each resource.	<pre> <jsd1:IndividualCPUCount> <jsd1:exact> 2.0</jsd1:exact> </jsd1:IndividualCPUCount> </pre>
IndividualPhysicalMemory	-R	Range value that specifies the amount of physical memory required on each resource, in bytes.	<pre> <jsd1:IndividualPhysicalMemory> <jsd1:LowerBoundedRange> 1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualPhysicalMemory> </pre>
IndividualVirtualMemory	-R	Range value that specifies the amount of virtual memory required for each resource, in bytes.	<pre> <jsd1:IndividualVirtualMemory> <jsd1:LowerBoundedRange> 1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualVirtualMemory> </pre>
IndividualNetworkBandwidth	-R	Range value that specifies the bandwidth requirements of each resource, in bits per second (bps). LSF uses the external resource bandwidth .	<pre> <jsd1:IndividualNetworkBandwidth> <jsd1:LowerBoundedRange> 104857600.0 </jsd1:LowerBoundedRange> </jsd1:IndividualNetworkBandwidth> </pre>
TotalCPUCount	-n	Range value that specifies the total number of CPUs required for the job.	<pre> <jsd1:TotalCPUCount> <jsd1:exact>2.0 </jsd1:exact> </jsd1:TotalCPUCount> </pre>

Element	bsub Option	Description	Example
TotalPhysicalMemory	-R	Range value that specifies the required amount of physical memory for all resources for the job, in bytes.	<pre> <jsd1:TotalPhysicalMemory> <jsd1:LowerBoundedRange> 10737418240.0 </jsd1:LowerBoundedRange> </jsd1:TotalPhysicalMemory> </pre>
TotalVirtualMemory	-R	Range value that specifies the required amount of virtual memory for the job, in bytes.	<pre> <jsd1:TotalVirtualMemory> <jsd1:LowerBoundedRange> 1073741824.0 </jsd1:LowerBoundedRange> </jsd1:TotalVirtualMemory> </pre>
TotalResourceCount	-n	Range value that specifies the total number of resources required by the job.	<pre> <jsd1:Resources>... <jsd1:TotalResourceCount> <jsd1:exact> 5.0</jsd1:exact> </jsd1:TotalResourceCount> </pre>
Data Staging Elements			
FileName	-f	String that specifies the local name of the file or directory on the execution host. For a directory, you must specify the relative path.	<pre> <jsd1:DataStaging> <jsd1:FileName> job1/input/control.txt </jsd1:FileName> ... </jsd1:DataStaging> </pre>
CreationFlag	-f	Specifies whether the file created on the local execution system overwrites or append to an existing file.	<pre> <jsd1:DataStaging> <jsd1:CreationFlag> overwrite </jsd1:CreationFlag> ... </jsd1:DataStaging> </pre>
Source	Not applicable	Contains the location of the file or directory on the remote system. In LSF, the file location is specified by the URI element. The file is staged in before the job is executed. See the example for the Target element.	
URI	-f	Specifies the location used to stage in (Source) or stage out (Target) a file. For use with LSF, the URI must be a file path only, without a protocol.	

Element	bsub Option	Description	Example
Target	Not applicable	Contains the location of the file or directory on the remote system. In LSF, the file location is specified by the URI element. The file is staged out after the job is executed.	<pre> <jsd1:DataStaging> <jsd1:Source> <jsd1:URI> //input/myjobs/control.txt </jsdl:URI> </jsdl:Source> <jsd1:Target> <jsd1:URI> //output/myjobs/control.txt </jsdl:URI> </jsdl:Target> ... </jsdl:DataStaging> </pre>
POSIX Extension Elements			
Executable	Not applicable	String that specifies the command to execute.	<pre> <jsd1-posix:Executable>myscript </jsdl-posix:Executable> </pre>
Argument	Not applicable	Constrained normalized string that specifies an argument for the application or command.	<pre> <jsd1-posix:Argument>10 </jsdl-posix:Argument> </pre>
Input	-i	String that specifies the Standard Input for the command.	<pre> ...<jsd1-posix:Input> input.txt </jsdl-posix:Input> ... </pre>
Output	-o	String that specifies the Standard Output for the command.	<pre> ...<jsd1-posix:Output> output.txt </jsdl-posix:Output> ... </pre>
Error	-e	String that specifies the Standard Error for the command.	<pre> ...<jsd1-posix:Error> error.txt </jsdl-posix:Error> ... </pre>
WorkingDirectory	Not applicable	String that specifies the starting directory required for job execution. If no directory is specified, LSF sets the starting directory on the execution host to the current working directory on the submission host. If the current working directory is not accessible on the execution host, LSF runs the job in the /tmp directory on the execution host.	<pre> ...<jsd1-posix:WorkingDirectory> ./home</jsdl-posix:WorkingDirectory> ... </pre>

Element	bsub Option	Description	Example
Environment	Not applicable	Specifies the name and value of an environment variable defined for the job in the execution environment. LSF maps the JSDL element definitions to the matching LSF environment variables.	<pre><jsd1- posix:Enviro nment name="SHELL" > /bin/bash</j sdl- posix:Enviro nment></pre>
WallTimeLimit	-W	Positive integer that specifies the soft limit on the duration of the application's execution, in seconds.	<pre><jsd1- posix:WallTi meLimit>60 </jsdl- posix:WallTi meLimit></pre>
FileSizeLimit	-F	Positive integer that specifies the maximum size of any file associated with the job, in bytes.	<pre><jsd1- posix:FileSi zeLimit> 1073741824 </jsdl- posix: FileSizeLimi t></pre>
CoreDumpLimit	-C	Positive integer that specifies the maximum size of core dumps a job may create, in bytes.	<pre><jsd1- posix:CoreDu mpLimit>0 </jsdl- posix:CoreDu mpLimit></pre>
DataSegmentLimit	-D	Positive integer that specifies the maximum data segment size, in bytes.	<pre><jsd1- posix:DataSe gmentLimit> 32768 </jsdl- posix:DataSe gmentLimit></pre>
MemoryLimit	-M	Positive integer that specifies the maximum amount of physical memory that the job can use during execution, in bytes.	<pre><jsd1- posix:Memory Limit> 67108864 </jsdl- posix:Memory Limit></pre>
StackSizeLimit	-S	Positive integer that specifies the maximum size of the execution stack for the job, in bytes.	<pre><jsd1- posix:StackS izeLimit> 1048576 </jsdl- posix:StackS izeLimit></pre>
CPUTimeLimit	-c	Positive integer that specifies the number of CPU time seconds a job can consume before a SIGXCPU signal is sent to the job.	<pre><jsd1- posix:CPUTim eLimit>30 </jsdl- posix:CPUTim eLimit></pre>
ProcessCountLimit	-p	Positive integer that specifies the maximum number of processes the job can spawn.	<pre><jsd1- posix:Proces sCountLimit> 8 </jsdl- posix:Proces sCountLimit></pre>

Element	bsub Option	Description	Example
VirtualMemoryLimit	-v	Positive integer that specifies the maximum amount of virtual memory the job can allocate, in bytes.	<pre><jsd1- posix:VirtualMemoryLimit > 134217728 </jsdl- posix:VirtualMemoryLimit ></pre>
ThreadCountLimit	-T	Positive integer that specifies the number of threads the job can create.	<pre><jsd1- posix:ThreadCountLimit>8 </jsdl- posix:VirtualMemoryLimit ></pre>

LSF extension elements

To use all available LSF features, add the elements described in the following table to your JSDL file.

Table 2. LSF extension elements

Element	bsub Option	Description
SchedulerParams	Not applicable	<p>Complex type element that specifies various scheduling parameters (starting with Queue and ending with JobGroup).</p> <pre><jsd1-lsf:SchedulerParams> <jsd1-lsf:ResourceRequirements> "select[swp>15 && hpux] order[ut]" </jsdl-lsf:ResourceRequirements> <jsd1-lsf:Start>12:06:09:55 </jsdl-lsf:Start> <jsd1-lsf:Deadline>8:22:15:50 </jsdl-lsf:Deadline> <jsd1-lsf:ReservationID>"user1#0" </jsdl-lsf:ReservationID> <jsd1-lsf:Dependencies>'done myjob1' </jsdl-lsf:Dependencies> <jsd1-lsf:Rerunnable>true </jsdl-lsf:Rerunnable> <jsd1-lsf:UserPriority>3 </jsdl-lsf:UserPriority> <jsd1-lsf:ServiceClass>platinum </jsdl-lsf:ServiceClass> <jsd1-lsf:Group>sysadmin</jsdl-lsf:Group> <jsd1-lsf:ExternalScheduler>pset </jsdl-lsf:ExternalScheduler> <jsd1-lsf:Hold>true </jsdl-lsf:Hold> <jsd1-lsf:JobGroup>/risk_group/portfolio1 /current </jsdl-lsf:JobGroup> </jsdl-lsf:SchedulerParams></pre>
Queue	-q	String that specifies the queue in which the job runs.
ResourceRequirements	-R	String that specifies one or more resource requirements of the job. Multiple rusage sections are not supported.
Start	-b	String that specifies the earliest time that the job can start.
Deadline	-t	String that specifies the job termination deadline.
ReservationID	-U	String that specifies the reservation ID returned when you use brsvadd to add a reservation.
Dependencies	-w	String that specifies a dependency expression. LSF does not run your job unless the dependency expression evaluates to TRUE .
Rerunnable	-r	Boolean value that specifies whether to reschedule a job on another host if the execution host becomes unavailable while the job is running.

Element	bsub Option	Description
UserPriority	-sp	Positive integer that specifies the user-assigned job priority. This allows users to order their own jobs within a queue.
ServiceClass	-sla	String that specifies the service class where the job is to run.
Group	-G	String that associates the job with the specified group for fair share scheduling.
ExternalScheduler	-ext [sched]	String used to set application-specific external scheduling options for the job.
Hold	-H	Boolean value that tells LSF to hold the job in the PSUSP state when the job is submitted. The job is not scheduled until you tell the system to resume the job.
JobGroup	-g	String that specifies the job group to which the job is submitted.
NotificationParams	Not applicable	Complex type element that tells LSF when and where to send notification email for a job. See the following example: <pre><jsd1-lsf:NotificationParams> <jsd1-lsf:NotifyAtStart> true</jsdl-lsf:NotifyAtStart> <jsd1-lsf:NotifyAtFinish> true</jsdl-lsf:NotifyAtFinish> <jsd1-lsf:NotificationEmail> -u user10</jsdl-lsf:NotificationEmail> </jsdl-lsf:NotificationParams></pre>
NotifyAtStart	-B	Boolean value that tells LSF to notify the user who submitted the job that the job has started.
NotifyAtFinish	-N	Boolean value that tells LSF to notify the user who submitted the job that the job has finished.
NotificationEmail	-u	String that specifies the user who receives notification emails.
RuntimeParams	Not applicable	Complex type element that contains values for LSF runtime parameters. <pre><jsd1-lsf:RuntimeParams> <jsd1-lsf:Interactive>I</jsdl-lsf:Interactive> <jsd1-lsf:Block>true </jsdl-lsf:Block><jsd1-lsf:PreExec>myscript </jsdl-lsf:PreExec><jsd1-lsf:Checkpoint> myjobs/checkpointdir</jsdl-lsf:Checkpoint> <jsd1-lsf:LoginShell>/csh</jsdl-lsf:LoginShell> <jsd1-lsf:SignalJob> 18</jsdl-lsf:SignalJob> <jsd1-lsf:WarningAction> 'URG'</jsdl-lsf:WarningAction> <jsd1-lsf:WarningTime> '2'</jsdl-lsf:WarningTime> <jsd1-lsf:SpoolCommand>true </jsdl-lsf:SpoolCommand> <jsd1-lsf:Checkpoint></jsdl-lsf:RuntimeParams></pre>
Interactive	-I[s p]	String that specifies an interactive job with a defined pseudo-terminal mode.
Block	-K	Boolean value that tells LSF to complete the submitted job before allowing the user to submit another job.
PreExec	-E	String that specifies a pre-exec command to run on the batch job's execution host before actually running the job. For a parallel job, the pre-exec command runs on the first host selected for the parallel job.
Checkpoint	-k	String that makes a job checkpoint-able and specifies the checkpoint directory.
LoginShell	-L	For UNIX jobs, string that tells LSF to initialize the execution environment using the specified login shell.
SignalJob	-s	String that specifies the signal to send when a queue-level run window closes. Use this to override the default signal that suspends jobs running in the queue.
WarningAction	-wa	String that specifies the job action prior to the job control action. Requires that you also specify the job action warning time.
WarningTime	-wt	Positive integer that specifies the amount of time prior to a job control action that the job warning action should occur.
SpoolCommand	-is	Boolean value that spools a job command file to the directory specified by JOB_SPOOL_DIR, and uses the spooled file as the command file for the job.

