

# SAS® Scheduling: Getting the Most Out of Your Time and Resources

Allen Tran, Platform Computing Corporation, Markham, Ontario

Randy Williams, SAS Institute Inc., Cary, NC

Alan Wong, Platform Computing Corporation, Markham, Ontario

## ABSTRACT

This paper illustrates how organizations can better leverage the scheduling capabilities in SAS software. If you are new to SAS Scheduling, you will learn how to take advantage of the integrated SAS Scheduling capabilities for your SAS applications. For those already implementing SAS Scheduling solutions, information will be given on how to extend your scheduling capabilities within your environment. Some of the topics covered include various configurations of operating systems and machines, methods for implementing site policies, and ways to extend the capabilities through the integration with Platform JobScheduler for SAS.

## INTRODUCTION

Platform JobScheduler for SAS is an integrated, enterprise job scheduler that is specifically designed to manage your complex flows of SAS jobs more efficiently. Platform JobScheduler for SAS includes Platform LSF (an execution agent) and is available for use at no extra cost to customers who have purchased a SAS Enterprise ETL Server technology package. SAS Scheduling is directly integrated with SAS ETL Studio, SAS Marketing Automation, and SAS Web Report Studio. Platform JobScheduler for SAS is unlike other job schedulers because it offers resource virtualization, optimal resource sharing, enterprise scalability, and seamless manageability through resource clustering. Platform Computing and SAS are continuing to extend the integration to include other SAS applications.

Some of the benefits of Platform JobScheduler for SAS include:

- *Automation:* It uses sophisticated event-driven scheduling to reliably automate SAS workload processing. SAS flows are captured, graphically created, and stored for easy reuse in the future.
- *Reliability:* It ensures that SAS flows are automatically distributed for processing on available hosts.
- *Scalability:* Built on highly scalable grid technology, it is ideal for any complex IT environment that requires the capacity to support the mission-critical execution of jobs across your compute grids.
- *Effective Prioritization of Workload:* It offers a rich set of configuration options for building queues based on job priority, departmental policies, or project requirements.
- *Upgradeability:* You can extend the functionality of Platform JobScheduler for non-SAS jobs by upgrading to the full version.

## THE SAS SCHEDULING SOLUTION

### ARCHITECTURE

The architecture of the SAS Scheduling solution consists of three areas that enable your enterprise to configure and set up your environment, create jobs and schedule flows, and then execute the flows. These