-
- [Submit a job using a JSDL file](#)
 - [Submit a job using a JSDL file](#)

Unsupported JSDL and POSIX extension elements

The current version of LSF does not support the following elements:

Job structure elements

- `Description`

Job identity elements

- `JobAnnotation`

Resource elements

- `FileSystem`
- `MountPoint`
- `MountSource`
- `DiskSpace`
- `FileSystemType`
- `OperatingSystemType`
- `IndividualCPUTime`
- `IndividualDiskSpace`
- `TotalCPUTime`
- `TotalDiskSpace`

Data staging elements

- `FileSystemName`
- `DeleteOnTermination`

POSIX extension elements

- `LockedMemoryLimit`
- `OpenDescriptorsLimit`
- `PipeSizeLimit`
- `UserName`
- `GroupName`

Submit a job using a JSDL file

Procedure

To submit a job using a JSDL file, use one of the following **bsub** command options:

- To submit a job that uses elements included in the LSF extension, use the `-jsdl` option.
- To submit a job that uses only standard JSDL elements and POSIX extensions, use the `-jsdl_strict` option. Error messages indicate invalid elements, including:
 - Elements that are not part of the JSDL specification
 - Valid JSDL elements that are not supported in this version of LSF
 - Elements that are not part of the JSDL standard and POSIX extension schema

Results

If you specify duplicate or conflicting job submission parameters, LSF resolves the conflict by applying the following rules:

- The parameters specified in the command line override all other parameters.
- A job script or user input for an interactive job overrides parameters specified in the JSDL file.

Collect resource values using elim.jsdl

To support the use of JSDL files at job submission, LSF collects the following load indices:

Attribute name	Attribute type	Resource name
OperatingSystemName	string	osname
OperatingSystemVersion	string	osver
CPUArchitectureName	string	cpuarch
IndividualCPUSpeed	int64	cpuspeed
IndividualNetworkBandwidth	int64	bandwidth(This is the maximum bandwidth).

The file `elim.jsdl` is automatically configured to collect these resources, but you must enable its use by modifying the files `lsf.cluster.cluster_name` and `lsf.shared`.

- [Enabling JSDL resource collection](#)
- [Enabling JSDL resource collection](#)

Enabling JSDL resource collection

Procedure

1. In the file `lsf.cluster.cluster_name`, locate the `ResourcesMap` section.
2. In the file `lsf.shared`, locate the `Resource` section.
3. Uncomment the lines for the following resources in both files:
 - `osname`
 - `osver`
 - `cpuarch`
 - `cpuspeed`
 - `bandwidth`
4. To propagate the changes through the LSF system, run the following commands.
 - a. `lsadmin reconfig`
 - b. `badmin mbdrestart`

You have now configured LSF to use the `elim.jsdl` file to collect JSDL resources.

Submitting jobs using JSDL

The Job Submission Description Language (JSDL) provides a convenient format for describing job requirements. You can save a set of job requirements in a JSDL XML file, and then reuse that file as needed to submit jobs to LSF.

For detailed information about JSDL, see the "Job Submission Description Language (JSDL) Specification" at <http://www.gridforum.org/documents/GFD.56.pdf>.

- [Using JSDL files with LSF](#)
LSF complies with the JSDL specification by supporting most valid JSDL elements and POSIX extensions. The LSF extension schema allows you to use LSF features not included in the JSDL standard schema.
- [Collect resource values using elim.jsdl](#)
To support the use of JSDL files at job submission, LSF collects the following load indices:

Using JSDL files with LSF

LSF complies with the JSDL specification by supporting most valid JSDL elements and POSIX extensions. The LSF extension schema allows you to use LSF features not included in the JSDL standard schema.

The following sections describe how LSF supports the use of JSDL files for job submission.

Where to find the JSDL schema files

The JSDL schema (jsdl.xsd), the POSIX extension (jsdl-posix.xsd), and the LSF extension (jsdl-lsf.xsd) are located in the LSF_LIBDIR directory.

Supported JSDL and POSIX extension elements

The following table maps the supported JSDL standard and POSIX extension elements to LSF submission options.

Note: For information about how to specify JSDL element types such as range values, see the "Job Submission Description Language (JSDL) Specification" at <http://www.gridforum.org/documents/GFD.56.pdf>.

Table 1. Supported JSDL and POSIX extension elements

Element	bsub Option	Description	Example
Job Structure Elements			
JobDefinition	Not applicable	Root element of the JSDL document. Contains the mandatory child element JobDescription .	<code><JobDefinition> <JobDescription> ... </JobDescription> </JobDefinition></code>
JobDescription	-P	High-level container element that holds more specific description elements.	
Job Identity Elements			
JobName	-J	String used to name the job.	<code><jsdl:JobName>myjob</jsdl:JobName></code>
JobProject	-P	String that specifies the project to which the job belongs.	<code><jsdl:JobProject> myproject </jsdl:JobProject></code>
Application Elements			
Application	Not applicable	High-level container element that holds more specific application definition elements.	
ApplicationName	-app	String that defines the name of an application profile defined in lsb.applications.	<code><jsdl:Application> <jsdl:ApplicationName>Ap plicationX </jsdl:ApplicationName> </jsdl:Application></code>

Element	bsub Option	Description	Example
ApplicationVersion	-app	String that defines the version of the application defined in lsb.applications.	<pre> <jsd1:Application> <jsd1:ApplicationName> ApplicationX </jsdl:ApplicationName> <jsd1:ApplicationVersion>5.5 </jsdl:ApplicationVersion> ... </jsdl:Application> </pre>
Resource Elements			
CandidateHosts	-m	Complex type element that specifies the set of named hosts that can be selected to run the job.	<pre> <jsd1:CandidateHosts> <jsd1:HostName>host1 </jsdl:HostName> <jsd1:HostName>host2 </jsdl:HostName> </jsdl:CandidateHosts> </pre>
HostName	-m	Contains a single name of a host or host group. See the previous example (CandidateHosts).	
ExclusiveExecution	-x	Boolean that designates whether the job must have exclusive access to the resources it uses.	<pre> <jsd1:ExclusiveExecution>true </jsdl:ExclusiveExecution> </pre>
OperatingSystemName	-R	A token type that contains the operating system name. LSF uses the external resource osname .	<pre> <jsd1:OperatingSystemName>LINUX </jsdl:OperatingSystemName> </pre>
OperatingSystemVersion	-R	A token type that contains the operating system version. LSF uses the external resource osver .	<pre> <jsd1:OperatingSystemVersion>5.7 </jsdl:OperatingSystemVersion> </pre>
CPUArchitectureName	-R	Token that specifies the CPU architecture required by the job in the execution environment. LSF uses the external resource cpuarch .	<pre> <jsd1:CPUArchitectureName>sparc </jsdl:CPUArchitectureName> </pre>

Element	bsub Option	Description	Example
IndividualCPUSpeed	-R	Range value that specifies the speed of each CPU required by the job in the execution environment, in Hertz (Hz). LSF uses the external resource cpuspeed .	<pre> <jsd1:IndividualCPUSpeed> <jsd1:LowerBoundedRange> 1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualCPUSpeed> </pre>
IndividualCPUCount	-n	Range value that specifies the number of CPUs for each resource.	<pre> <jsd1:IndividualCPUCount> <jsd1:exact> 2.0</jsd1:exact> </jsd1:IndividualCPUCount> </pre>
IndividualPhysicalMemory	-R	Range value that specifies the amount of physical memory required on each resource, in bytes.	<pre> <jsd1:IndividualPhysicalMemory> <jsd1:LowerBoundedRange> 1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualPhysicalMemory> </pre>
IndividualVirtualMemory	-R	Range value that specifies the amount of virtual memory required for each resource, in bytes.	<pre> <jsd1:IndividualVirtualMemory> <jsd1:LowerBoundedRange> 1073741824.0 </jsd1:LowerBoundedRange> </jsd1:IndividualVirtualMemory> </pre>
IndividualNetworkBandwidth	-R	Range value that specifies the bandwidth requirements of each resource, in bits per second (bps). LSF uses the external resource bandwidth .	<pre> <jsd1:IndividualNetworkBandwidth> <jsd1:LowerBoundedRange> 104857600.0 </jsd1:LowerBoundedRange> </jsd1:IndividualNetworkBandwidth> </pre>
TotalCPUCount	-n	Range value that specifies the total number of CPUs required for the job.	<pre> <jsd1:TotalCPUCount> <jsd1:exact>2.0 </jsd1:exact> </jsd1:TotalCPUCount> </pre>

Element	bsub Option	Description	Example
TotalPhysicalMemory	-R	Range value that specifies the required amount of physical memory for all resources for the job, in bytes.	<pre> <jsd1:TotalPhysicalMemory> <jsd1:LowerBoundedRange> 10737418240.0 </jsd1:LowerBoundedRange> </jsd1:TotalPhysicalMemory> </pre>
TotalVirtualMemory	-R	Range value that specifies the required amount of virtual memory for the job, in bytes.	<pre> <jsd1:TotalVirtualMemory> <jsd1:LowerBoundedRange> 1073741824.0 </jsd1:LowerBoundedRange> </jsd1:TotalVirtualMemory> </pre>
TotalResourceCount	-n	Range value that specifies the total number of resources required by the job.	<pre> <jsd1:Resources>... <jsd1:TotalResourceCount> <jsd1:exact> 5.0 </jsd1:exact> </jsd1:TotalResourceCount> </pre>
Data Staging Elements			
FileName	-f	String that specifies the local name of the file or directory on the execution host. For a directory, you must specify the relative path.	<pre> <jsd1:DataStaging> <jsd1:FileName> job1/input/control.txt </jsd1:FileName> ... </jsd1:DataStaging> </pre>
CreationFlag	-f	Specifies whether the file created on the local execution system overwrites or append to an existing file.	<pre> <jsd1:DataStaging> <jsd1:CreationFlag> overwrite </jsd1:CreationFlag> ... </jsd1:DataStaging> </pre>
Source	Not applicable	Contains the location of the file or directory on the remote system. In LSF, the file location is specified by the URI element. The file is staged in before the job is executed. See the example for the Target element.	
URI	-f	Specifies the location used to stage in (Source) or stage out (Target) a file. For use with LSF, the URI must be a file path only, without a protocol.	

Element	bsub Option	Description	Example
Target	Not applicable	Contains the location of the file or directory on the remote system. In LSF, the file location is specified by the URI element. The file is staged out after the job is executed.	<pre> <jsd1:DataStaging> <jsd1:Source> <jsd1:URI> //input/myjobs/ control.txt </jsdl:URI> </jsdl:Source> <jsd1:Target> <jsd1:URI> //output/myjobs/control.txt </jsdl:URI> </jsdl:Target> ... </jsdl:DataStaging> </pre>
POSIX Extension Elements			
Executable	Not applicable	String that specifies the command to execute.	<pre> <jsd1-posix:Executable>myscript </jsdl-posix:Executable> </pre>
Argument	Not applicable	Constrained normalized string that specifies an argument for the application or command.	<pre> <jsd1-posix:Argument>10 </jsdl-posix:Argument> </pre>
Input	-i	String that specifies the Standard Input for the command.	<pre> ...<jsd1-posix:Input> input.txt </jsdl-posix:Input> ... </pre>
Output	-o	String that specifies the Standard Output for the command.	<pre> ...<jsd1-posix:Output> output.txt </jsdl-posix:Output> ... </pre>
Error	-e	String that specifies the Standard Error for the command.	<pre> ...<jsd1-posix:Error> error.txt </jsdl-posix:Error> ... </pre>
WorkingDirectory	Not applicable	String that specifies the starting directory required for job execution. If no directory is specified, LSF sets the starting directory on the execution host to the current working directory on the submission host. If the current working directory is not accessible on the execution host, LSF runs the job in the /tmp directory on the execution host.	<pre> ...<jsd1-posix:WorkingDirectory> ./home</jsdl-posix:WorkingDirectory> ... </pre>

Element	bsub Option	Description	Example
Environment	Not applicable	Specifies the name and value of an environment variable defined for the job in the execution environment. LSF maps the JSDL element definitions to the matching LSF environment variables.	<pre><jsd1- posix:Enviro nment name="SHELL" > /bin/bash</j sdl- posix:Enviro nment></pre>
WallTimeLimit	-W	Positive integer that specifies the soft limit on the duration of the application's execution, in seconds.	<pre><jsd1- posix:WallTi meLimit>60 </jsdl- posix:WallTi meLimit></pre>
FileSizeLimit	-F	Positive integer that specifies the maximum size of any file associated with the job, in bytes.	<pre><jsd1- posix:FileSi zeLimit> 1073741824 </jsdl- posix: FileSizeLimi t></pre>
CoreDumpLimit	-C	Positive integer that specifies the maximum size of core dumps a job may create, in bytes.	<pre><jsd1- posix:CoreDu mpLimit>0 </jsdl- posix:CoreDu mpLimit></pre>
DataSegmentLimit	-D	Positive integer that specifies the maximum data segment size, in bytes.	<pre><jsd1- posix:DataSe gmentLimit> 32768 </jsdl- posix:DataSe gmentLimit></pre>
MemoryLimit	-M	Positive integer that specifies the maximum amount of physical memory that the job can use during execution, in bytes.	<pre><jsd1- posix:Memory Limit> 67108864 </jsdl- posix:Memory Limit></pre>
StackSizeLimit	-S	Positive integer that specifies the maximum size of the execution stack for the job, in bytes.	<pre><jsd1- posix:StackS izeLimit> 1048576 </jsdl- posix:StackS izeLimit></pre>
CPUTimeLimit	-c	Positive integer that specifies the number of CPU time seconds a job can consume before a SIGXCPU signal is sent to the job.	<pre><jsd1- posix:CPUTim eLimit>30 </jsdl- posix:CPUTim eLimit></pre>
ProcessCountLimit	-p	Positive integer that specifies the maximum number of processes the job can spawn.	<pre><jsd1- posix:Proces sCountLimit> 8 </jsdl- posix:Proces sCountLimit></pre>

Element	bsub Option	Description	Example
VirtualMemoryLimit	-v	Positive integer that specifies the maximum amount of virtual memory the job can allocate, in bytes.	<pre><jsd1- posix:VirtualMemoryLimit > 134217728 </jsdl- posix:VirtualMemoryLimit ></pre>
ThreadCountLimit	-T	Positive integer that specifies the number of threads the job can create.	<pre><jsd1- posix:ThreadCountLimit>8 </jsdl- posix:VirtualMemoryLimit ></pre>

LSF extension elements

To use all available LSF features, add the elements described in the following table to your JSDL file.

Table 2. LSF extension elements

Element	bsub Option	Description
SchedulerParams	Not applicable	<p>Complex type element that specifies various scheduling parameters (starting with Queue and ending with JobGroup).</p> <pre><jsd1-lsf:SchedulerParams> <jsd1-lsf:ResourceRequirements> "select[swp>15 && hpux] order[ut]" </jsdl-lsf:ResourceRequirements> <jsd1-lsf:Start>12:06:09:55 </jsdl-lsf:Start> <jsd1-lsf:Deadline>8:22:15:50 </jsdl-lsf:Deadline> <jsd1-lsf:ReservationID>"user1#0" </jsdl-lsf:ReservationID> <jsd1-lsf:Dependencies>'done myjob1' </jsdl-lsf:Dependencies> <jsd1-lsf:Rerunnable>true </jsdl-lsf:Rerunnable> <jsd1-lsf:UserPriority>3 </jsdl-lsf:UserPriority> <jsd1-lsf:ServiceClass>platinum </jsdl-lsf:ServiceClass> <jsd1-lsf:Group>sysadmin</jsdl-lsf:Group> <jsd1-lsf:ExternalScheduler>pset </jsdl-lsf:ExternalScheduler> <jsd1-lsf:Hold>true </jsdl-lsf:Hold> <jsd1-lsf:JobGroup>/risk_group/portfolio1 /current </jsdl-lsf:JobGroup> </jsdl-lsf:SchedulerParams></pre>
Queue	-q	String that specifies the queue in which the job runs.
ResourceRequirements	-R	String that specifies one or more resource requirements of the job. Multiple rusage sections are not supported.
Start	-b	String that specifies the earliest time that the job can start.
Deadline	-t	String that specifies the job termination deadline.
ReservationID	-U	String that specifies the reservation ID returned when you use brsvadd to add a reservation.
Dependencies	-w	String that specifies a dependency expression. LSF does not run your job unless the dependency expression evaluates to TRUE .
Rerunnable	-r	Boolean value that specifies whether to reschedule a job on another host if the execution host becomes unavailable while the job is running.

Element	bsub Option	Description
UserPriority	-sp	Positive integer that specifies the user-assigned job priority. This allows users to order their own jobs within a queue.
ServiceClass	-sla	String that specifies the service class where the job is to run.
Group	-G	String that associates the job with the specified group for fair share scheduling.
ExternalScheduler	-ext [sched]	String used to set application-specific external scheduling options for the job.
Hold	-H	Boolean value that tells LSF to hold the job in the PSUSP state when the job is submitted. The job is not scheduled until you tell the system to resume the job.
JobGroup	-g	String that specifies the job group to which the job is submitted.
NotificationParams	Not applicable	Complex type element that tells LSF when and where to send notification email for a job. See the following example: <pre><jsd1-lsf:NotificationParams> <jsd1-lsf:NotifyAtStart> true</jsdl-lsf:NotifyAtStart> <jsd1-lsf:NotifyAtFinish> true</jsdl-lsf:NotifyAtFinish> <jsd1-lsf:NotificationEmail> -u user10</jsdl-lsf:NotificationEmail> </jsdl-lsf:NotificationParams></pre>
NotifyAtStart	-B	Boolean value that tells LSF to notify the user who submitted the job that the job has started.
NotifyAtFinish	-N	Boolean value that tells LSF to notify the user who submitted the job that the job has finished.
NotificationEmail	-u	String that specifies the user who receives notification emails.
RuntimeParams	Not applicable	Complex type element that contains values for LSF runtime parameters. <pre><jsd1-lsf:RuntimeParams> <jsd1-lsf:Interactive>I</jsdl-lsf:Interactive> <jsd1-lsf:Block>true </jsdl-lsf:Block><jsd1-lsf:PreExec>myscript </jsdl-lsf:PreExec><jsd1-lsf:Checkpoint> myjobs/checkpointdir</jsdl-lsf:Checkpoint> <jsd1-lsf:LoginShell>/csh</jsdl-lsf:LoginShell> <jsd1-lsf:SignalJob> 18</jsdl-lsf:SignalJob> <jsd1-lsf:WarningAction> 'URG'</jsdl-lsf:WarningAction> <jsd1-lsf:WarningTime> '2'</jsdl-lsf:WarningTime> <jsd1-lsf:SpoolCommand>true </jsdl-lsf:SpoolCommand> </jsdl-lsf:Checkpoint></jsdl-lsf:RuntimeParams></pre>
Interactive	-I[s p]	String that specifies an interactive job with a defined pseudo-terminal mode.
Block	-K	Boolean value that tells LSF to complete the submitted job before allowing the user to submit another job.
PreExec	-E	String that specifies a pre-exec command to run on the batch job's execution host before actually running the job. For a parallel job, the pre-exec command runs on the first host selected for the parallel job.
Checkpoint	-k	String that makes a job checkpoint-able and specifies the checkpoint directory.
LoginShell	-L	For UNIX jobs, string that tells LSF to initialize the execution environment using the specified login shell.
SignalJob	-s	String that specifies the signal to send when a queue-level run window closes. Use this to override the default signal that suspends jobs running in the queue.
WarningAction	-wa	String that specifies the job action prior to the job control action. Requires that you also specify the job action warning time.
WarningTime	-wt	Positive integer that specifies the amount of time prior to a job control action that the job warning action should occur.
SpoolCommand	-is	Boolean value that spools a job command file to the directory specified by JOB_SPOOL_DIR, and uses the spooled file as the command file for the job.

-
- [Submit a job using a JSDL file](#)
 - [Submit a job using a JSDL file](#)

Unsupported JSDL and POSIX extension elements

The current version of LSF does not support the following elements:

Job structure elements

- `Description`

Job identity elements

- `JobAnnotation`

Resource elements

- `FileSystem`
- `MountPoint`
- `MountSource`
- `DiskSpace`
- `FileSystemType`
- `OperatingSystemType`
- `IndividualCPUTime`
- `IndividualDiskSpace`
- `TotalCPUTime`
- `TotalDiskSpace`

Data staging elements

- `FileSystemName`
- `DeleteOnTermination`

POSIX extension elements

- `LockedMemoryLimit`
- `OpenDescriptorsLimit`
- `PipeSizeLimit`
- `UserName`
- `GroupName`

Submit a job using a JSDL file

Procedure

To submit a job using a JSDL file, use one of the following **bsub** command options:

- To submit a job that uses elements included in the LSF extension, use the `-jsdl` option.
- To submit a job that uses only standard JSDL elements and POSIX extensions, use the `-jsdl_strict` option. Error messages indicate invalid elements, including:
 - Elements that are not part of the JSDL specification
 - Valid JSDL elements that are not supported in this version of LSF
 - Elements that are not part of the JSDL standard and POSIX extension schema

Results

If you specify duplicate or conflicting job submission parameters, LSF resolves the conflict by applying the following rules:

- The parameters specified in the command line override all other parameters.
- A job script or user input for an interactive job overrides parameters specified in the JSDL file.

Collect resource values using elim.jsdl

To support the use of JSDL files at job submission, LSF collects the following load indices:

Attribute name	Attribute type	Resource name
OperatingSystemName	string	osname
OperatingSystemVersion	string	osver
CPUArchitectureName	string	cpuarch
IndividualCPUSpeed	int64	cpuspeed
IndividualNetworkBandwidth	int64	bandwidth(This is the maximum bandwidth).

The file `elim.jsdl` is automatically configured to collect these resources, but you must enable its use by modifying the files `lsf.cluster.cluster_name` and `lsf.shared`.

- [Enabling JSDL resource collection](#)
- [Enabling JSDL resource collection](#)

Enabling JSDL resource collection

Procedure

1. In the file `lsf.cluster.cluster_name`, locate the `ResourcesMap` section.
2. In the file `lsf.shared`, locate the `Resource` section.
3. Uncomment the lines for the following resources in both files:
 - `osname`
 - `osver`
 - `cpuarch`
 - `cpuspeed`
 - `bandwidth`
4. To propagate the changes through the LSF system, run the following commands.
 - a. `lsadmin reconfig`
 - b. `badmin mbdrestart`

You have now configured LSF to use the `elim.jsdl` file to collect JSDL resources.

IBM Spectrum LSF Session Scheduler

Install, manage, and use IBM® Spectrum LSF Session Scheduler. Run large collections of short duration tasks within the allocation of a single LSF job by using a job-level task scheduler that allocates resources for the job once, and reuses the allocated resources for each task. IBM Spectrum LSF Session Scheduler is ideal for running short jobs, whether they are a list of tasks, or job arrays with parametric execution.

- [About IBM Spectrum LSF Session Scheduler](#)

LSF Session Scheduler enables users to run large collections of short duration tasks within the allocation of a single LSF job using a job-level task scheduler that allocates resources for the job once, and reuses the allocated resources for each task. LSF Session Scheduler implements a hierarchical, personal scheduling paradigm that provides very low-latency execution. With very low latency per job, LSF Session Scheduler is ideal for executing very short jobs, whether they are a list of tasks, or job arrays with parametric execution.

- [Installing IBM Spectrum LSF Session Scheduler](#)
LSF Session Scheduler must be deployed on an LSF server host.
- [How LSF Session Scheduler runs tasks](#)
- [Running and monitoring LSF Session Scheduler jobs](#)
- [Troubleshooting](#)
Use any of the following methods to troubleshoot your LSF Session Scheduler jobs.

About IBM Spectrum LSF Session Scheduler

LSF Session Scheduler enables users to run large collections of short duration tasks within the allocation of a single LSF job using a job-level task scheduler that allocates resources for the job once, and reuses the allocated resources for each task. LSF Session Scheduler implements a hierarchical, personal scheduling paradigm that provides very low-latency execution. With very low latency per job, LSF Session Scheduler is ideal for executing very short jobs, whether they are a list of tasks, or job arrays with parametric execution.

While traditional LSF job submission, scheduling, and dispatch methods such as job arrays or job chunking are well suited to a mix of long and short running jobs, or jobs with dependencies on each other, LSF Session Scheduler is ideal for large volumes of independent jobs with short run times.

As clusters grow and the volume of workload increases, the need to delegate scheduling decisions increases. LSF Session Scheduler improves throughput and performance of the LSF scheduler by enabling multiple tasks to be submitted as a single LSF job.

Each LSF Session Scheduler is dynamically scheduled in a similar manner to a parallel job. Each instance of the **ssched** command then manages its own workload within its assigned allocation. Work is submitted as a task array or a task definition file.

LSF Session Scheduler satisfies the following goals for running a large volume of short jobs:

- Minimize the latency when scheduling short jobs
- Improve overall cluster utilization and system performance
- Allocate resources according to LSF policies
- Support existing LSF pre-execution, post-execution programs, job starters, resources limits, etc.
- Handle thousands of users and more than 50000 short jobs per user

System requirements

Supported operating systems

LSF Session Scheduler is delivered in the following distribution:

- ls10.1.0.12_ssched_lnx26-libc23-x64.tar.Z

Required libraries

Note: These libraries may not be installed by default by all Linux distributions.

On Linux 2.6 (x86_64), the following external libraries are required:

- libstdc++.so.6
- libpthread-2.3.4.so or later

Compatible Linux distributions

Certified compatible distributions include:

- Red Hat Enterprise Linux AS 3 or later
- SUSE Linux Enterprise Server 10

IBM Spectrum LSF

LSF Session Scheduler is included with IBM Spectrum LSF Advanced Edition and is available as an add-on for other editions of IBM Spectrum LSF:

- If you are using IBM Spectrum LSF Advanced Edition, download the LSF Session Scheduler distribution package from the same download page as the IBM Spectrum LSF Advanced Edition distribution packages.
- If you are using other editions of IBM Spectrum LSF, purchase LSF Session Scheduler as a separate add-on, then download the distribution package from the LSF Session Scheduler download page.

LSF Session Scheduler terminology

Job

A traditional LSF job that is individually scheduled and dispatched to **sbatchd** by **mbatchd** and **mbschd**

Task

Similar to a job, a unit of workload that describes an executable and its environment that runs on an execution node. Tasks are managed and dispatched by the LSF Session Scheduler.

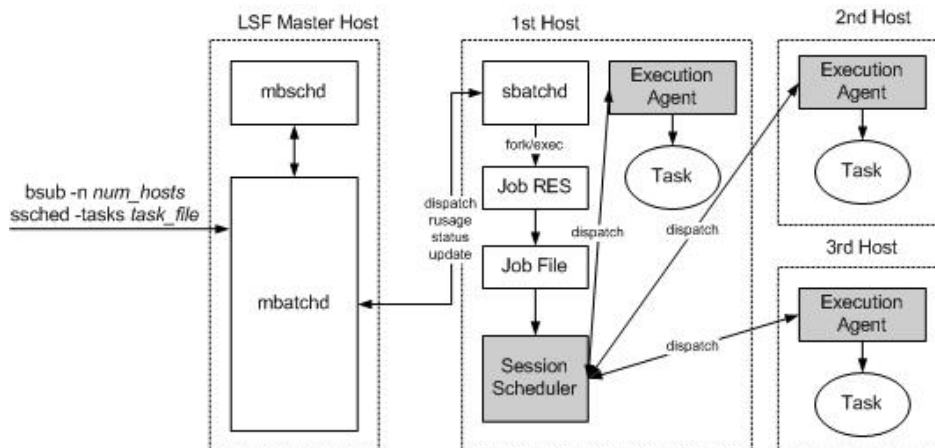
Job Session

An LSF job that is individually scheduled by **mbatchd**, but is not dispatched as an LSF job. Instead, a running LSF Session Scheduler job session represents an allocation of nodes for running large collections of tasks

Scheduler

The component that accepts and dispatches tasks within the nodes allocated for a job session.

Architecture



LSF Session Scheduler jobs are submitted, scheduled, and dispatched like normal LSF jobs.

When the LSF Session Scheduler begins running, it starts one LSF Session Scheduler execution agent on each host in its allocation.

The LSF Session Scheduler then reads in the task definition file, which contains a list of tasks to run. Tasks are sent to an execution agent and run. When a task finishes, the next task in the list is dispatched to the available host. This continues until all tasks have been run.

Tasks submitted through LSF Session Scheduler bypass the LSF **mbatchd** and **mbschd**. The LSF **mbatchd** is unaware of individual tasks.

Components

LSF Session Scheduler comprises the following components.

LSF Session Scheduler command (ssched)

The **ssched** command accepts and dispatches tasks within the nodes allocated for a job session. It reads the task definition file and sends tasks to the execution agents. **ssched** also logs errors, performs task accounting, and requeues tasks as necessary.

sservice and sschild

These components are the execution agents. They run on each remote host in the allocation. They set up the task execution environment, run the tasks, and enable task monitoring and resource usage collection.

Performance

LSF Session Scheduler has been tested to support up to 50,000 tasks. Based on performance tests, the best maximum allocation size (specified by **bsub -n**) depends on the average runtime of the tasks. Here are some typical results:

Average Runtime (seconds)	Recommended maximum allocation size (slots)
0	12
5	64
15	256
30	512

Installing IBM Spectrum LSF Session Scheduler

LSF Session Scheduler must be deployed on an LSF server host.

Before you begin

Before you install LSF Session Scheduler, you must know the following information:

- The following installation parameters (in the `install.config` file) for the LSF cluster: `LSF_TOP`, `LSF_ADMINS`, `LSF_CLUSTER_NAME`, `LSF_MASTER_LIST`, and `LSF_ENTITLEMENT_FILE`
- OS account name and group of the primary LSF administrator

In addition, the **ed** application must be installed. If **ed** is not installed, use the **yum install ed** command to install it.

Procedure

1. Log in to the LSF Session Scheduler installation server as root.
If you are not root, see [If you install as a non-root user](#) for more details.
2. Get the LSF Session Scheduler installer script package and extract it.

```
zcat lsf10.1_ssched_install.tar.Z | tar xvf -
```
3. Navigate to the extracted `lsf10.1.0_ssched_install/` directory.
4. Get the LSF Session Scheduler distribution packages for all host types that you need and put them in the `lsf10.1.0_ssched_install/` directory without extracting the packages.
For example, for Linux kernel 2.6 glibc version 2.3, the distribution package is `lsf10.1.0_ssched_lnx26-x64.tar.Z`.
5. Edit the `install.config` file and specify the installation variables that you want.
Uncomment the options that you want in the `install.config` template file and replace the example values with your own settings.

Note: The sample values in the `install.config` template files are examples only. They are not default installation values.
 - `LSF_TOP`, `LSF_ADMINS`, `LSF_CLUSTER_NAME`, `LSF_MASTER_LIST`, and `LSF_ENTITLEMENT_FILE`: Use the same parameter values as the LSF cluster on which you plan to install LSF Session Scheduler.
 - `LSF_TARDIR`: Set this parameter to the location in which you put the LSF Session Scheduler distribution files.
6. Run the **ssinstall** script command while specifying the `install.config` file to install the cluster.

```
ssinstall -f install.config
```

How LSF Session Scheduler runs tasks

Once a LSF Session Scheduler session job has been dispatched and starts running, LSF Session Scheduler parses the task definition file specified on the **ssched** command. Each line of the task definition file is one task. Tasks run on the hosts in the allocation in any order. Dependencies between tasks are not supported.

LSF Session Scheduler status is posted to the LSF Session Scheduler session job through the LSF **bpost** command. Use **bread** or **bjobs -l** to view LSF Session Scheduler status. The status includes the current number of pending, running and completed tasks. LSF administrators can configure how often the status is updated.

When all tasks are completed, the LSF Session Scheduler exits normally.

ssched runs under the submission user account. Any processes it creates, either locally or remotely, also run under the submission user account. LSF Session Scheduler does not require any privileges beyond those normally granted a user.

LSF Session Scheduler job sessions

The LSF Session Scheduler session job is compatible with all currently supported LSF job submission and execution parameters, including pre-execution, post-execution, job-starters, I/O redirection, queue and application profile configuration.

Run limits are interpreted and enforced as normal LSF parallel jobs. Application-level checkpointing is also supported. Job chunking is not relevant to LSF Session Scheduler jobs since a single LSF Session Scheduler session is generally long running and should not be chunked.

If the LSF Session Scheduler session is killed (**bkill**) or re-queued (**brequeue**), the LSF Session Scheduler kills all running tasks, execution agents, and any other processes it has started, both local and remote. The session scheduler also cleans up any temporary files created and then exits. If the LSF Session Scheduler is then re-queued and restarted, all tasks are rerun.

If the LSF Session Scheduler session is suspended (**bstop**), the LSF Session Scheduler and all local and remote components will be stopped until the session is resumed (**brresume**).

LSF Session Scheduler tasks

ssched and **sservice** and **sschild** execution agents ensure that the user submission environment variables are set correctly for each task. In order to minimize the load on the LSF, **mbatchd** does not have any knowledge of individual tasks.

Task definition file format

The task definition file is an ASCII file. Each line represents one task, or an array of tasks. Each line has the following format.

```
[task_options] command [arguments]
```

Session and task accounting

Jobs corresponding to the LSF Session Scheduler session have one record in `lsb.acct`. This record represents the aggregate resource usage of all tasks in the allocation.

If task accounting is enabled with `SSCHED_ACCT_DIR` in `lsb.params`, Session Scheduler creates task accounting files for each LSF Session Scheduler session job and appends an accounting record to the end of the file. This record follows a similar format to the LSF accounting file `lsb.acct` format, but with additional fields/

The accounting file is named `jobID.sched.acct`. If no directory is specified, accounting records are not written.

The LSF Session Scheduler accounting directory must be accessible and writable from all hosts in the cluster. Each LSF Session Scheduler session (each **ssched** instance) creates one accounting file. Each file contains one accounting entry for each task. Each completed task index has one line in the file. Each line records the resource usage of one task.

Task accounting file format

Task accounting records have a similar format as the `lsb.acct JOB_FINISH` event record.

Field	Description
Event type (%s)	<code>TASK_FINISH</code>

Field	Description
Version Number (%s)	10.1.0
Event Time (%d)	Time the event was logged (in seconds since the epoch)
jobId (%d)	ID for the job
userId (%d)	UNIX user ID of the submitter
options (%d)	Always 0
numProcessors (%d)	Always 1
submitTime (%d)	Task enqueue time
beginTime (%d)	Always 0
termTime (%d)	Always 0
startTime (%d)	Task start time
userName (%s)	User name of the submitter
queue (%s)	Always empty
resReq (%s)	Always empty
dependCond (%s)	Always empty
preExecCmd (%s)	Task pre-execution command
fromHost (%s)	Submission host name
cwd (%s)	Execution host current working directory (up to 4094 characters)
inFile (%s)	Task input file name (up to 4094 characters)
outFile (%s)	Task output file name (up to 4094 characters)
errFile (%s)	Task error output file name (up to 4094 characters)
jobFile (%s)	Task script file name
numAskedHosts (%d)	Always 0
askedHosts (%s)	Name of the asked execution host for the task. When numAskedHosts is 0, this value can be ignored.
numExHosts (%d)	Always 1
execHosts (%s)	Name of task execution host
jStatus (%d)	64 indicates task completed normally. 32 indicates task exited abnormally
hostFactor (%f)	CPU factor of the task execution host
jobName (%s)	Always empty
command (%s)	Complete batch task command specified by the user (up to 4094 characters)
lsfRusage (%f)	All rusage fields contain resource usage information for the task. The resource usage information is similar to lsfRusage logged in the lsf.acct file; the difference is that some fields use the %1.0f format.
mailUser (%s)	Always empty
projectName (%s)	Always empty
exitStatus (%d)	UNIX exit status of the task
maxNumProcessors (%d)	Always 1
loginShell (%s)	Always empty
timeEvent (%s)	Always empty
idx (%d)	Session job index
maxRMem (%d)	Always 0
maxRSwap (%d)	Always 0
inFileSpool (%s)	Always empty

Field	Description
commandSpool (%s)	Always empty
rsvId (%s)	Always empty
sla (%s)	Always empty
exceptMask (%d)	Always 0
additionalInfo (%s)	Always empty
exitInfo (%d)	Always 0
warningAction (%s)	Always empty
warningTimePeriod (%d)	Always 0
chargedSAAP (%s)	Always empty
licenseProject (%s)	Always empty
app (%s)	Always empty
taskID (%d)	Task ID
taskIdx (%d)	Task index
taskName (%s)	Task name
taskOptions (%d)	Bit mask of task options: <ul style="list-style-type: none"> • TASK_IN_FILE (0x01): specify input file • TASK_OUT_FILE (0x02): specify output file • TASK_ERR_FILE (0x04): specify error file • TASK_PRE_EXEC (0x08): specify pre-execution command • TASK_POST_EXEC (0x10): specify post-execution command • TASK_NAME (0x20): specify task name
taskExitReason (%d)	Task exit reason: <ul style="list-style-type: none"> • TASK_EXIT_NORMAL = 0: normal exit • TASK_EXIT_INIT = 1: generic task initialization failure • TASK_EXIT_PATH = 2: failed to initialize path • TASK_EXIT_NO_FILE = 3: failed to create task file • TASK_EXIT_PRE_EXEC = 4: task pre-execution failed • TASK_EXIT_NO_PROCESS = 5: fork failed • TASK_EXIT_XDR = 6: XDR communication error • TASK_EXIT_NOMEM = 7: no memory • TASK_EXIT_SYS = 8: system call failed • TASK_EXIT_TSCHILD_EXEC = 9: failed to run sschild • TASK_EXIT_RUNLIMIT = 10: task reached its run limit • TASK_EXIT_IO = 11: input or output failure • TASK_EXIT_RSRC_LIMIT = 12: set task resource limit failed

Running and monitoring LSF Session Scheduler jobs

Create an LSF Session Scheduler session and run tasks

Procedure

1. Create task definition file.

For example:

```
cat my.tasks
sleep 10
hostname
uname
ls
```

2. Use **bsub** with the **ssched** application profile to submit an LSF Session Scheduler job with the task definition.

```
bsub -app ssched bsub_options ssched [task_options] [-tasks task_definition_file]
[command [arguments]]
```

For example:

```
bsub -app ssched ssched -tasks my.tasks
```

Results

When all tasks finish, LSF Session Scheduler exits, all temporary files are deleted, the session job is cleaned from the system, and LSF Session Scheduler output is captured and included in the standard LSF job e-mail.

You can also submit an LSF Session Scheduler job without a task definition file to specify a single task.

Note:

The submission directory path can contain up to 4094 characters.

See the **ssched** command reference for detailed information about all task options.

Submit an LSF Session Scheduler job as a parallel Platform LSF job

Procedure

Use the **-n** option of **bsub** to submit an LSF Session Scheduler job as a parallel LSF job.

```
bsub -app ssched -n num_hosts ssched [task_options] [-tasks task_definition_file]
[command [arguments]]
```

For example:

```
bsub -app ssched -n 2 ssched -tasks my.tasks
```

Submit task array jobs

Procedure

Use the **-J** option to submit a task array via the command line, and no task definition file is needed:

```
-J task_name[index_list]
```

The index list must be enclosed in square brackets. The index list is a comma-separated list whose elements have the syntax *start*[-*end*[:*step*]] where *start*, *end* and *step* are positive integers. If the step is omitted, a step of one (1) is assumed. The task array index starts at one (1).

All tasks in the array share the same option parameters. Each element of the array is distinguished by its array index.

See the **ssched** command reference for detailed information about all task options.

Submit tasks with automatic task requeue

Procedure

Use the **-Q** option to specify requeue exit values for the tasks:

```
-Q "exit_code ..."
```

-Q enables automatic task requeue and sets the LSB_EXIT_REQUEUE environment variable. Use spaces to separate multiple exit codes. LSF does not save the output from the failed task, and does not notify the user that the task failed.

If a job is killed by a signal, the exit value is 128+*signal_value*. Use the sum of 128 and the signal value as the exit code in the parameter. For example, if you want a task to rerun if it is killed with a signal 9 (SIGKILL), the exit value is 128+9=137.

The SSCHED_REQUEUE_LIMIT setting limits the number of times a task can be requeued.

See the ssched command reference for detailed information about all task options.

Integrate LSF Session Scheduler with bsub

Integrate LSF Session Scheduler with **bsub** to make the execution of LSF Session Scheduler jobs transparent. You can then use **bsub** to submit LSF Session Scheduler jobs without specifying the LSF Session Scheduler application profile and options.

The **bsub** command recognizes two environment variables to support LSF Session Scheduler job submission: LSB_TASKLIST (the task definition file) and LSB_BSUB_MODE (the current **bsub** mode). If LSB_BSUB_MODE is "ssched", running **bsub** does not submit a job to **mbatchd**. Instead, running **bsub** opens the task definition file (LSB_TASKLIST) and inserts the submitted job as a task into the task definition file.

This integration supports the following **bsub** options: -E, -Ep, -e, -i, -J, -j, -o, -M, -Q, and -W.

Other **bsub** options are ignored.

Set up the integrated execution environment

Create the script files necessary for setting up the execution environment to integrate LSF Session Scheduler with **bsub**.

Procedure

1. Create the begin_ssched.sh script, which creates an LSF Session Scheduler job and sets the necessary environment variables.

```
#!/bin/sh -x

TMPDIR=~/.ssched

LSB_TASKLIST=$TMPDIR/task.lst.$$
export LSB_TASKLIST

if [ ! -d $TMPDIR ]
then
    mkdir -p $TMPDIR
fi

#
# make sure no two sessions conflict each other
#
i=0
while [ -f $LSB_TASKLIST ]
do
    let i=i+1
    LSB_TASKLIST=$TMPDIR/task.lst.$$.$i
    export LSB_TASKLIST
done

JID=`bsub -H -Ep "rm -f $LSB_TASKLIST" $* ssched -tasks $LSB_TASKLIST | cut -f2 -d'<' |`
cut -f1 -d'>'`
export JID

LSB_BSUB_MODE=ssched
export LSB_BSUB_MODE
```

2. Create the end_ssched.sh script, to schedule and execute the LSF Session Scheduler job.

```
#!/bin/sh
```

```
bresume $JID > /dev/null 2>&1
```

```
unset LSB_BSUB_MODE
unset LSB_TASKLIST
```

3. Copy the two script files into the LSF_BINDIR directory.
4. Set the file permissions of the two script files to be executable for all users.

Use the integrated execution environment

Use **bsub** to submit LSF Session Scheduler jobs without specifying the LSF Session Scheduler application profile and options.

Procedure

1. Run the **begin_ssched.sh** script to create an LSF Session Scheduler job and set up the environment variables. You can use standard **bsub** options with **begin_ssched.sh** to apply to the session.

For example, to create a session job with two slots and send the output to a.out:

```
. begin_ssched.sh -n2 -o a.out
```

2. Run **bsub** for each batch job you want to include in the session. You can run **bsub** with the following options: -E, -Ep, -e, -i, -J, -j, -o, -M, -Q, and -W.
3. Run the **end_ssched.sh** script to have LSF create an LSF Session Scheduler job and set up the environment variables.

```
. end_ssched.sh
```

The task definition file is automatically deleted after the LSF Session Scheduler job is complete.

What to do next

You can also run these commands entirely from a script. For example:

```
#!/bin/sh
```

```
. begin_ssched.sh -n2
```

```
bsub task1
bsub task2
```

```
. end_ssched.sh
```

Monitor LSF Session Scheduler jobs

Procedure

1. Run **bjobs -ss** to get summary information for LSF Session Scheduler jobs and tasks.

JOBID	OWNER	JOB_NAME	NTASKS	PEND	DONE	RUN	EXIT
1	lsfadmin	job1	10	4	4	2	0
2	lsfadmin	job2	10	10	0	0	0
3	lsfadmin	job3	10	10	0	0	0

Information displays about your LSF Session Scheduler job, including Job ID, the owner, the job name, the number of total tasks, and the number of tasks in any of the following states: pend, run, done, exit.

2. Use **bjobs -l -ss** or **bread** to track the progress of the Session Scheduler job.

Kill an LSF Session Scheduler session

Procedure

Use **bkill** to kill the Session Scheduler session. All temporary files are deleted, and the session job is cleaned from the system.

Check your job submission

Procedure

Use the **-C** option to sanity-check all parameters and the task definition file.

ssched exits after the check is complete. An exit code of 0 indicates no errors were found. A non-zero exit code indicates errors. You can run **ssched -C** outside of LSF.

See the **ssched** command reference for detailed information about all task options.

Example output of **ssched -C**:

```
ssched -C -tasks my.tasks
Error in tasks file line 1: -XXX 123 sleep 0
Unsupported option: -XXX
Error in tasks file line 2: -o my.out
A command must be specified
```

Results

Only the **ssched** parameters are checked, not the **ssched** task command itself. The task command must exist and be executable. **ssched -C** cannot detect whether the task command exists or is executable. To check a task definitions file, remember to specify the **-tasks** option.

Enable recoverable LSF Session Scheduler sessions

About this task

By default, LSF Session Scheduler sessions are unrecoverable. In the event of a system crash, the session job must be resubmitted and all tasks are resubmitted and rerun.

However, the Session Scheduler supports application-level checkpoint/restart using Platform LSF's existing facilities. If the user specifies a checkpoint directory when submitting the session job, the job can be restarted using **brestart**. After a restart, only those tasks that have not yet completed are resubmitted and run.

Procedure

To enable recoverable sessions, when submitting the session job:

- Provide a writable directory on a shared file system.
- Specify the **ssched** checkpoint method with the **bsub -k** option.

Results

You do not need to call **bchkpnt**. The LSF Session Scheduler automatically checkpoints itself after each task completes.

Example

For example:

```
bsub -app ssched -k "/share/scratch method=ssched" -n 8 ssched -tasks simpton.tasks
Job <123> is submitted to default queue <normal>.
...
brestart /share/scratch 123
```

Troubleshooting

Use any of the following methods to troubleshoot your LSF Session Scheduler jobs.

ssched environment variables

Before submitting the **ssched** command, You can set the following environment variables to enable additional debugging information:

SSCHED_DEBUG_LOG_MASK=[LOG_INFO | LOG_DEBUG | LOG_DEBUG1 | ...]

Controls the amount of logging

SSCHED_DEBUG_CLASS=ALL or **SSCHED_DEBUG_CLASS**=[LC_TRACE] [LC_FILE] [...]

- Filters out some log classes, or shows all log classes
- By default, no log classes are shown

SSCHED_DEBUG_MODULES=ALL or **SSCHED_DEBUG_MODULES**=[ssched] [libvem.so] [sservice] [sschild]

- Enables logging on some or all components
- By default, logging is disabled on all components
- libvem.so controls logging by the libvem.so loaded by the SD, SSM and **ssched**
- Enabling debugging of the LSF Session Scheduler automatically enables logging by the libvem.so loaded by the Session Scheduler

SSCHED_DEBUG_REMOTE_HOSTS=ALL or **SSCHED_DEBUG_REMOTE_HOSTS**=[hostname1] [hostname2] [...]

- Enables logging on some/all hosts
- By default, logging is disabled on all remote hosts

SSCHED_DEBUG_REMOTE_FILE=Y

- Directs logging to /tmp/ssched/job_ID.job_index/ instead of stderr on each remote host
- Useful if too much debugging info is slowing down the network connection
- By default, debugging info is sent to stderr

ssched debug options

The **ssched** options -1, -2, and -3 are shortcuts for the following environment variables.

ssched -1

Is a shortcut for:

- **SSCHED_DEBUG_LOG_MASK**=LOG_WARNING
- **SSCHED_DEBUG_CLASS**=ALL
- **SSCHED_DEBUG_MODULES**=ALL

ssched -2

Is a shortcut for:

- **SSCHED_DEBUG_LOG_MASK**=LOG_INFO
- **SSCHED_DEBUG_CLASS**=ALL
- **SSCHED_DEBUG_MODULES**=ALL

ssched -3

Is a shortcut for:

- **SSCHED_DEBUG_LOG_MASK**=LOG_DEBUG
- **SSCHED_DEBUG_CLASS**=ALL
- **SSCHED_DEBUG_MODULES**=ALL

Example output of ssched -2:

Example output of ssched -2:

```
Nov 22 22:22:45 2022 18275 6 10.1.0 SSCHED_UPDATE_SUMMARY_INTERVAL = 1
Nov 22 22:22:45 2022 18275 6 10.1.0 SSCHED_UPDATE_SUMMARY_BY_TASK = 0
Nov 22 22:22:45 2022 18275 6 10.1.0 SSCHED_REQUEUE_LIMIT = 1
Nov 22 22:22:45 2022 18275 6 10.1.0 SSCHED_RETRY_LIMIT = 1
Nov 22 22:22:45 2022 18275 6 10.1.0 SSCHED_MAX_TASKS = 10
```

```

Nov 22 22:22:45 2022 18275 6 10.1.0 SSCHED_MAX_RUNLIMIT = 600
Nov 22 22:22:45 2022 18275 6 10.1.0 SSCHED_ACCT_DIR = /home/user1/ssched
Nov 22 22:22:45 2022 18275 6 10.1.0 Task <1> parsed.
Nov 22 22:22:45 2022 18275 6 10.1.0 Task <2> parsed.
Nov 22 22:22:45 2022 18275 6 10.1.0 Task <3> parsed.
Nov 22 22:22:45 2022 18275 6 10.1.0 Task <4> parsed.
Nov 22 22:22:45 2022 18275 6 10.1.0 Task <5> parsed.
Nov 22 22:22:47 2022 18275 6 10.1.0 Task <1> submitted. Command <sleep 0>;
Nov 22 22:22:47 2022 18275 6 10.1.0 Task <2> submitted. Command <sleep 0>;
Nov 22 22:22:47 2022 18275 6 10.1.0 Task <3> submitted. Command <sleep 0>;
Nov 22 22:22:47 2022 18275 6 10.1.0 Task <4> submitted. Command <sleep 0>;
Nov 22 22:22:47 2022 18275 6 10.1.0 Task <5> submitted. Command <sleep 0>;
Nov 22 22:22:54 2022 18275 6 10.1.0 Task <1> done successfully.
Nov 22 22:22:54 2022 18275 6 10.1.0 Task <2> done successfully.
Nov 22 22:22:54 2022 18275 6 10.1.0 Task <4> done successfully.
Nov 22 22:22:54 2022 18275 6 10.1.0 Task <3> done successfully.
Nov 22 22:22:54 2022 18275 6 10.1.0 Task <5> done successfully.

```

Task Summary

```

Submitted:      5
Done:           5

```

Example output of ssched -2 with requeue

```

Nov 22 22:28:36 2022 19409 6 10.1.0 SSCHED_UPDATE_SUMMARY_INTERVAL = 1
Nov 22 22:28:36 2022 19409 6 10.1.0 SSCHED_UPDATE_SUMMARY_BY_TASK = 0
Nov 22 22:28:36 2022 19409 6 10.1.0 SSCHED_REQUEUE_LIMIT = 1
Nov 22 22:28:36 2022 19409 6 10.1.0 SSCHED_RETRY_LIMIT = 1
Nov 22 22:28:36 2022 19409 6 10.1.0 SSCHED_MAX_TASKS = 10
Nov 22 22:28:36 2022 19409 6 10.1.0 SSCHED_MAX_RUNLIMIT = 600
Nov 22 22:28:36 2022 19409 6 10.1.0 SSCHED_ACCT_DIR = /home/user1/ssched
Nov 22 22:28:36 2022 19409 6 10.1.0 Task <1> parsed.
Nov 22 22:28:38 2022 19409 6 10.1.0 Task <1> submitted. Command <exit 1>;
Nov 22 22:28:43 2022 19409 6 10.1.0 Task <1> exited with code 1.
Nov 22 22:28:43 2022 19409 6 10.1.0 Task <1> submitted. Command <exit 1>;
Nov 22 22:28:43 2022 19409 6 10.1.0 Task <1> exited with code 1.

```

Task Summary

```

Submitted:      1
Requeued:       1

```

```

Done:           0

```

```

Exited:         2

```

```

Execution Errors: 2
Dispatch Errors:  0
Other Errors:     0

```

Task Error Summary

Execution Error

```

Task ID:      1
Submit Time:  Thu Nov 22 22:28:38 2022
Start Time:   Thu Nov 22 22:28:43 2022
End Time:     Thu Nov 22 22:28:43 2022
Exit Code:    1
Exit Reason:  Normal exit
Exec Hosts:   hostA
Exec Home:    /home/user1/
Exec Dir:     /home/user1/src/lsf10.1.0ss/ssched
Command:      exit 1
Action:       Requeue exit value match; task will be requeued

```

Execution Error

```

Task ID:      1
Submit Time:  Thu Nov 22 22:28:43 2022
Start Time:   Thu Nov 22 22:28:43 2022
End Time:     Thu Nov 22 22:28:43 2022
Exit Code:    1
Exit Reason:  Normal exit
Exec Hosts:   hostA
Exec Home:    /home/user1/

```

```

Exec Dir:          /home/user1/src/lsf10.1.0ss/ssched
Command:          exit 1
Action:           Task requeue limit reached; task will not be requeued

```

Example output of ssched -2 with retry

```

Nov 22 22:35:40 2022 20769 6 10.1.0 SSCHED_UPDATE_SUMMARY_INTERVAL = 1
Nov 22 22:35:40 2022 20769 6 10.1.0 SSCHED_UPDATE_SUMMARY_BY_TASK = 0
Nov 22 22:35:40 2022 20769 6 10.1.0 SSCHED_REQUEUE_LIMIT = 1
Nov 22 22:35:40 2022 20769 6 10.1.0 SSCHED_RETRY_LIMIT = 1
Nov 22 22:35:40 2022 20769 6 10.1.0 SSCHED_MAX_TASKS = 10
Nov 22 22:35:40 2022 20769 6 10.1.0 SSCHED_MAX_RUNLIMIT = 600
Nov 22 22:35:40 2022 20769 6 10.1.0 SSCHED_ACCT_DIR = /home/user1/ssched
Nov 22 22:35:40 2022 20769 6 10.1.0 Task <1> parsed.
Nov 22 22:35:42 2022 20769 6 10.1.0 Task <1> submitted. Command <sleep 0>;
Nov 22 22:35:47 2022 20769 6 10.1.0 Task <1> had a dispatch error. Task will be retried.
Nov 22 22:35:47 2022 20769 6 10.1.0 Task <1> submitted. Command <sleep 0>;
Nov 22 22:35:47 2022 20769 6 10.1.0 Task <1> had a dispatch error. Retry limit reached.

```

Task Summary

```

Submitted:        1
Done:             0
Exited:          1
  Execution Errors: 0
  Dispatch Errors: 1
  Other Errors:    0

```

Task Error Summary

Dispatch Error

```

Task ID:          1
Submit Time:      Thu Nov 22 22:35:47 2022
Failure Reason:   Pre-execution command failed
Command:          sleep 0
Pre-Exec:         exit 1
Start time:       Thu Nov 22 22:35:47 2022
Execution host:   hostA
Action:           Task retry limit reached; task will not be retried

```

Note:

The "Task Summary" and "Summary of Errors" sections are sent to stdout. All other output is sent to stderr.

Send SIGUSR1 signal

After the tasks have been submitted to the LSF Session Scheduler and started, users can enable additional debugging by LSF Session Scheduler components by sending a SIGUSR1 signal.

To enable additional debugging by the **ssched** and **libvem** components, send a SIGUSR1 to the ssched_real process. This enables the following:

- SSCHED_DEBUG_LOG_MASK=LOG_DEBUG
- SSCHED_DEBUG_CLASS=ALL
- SSCHED_DEBUG_MODULES=ALL

The additional log messages are sent to stderr.

To enable additional debugging by the **sservice** and **sschild** components, send a SIGUSR1 on the remote host to the **sservice** process. This enables the following:

- SSCHED_DEBUG_LOG_MASK=LOG_DEBUG
- SSCHED_DEBUG_CLASS=ALL
- SSCHED_DEBUG_MODULES=ALL
- SSCHED_DEBUG_REMOTE_HOSTS=ALL
- SSCHED_DEBUG_REMOTE_FILE=Y

The debug messages are saved to a file in /tmp/ssched/. You are responsible for deleting this file when it is no longer needed.

Send SIGUSR2 signal

If a SIGUSR1 signal is sent, SIGUSR2 restores debugging to its original level.

Known issues and limitations

General issues

- The LSF Session Scheduler caches host info from LIM. If the host factor of a host is changed after the LSF Session Scheduler starts, the LSF Session Scheduler will not see the updated host factor. The host factor is used in the task accounting log.
- LSF Session Scheduler does not support per task memory or swap utilization tracking from **ssacct**. Run **bacct** to see aggregate memory and swap utilization.
- When specifying a multiline command line as a **ssched** command line parameter, you must enclose the command in quotes. A multiline command line is any command containing a semi-colon (;). For example:

```
ssched -o my.out "hostname; ls"
```

When specifying a multiline command line as a parameter in a task definition file, you must NOT use quotes. For example:

```
cat my.tasks  
-o my.out hostname; ls
```

- If you submit a shell script containing multiple **ssched** commands, **bjobs -l** only shows the task summary for the currently running **ssched** instance. Enable task accounting and examine the accounting file to see information for tasks from all **ssched** instances in the shell script.
- Submitting a large number of tasks as part of one session may cause a slight delay between when the LSF Session Scheduler starts and when tasks are dispatched to execution agents. The LSF Session Scheduler must parse and submit each task before it begins dispatching any tasks. Parsing 50,000 tasks can take up to 2 minutes before dispatching starts.
- After all tasks have completed, the LSF Session Scheduler will take some time to terminate all execution agents and to clean up temporary files. A minimum of 20 seconds is normal, longer for larger allocations.
- LSF Session Scheduler handles the following signals: SIGINT, SIGTERM, SIGUSR1, SIGSTOP, SIGTSTP, and SIGCONT. All other signals cause **ssched** to exit immediately. No summary is output and task accounting information is not saved. The signals LSF Session Scheduler handles will be expanded in future releases.

IBM Spectrum LSF on Cray Linux

The IBM® Spectrum LSF integration with Cray Linux applies to LSF Version 8.0 or later, and supports integration with Cray Linux Environment 4.0 or later. You must have LSF Standard Edition or LSF Advanced Edition. The Cray Linux integration is not supported on LSF Express Edition.

- [Downloading and installing the integration package](#)
Download and install the integration package to run LSF on Cray Linux systems
- [Configuring the LSF integration on Cray Linux](#)
Set configuration parameters for the LSF integration on Cray Linux
- [Integration directory and file structure](#)
After installation and configuration, the LSF integration on Cray Linux is installed under LSF_TOP (for example, /software/lsf/).
- [Submitting and running jobs](#)
Submit and run jobs with the LSF integration on Cray Linux.
- [Assumptions and limitations](#)
After the patch has been installed and configured, advance reservation, preemption, and reservation scheduling policies are supported with some limitations.

Downloading and installing the integration package

Download and install the integration package to run LSF on Cray Linux systems

Procedure

1. Download the installation package and the distribution tar file for the LSF integration on Cray Linux (on Cray XT/XE/XC) integration.

For example, in LSF Version 10.1.0, the following files are needed:

- lsf10.1.0_inx26-lib23-x64-cray.tar.Z
- Intstaller packages:
 - lsf10.1.0_lsfinstall.tar.Z
This is the standard installer package. Use this package in a heterogeneous cluster with a mix of systems other than x86-64 (except zLinux). Requires approximately 1 GB free space.
 - lsf10.1.0_lsfinstall_linux_x86_64.tar.Z
Use this smaller installer package in a homogeneous x86-64 cluster. If you add other non x86-64 hosts you must use the standard installer package. Requires approximately 100 MB free space.
 - lsf10.1.0_no_jre_lsfinstall.tar.Z For all platforms not requiring the JRE. JRE version 1.4 or higher must already be installed on the system. Requires approximately 1 MB free space.

2. Confirm that the Cray Linux system is working.

- a. On CLE 4.0 or later, confirm the existence of /opt/cray/rca/default/bin/rca-helper, /etc/xhostname and /etc/opt/cray/sdb/node_classes. Otherwise, confirm that the **xtuname** and **xhostname** commands exist and are in the \$PATH.
- b. Confirm that all compute PEs are in batch mode. If not, switch all compute PEs to batch mode and restart ALPS services on the boot node:
 - **xtprocadm -k m batch**
 - **\$/etc/init.d/alps restart** (optional)
 - **apstat -rn** (optional)

3. Follow the standard LSF installation procedure to install LSF on the boot nodes.

- a. Run the **xtopview** command to switch to a shared root file system.
- b. Add the Cray Linux machine to an existing cluster.
For more details, refer to the upgrade/migration instructions.
- c. Edit the install.config file and set the following installation parameters:
 - LSF_TOP=/software/lsf
 - LSF_CLUSTER_NAME=<crayxt_machine_name>
 - LSF_MASTER_LIST=<management_host_candidates> (a list of login node names or service node names)
 - EGO_DAEMON_CONTROL=N
 - ENABLE_DYNAMIC_HOSTS=N
 - LSF_ADD_SERVERS=<service or login nodes>
 - CONFIGURATION_TEMPLATE=PARALLEL

The LSF_MASTER_LIST and LSF_ADD_SERVERS parameters should only include login nodes or service nodes.

The startup and shutdown script for LSF daemons is located in \$LSF_SERVERDIR/lsf_daemons.

4. As the LSF administrator, create and use the **LSF-HPC** service.

- a. Add the following lines to the /opt/xt-boot/default/etc/serv_cmd file:
 - service_cmd_info='LSF-HPC',service_num=XXX,heartbeat=null
 - start_cmd='<\$LSF_SERVERDIR>/lsf_daemons start'
 - stop_cmd='<\$LSF_SERVERDIR>/lsf_daemons stop'
 - restart_cmd='<\$LSF_SERVERDIR>/lsf_daemons restart'
 - fail_cmd='<\$LSF_SERVERDIR>/lsf_daemons stop'
- b. Create a service command: **xtservcmd2db -f /opt/xt-boot/default/etc/serv_cmd**.
- c. Assign the LSF-HPC service to **serv_cmd**: **xtservconfig -c login add LSF-HPC**.
- d. Exit **xtopview** and access a login node.

- Make sure /ufs is shared among all login/service nodes and root and LSF administrators have write permission.
 - Set up sub-directories under /ufs the same as /opt/xt-lsfhpc/log and /opt/xt-lsfhpc/work (see section "File Structure" for details).
 - Make sure the directory ownership and permission mode are reserved (you can use the **cp -r** command), and that root and LSF administrators have write permission to the sub-directories under **/ufs/lsfhpc**.
5. Use the **module** command to set the LSF environment variables.
- ```
module load xt-lsfhpc
```

## Configuring the LSF integration on Cray Linux

Set configuration parameters for the LSF integration on Cray Linux

### Procedure

1. Modify \$LSF\_ENVDIR/lsf.conf.

Some of the following parameters may have been added by the LSF installation:

- LSB\_SHAREDIR=/ufs/lsfhpc/work - A shared file system that is accessible by root and the LSF administrator on both management hosts and Cray Linux login/service nodes.
- LSF\_LOGDIR=/ufs/lsfhpc/log - A shared file system that is accessible by root and the LSF administrator on both management hosts and Cray Linux login/service nodes.
- LSF\_LIVE\_CONFDIR=/ufs/lsfhpc/work/<cluster\_name>/live\_confdir - A shared file system that is accessible by root and the LSF administrator on both management hosts and Cray Linux login/service nodes.
- LSB\_RLA\_PORT=21787 - a unique port
- LSB\_SHORT\_HOSTLIST=1
- LSF\_ENABLE\_EXTSCHEDULER=Y
- LSB\_SUB\_COMMANDNAME=Y
- LSF\_CRAY\_PS\_CLIENT=/usr/bin/apbasil
- LSF\_LIMSIM\_PLUGIN="liblimsim\_craylinux"
- LSF\_CRAYLINUX\_FRONT\_NODES="nid00060 nid00062" - A list of Cray Linux login/service nodes with LSF daemons started and running.
- LSF\_CRAYLINUX\_FRONT\_NODES\_POLL\_INTERVAL=120 - Interval for the management host LIM polling RLA to query computer node status and configuration information. Default value is 120 seconds. Any value less than 120 seconds will be reset to default
- LSB\_MIG2PEND=1
- LSF\_CRAY\_RUR\_DIR=/ufs/lsfhpc/work/<cluster\_name>/craylinux/<cray\_machine\_name>/rur - Location of the RUR data files, which is a shared file system that is accessible from any potential first execution host. An RUR data file for jobs that are submitted by all users is named rur.output. A job-specific RUR data file for specific job IDs are named rur.<jobID>. The default value is  
LSF\_SHARED\_DIR/<cluster\_name>/craylinux/<cray\_machine\_name>/rur.  
You can use the %U special character to represent the home directory of the user that submitted the job. For example, if you specify LSF\_CRAY\_RUR\_DIR=%U/.rur, and userA and userB submitted jobs, the RUR data files are located in /home/userA/.rur for userA and /home/userB/.rur for userB.
- LSF\_CRAY\_RUR\_PROLOG\_PATH=<path\_to\_rur\_prologue.py> - File path to the RUR prolog script file. Default value is /opt/cray/rur/default/bin/rur\_prologue.py.  
Note: LSF runs the prolog script file with the -j <jobID> option. Therefore, the prolog script file must support the -j option with the job ID as the argument.
- LSF\_CRAY\_RUR\_EPILOG\_PATH=<path\_to\_rur\_epilogue.py> - File path to the RUR epilog script file. Default value is /opt/cray/rur/default/bin/rur\_epilogue.py.  
Note: LSF runs the epilog script file with the -j <jobID> option. Therefore, the epilog script file must support the -j option with the job ID as the argument.

2. From a Cray login node, run the \$LSF\_BINDIR/genVnodeConf command.

This command generates a list of compute nodes in BATCH mode. You can add the compute nodes to the HOST section in \$LSF\_ENVDIR/lsf.cluster.<cluster\_name>.

| HOSTNAME | model | type | server | r1m | mem | swp | RESOURCES         |
|----------|-------|------|--------|-----|-----|-----|-------------------|
| nid00038 | !     | !    | 1      | 3.5 | ()  | ()  | (craylinux vnode) |

|          |   |   |   |     |    |    |                       |
|----------|---|---|---|-----|----|----|-----------------------|
| nid00039 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00040 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00041 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00042 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode gpu) |
| nid00043 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode gpu) |
| nid00044 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00045 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00046 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00047 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00048 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00049 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00050 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00051 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00052 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode gpu) |
| nid00053 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode gpu) |
| nid00054 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00055 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00056 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |
| nid00057 | ! | ! | 1 | 3.5 | () | () | (craylinux vnode)     |

### 3. Configure \$LSF\_ENVDIR/hosts.

Make sure that the IP addresses of compute nodes do not conflict with any IP address that is already in use.

```
cat $LSF_ENVDIR/hosts
```

|                |          |            |        |        |           |
|----------------|----------|------------|--------|--------|-----------|
| 10.128.0.34    | nid00033 | c0-0cls0n3 | sdb001 | sdb002 |           |
| 10.128.0.61    | nid00060 | c0-0cls1n0 | login  | login1 | castor-p2 |
| 10.128.0.36    | nid00035 | c0-0cls1n3 |        |        |           |
| 10.128.0.59    | nid00058 | c0-0cls2n0 |        |        |           |
| 10.128.0.38    | nid00037 | c0-0cls2n3 |        |        |           |
| 10.128.0.57    | nid00056 | c0-0cls3n0 |        |        |           |
| 10.128.0.58    | nid00057 | c0-0cls3n1 |        |        |           |
| 10.128.0.39    | nid00038 | c0-0cls3n2 |        |        |           |
| 10.128.0.40    | nid00039 | c0-0cls3n3 |        |        |           |
| 10.128.0.55    | nid00054 | c0-0cls4n0 |        |        |           |
| 10.128.0.56    | nid00055 | c0-0cls4n1 |        |        |           |
| 10.128.0.41    | nid00040 | c0-0cls4n2 |        |        |           |
| 10.128.0.42    | nid00041 | c0-0cls4n3 |        |        |           |
| 10.128.0.53    | nid00052 | c0-0cls5n0 |        |        |           |
| 10.128.0.54    | nid00053 | c0-0cls5n1 |        |        |           |
| 10.128.0.43    | nid00042 | c0-0cls5n2 |        |        |           |
| 10.128.0.44    | nid00043 | c0-0cls5n3 |        |        |           |
| 10.128.0.51    | nid00050 | c0-0cls6n0 |        |        |           |
| 10.128.0.52    | nid00051 | c0-0cls6n1 |        |        |           |
| 10.128.0.45    | nid00044 | c0-0cls6n2 |        |        |           |
| 10.128.0.46    | nid00045 | c0-0cls6n3 |        |        |           |
| 10.128.0.49    | nid00048 | c0-0cls7n0 |        |        |           |
| 10.128.0.50    | nid00049 | c0-0cls7n1 |        |        |           |
| 10.128.0.47    | nid00046 | c0-0cls7n2 |        |        |           |
| 10.128.0.48    | nid00047 | c0-0cls7n3 |        |        |           |
| 10.131.255.251 | sdb      | sdb-p2     | syslog | ufs    |           |

### 4. Modify \$LSF\_ENVDIR/lbbatch/<cluster\_name>/configdir/lbhosts.

Make sure to set a large number in the MXJ column for the Cray Linux login and service nodes that are also LSF server hosts. The number should be larger than the total number of PEs.

```
Begin Host
```

| HOST_NAME | MXJ  | rlm | pg | ls | tmp | DISPATCH_WINDOW | # Keywords |
|-----------|------|-----|----|----|-----|-----------------|------------|
| nid00060  | 9999 | ()  | () | () | ()  | ()              | # Example  |
| nid00062  | 9999 | ()  | () | () | ()  | ()              | # Example  |
| default   | !    | ()  | () | () | ()  | ()              | # Example  |

```
End Host
```

### 5. Modify \$LSF\_ENVDIR/lbbatch/<cluster\_name>/configdir/lbqueues.

- JOB\_CONTROLS and RERUNNABLE are required.
- Comment out all **loadSched/loadStop** lines.
- DEFAULT\_EXTSCHEd and MANDATORY\_EXTSCHEd are optional.
- To run CCM jobs, you must get the pre-execution and post-execution binary files from Cray. Refer to Cray documentation to find these files.

```

Begin Queue
QUEUE_NAME = normal
PRIORITY = 30
NICE = 20
PREEMPTION = PREEMPTABLE
JOB_CONTROLS = SUSPEND[bmig $LSB_BATCH_JID]
RERUNNABLE = Y
#RUN_WINDOW = 5:19:00-1:8:30 20:00-8:30
#r1m = 0.7/2.0 # loadSched/loadStop
#r15m = 1.0/2.5
#pg = 4.0/8
#ut = 0.2
#io = 50/240
#CPULIMIT = 180/hostA # 3 hours of hostA
#FILELIMIT = 20000
#DATA LIMIT = 20000 # jobs data segment limit
#CORELIMIT = 20000
#TASKCLIMIT = 5 # job task limit
#USERS = all # users who can submit jobs to this queue
#HOSTS = all # hosts on which jobs in this queue can run
#PRE_EXEC = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
#POST_EXEC = /usr/local/lsf/misc/testq_post |grep -v "Hey"
#REQUEUE_EXIT_VALUES = 55 34 78
#APS_PRIORITY = WEIGHT[[RSRC, 10.0] [MEM, 20.0] [PROC, 2.5] [QRIORITY, 2.0]] \
#LIMIT[[RSRC, 3.5] [QRIORITY, 5.5]] \
#GRACE_PERIOD[[QRIORITY, 200s] [MEM, 10m] [PROC, 2h]]
DESCRIPTION = For normal low priority jobs, running only if hosts are lightly loaded.
End Queue

```

```

Begin Queue
QUEUE_NAME = owners
PRIORITY = 43
JOB_CONTROLS = SUSPEND[bmig $LSB_BATCH_JID]
RERUNNABLE = YES
PREEMPTION = PREEMPTIVE
NICE = 10
#RUN_WINDOW = 5:19:00-1:8:30 20:00-8:30
r1m = 1.2/2.6
#r15m = 1.0/2.6
#r15s = 1.0/2.6
pg = 4/15
io = 30/200
swp = 4/1
tmp = 1/0
#CPULIMIT = 24:0/hostA # 24 hours of hostA
#FILELIMIT = 20000
#DATA LIMIT = 20000 # jobs data segment limit
#CORELIMIT = 20000
#TASKCLIMIT = 5 # job task limit
#USERS = user1 user2
#HOSTS = hostA hostB
#ADMINISTRATORS = user1 user2
#PRE_EXEC = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
#POST_EXEC = /usr/local/lsf/misc/testq_post |grep -v "Hey"
#REQUEUE_EXIT_VALUES = 55 34 78
DESCRIPTION = For owners of some machines, only users listed in the HOSTS\
section can submit jobs to this queue.
End Queue

```

#### 6. Modify \$LSF\_ENVDIR/lsf.shared.

Make sure the following boolean resources are defined in the RESOURCE section:

```

vnode Boolean () () (sim node)
gpu Boolean () () (gpu)
frontnode Boolean () () (login/service node)
craylinux Boolean () () (Cray XT/XE MPI)

```

#### 7. By default, LSF\_CRAY\_RUR\_ACCOUNTING=Y is enabled for LSF to work with Resource Utility Reporting (RUR). If RUR is not installed in your environment, you must disable RUR by setting LSF\_CRAY\_RUR\_ACCOUNTING=N in lsf.conf.

#### 8. Modify /etc/opt/cray/rur/rur.conf.

Disable the default prolog and epilog scripts by commenting out the following lines in the **apsys** section:

## apsys

```
prologPath - location of the executable file to be run before application
prologPath /usr/local/adm/sbin/prolog
epilogPath - location of the executable file to be run after application
epilogPath /usr/local/adm/sbin/epilog
prologTimeout - time in seconds before prolog is aborted as "hung"
prologTimeout 10
epilogTimeout - time in seconds before epilog is aborted as "hung"
epilogTimeout 10
prologPath /opt/cray/rur/default/bin/rur_prologue.py
epilogPath /opt/cray/rur/default/bin/rur_epilogue.py
prologTimeout 100
epilogTimeout 100
```

/apsys

9. Modify /etc/opt/cray/alps/alps.conf.

Disable the default prolog and epilog scripts by commenting out the following lines in the **apsys** section:

## apsys

```
prologPath - location of the executable file to be run before application
prologPath /usr/local/adm/sbin/prolog
epilogPath - location of the executable file to be run after application
epilogPath /usr/local/adm/sbin/epilog
prologTimeout - time in seconds before prolog is aborted as "hung"
prologTimeout 10
epilogTimeout - time in seconds before epilog is aborted as "hung"
epilogTimeout 10
prologPath /opt/cray/rur/default/bin/rur_prologue.py
epilogPath /opt/cray/rur/default/bin/rur_epilogue.py
prologTimeout 100
epilogTimeout 100
```

/apsys

10. Restart the **alps** daemon on the login nodes to apply the changes to the alps.conf and rur.conf file.

/etc/init.d/alps restart

11. Use the **service** command to start and stop the LSF services as needed.

- service LSF-HPC start
- service LSF-HPC stop

---

## Integration directory and file structure

After installation and configuration, the LSF integration on Cray Linux is installed under LSF\_TOP (for example, /software/lsf/).

### File Structure

---

The integration after installation has the following directory layout:

```
/ufs
|-- lsfhpc
| |-- log
| |
| |-- work
| |-- <cluster_name>
| |-- craylinux
| |-- logdir
| |-- lsf_cmddir
| |-- live_confdir
| |-- lsf_indir
```

The following directories and three files are located in /software/lsf/:

```

|--<version>
|
|-- include
| |-- lsf
|
|-- install
| |-- instlib
| |-- patchlib
| |-- scripts
|
|-- linux2.6-glibc2.3-x86_64-cray
| |-- bin
| |-- etc
| | |-- scripts
| | |-- lib
|
|-- man
| |-- man1
| |-- man3
| |-- man5
| |-- man8
|
|-- misc
| |-- conf_tmpl
| | |-- eservice
| | | |-- esc
| | | | |-- conf
| | | | |-- services
| | | |-- esd
| | | | |-- conf
| | | | |-- named
| | | | | |-- conf
| | | | | |-- namedb
| | |-- kernel
| | | |-- conf
| | | | |-- mibs
| | |-- log
| | |-- work
|
| |-- config
| |-- examples
| | |-- blastparallel
| | |-- blogin
| | |-- dr
| | |-- eevent
| | |-- external_plugin
| | |-- extsched
| | |-- reselim
| | |-- web-lsf
| | | |-- cgi-bin
| | | |-- doc
| | | |-- lsf_html
| | |-- xelim
| |-- lsmake
| |-- src
|
|-- schema
| |-- samples
|
|-- scripts
|
|-- conf
| |-- ego
| | |-- <cluster_name>
| | | |-- eservice
| | | | |-- esc
| | | | | |-- conf
| | | | | |-- services
| | | |-- esd
| | | | |-- conf
| | | | |-- named
| | | | | |-- conf
| | | | | |-- namedb
| | |-- kernel
| | |-- mibs
| |-- lsbatch
| | |-- <cluster_name>
| | |-- configdir
|
|-- log
|-- patch
|-- backup

```

```
| |-- lock
| |-- patchdb
| |-- PackageInfo_LSF<version>_linux2.6-glibc2.3-x86_64-cray
|-- work
|-- <cluster_name>
| |-- ego
| |-- live_confdir
| |-- logdir
| |-- lsf_cmddir
| |-- lsf_indir
```

## Submitting and running jobs

Submit and run jobs with the LSF integration on Cray Linux.

### Before you begin

Before you submit jobs to the cluster, be aware that CLE4.0 does not support multiple jobs running on one compute node. All ALPS reservations created by LSF will have the "mode=EXCLUSIVE" attribute.

### About this task

Define a limit to make sure LSF does not dispatch jobs to compute nodes where a job has been running.

## Procedure

1. Modify \$LSF\_ENVDIR/lsubatch/<cluster\_name>/configdir/lsb.resources:

```
Begin Limit
NAME = COMPUTE_NODES_LIMIT
USERS = all
PER HOST = all ~<login_nodes>
JOBS = 1
End Limit
```

2. Note: There are other ways in LSF to enforce this limitation for ALPS.  
To submit a job that requires Cray Linux reservations (for example, an **aprun** job or a CCM job), you must use compound resource requirements.

```
bsub -extsched "CRAYLINUX[]" -R "1*{select[craylinux && \!vnode]} +
n*{select[vnode && craylinux] span[ptile=q*p]}" aprun -n y -d p -N q a.out
```

*n* must be greater than or equal to  $\text{MAX}(y*p, p*q)$ . The default of *y p q* is 1).

3. To submit a job that requires Cray Linux reservations with GPU (for example, an **aprun** job or a CCM job), use GPU on the CRAYLINUX external scheduler option.

```
bsub -extsched "CRAYLINUX[GPU]" -R "1*{select[craylinux && \!vnode]} + n*{select[vnode
&& craylinux && gpu] span[ptile=q*p] rusage[jobcnt=1]}" aprun -n y -d p -N q a.out
```

*n* must be greater than or equal to  $\text{MAX}(y*p, p*q)$ . The default of *y p q* is 1).

## What to do next

1. To submit a job that runs on Cray service login nodes without creating Cray Linux reservations:  

```
bsub -R "select[craylinux && frontnodes]" hostname
```
2. The following jobs with wrong RESREQ will be detected and put in pending state:
  - Jobs asking for a vnode but without CRAYLINUX[] specified. The pending reason is: The job cannot run on hosts with vnode.



- Jobs with `CRAYLINUX[]` but the allocation by LSF does not contain at least one frontend node and at least one vnode. The pending reason is: `Cannot create/confirm a reservation by apbasil/catnip`
3. To create an advance reservation, complete the following steps:
    - a. Create an advance reservation on compute nodes (hosts with `craylinux` && `vnode` resources).
    - b. Add slots on frontend nodes (hosts with `craylinux` && `!\vnode` resources).
    - c. Submit jobs and specify the advance reservation for the job.
  4. Use the **`bjobs`**, **`bhist`**, and **`bacct`** commands to display the reservation ID (`reservation_id`) under `additionalInfo`.
  5. If `LSF_COLLECT_ENERGY_USAGE=Y` is specified in `lsf.conf`, you can display the energy consumption by using the **`bacct -l`** command.

---

## Assumptions and limitations

After the patch has been installed and configured, advance reservation, preemption, and reservation scheduling policies are supported with some limitations.

- Not all scheduling policies behave the same way or automatically support the same things as standard LSF. ALPS in CLE4.0 only supports node exclusive reservations (no two jobs can run on the same node). Resource reservations (slot and resource) in LSF are impacted as jobs that reserved slots may not be able to run due to this ALPS limitation.
- Only one Cray Linux machine per cluster is allowed.
- Host based resource usage (`LSF_HPC_EXTENSIONS="HOST_RUSAGE"`) is not supported with Resource Utility Reporting (RUR).
- The LSF administrator must enable RUR plugins, including output plugins, to ensure that the `LSF_CRAY_RUR_DIR` directory contains per-job accounting files (`rur.<job_id>`) or a flat file (`rur.output`).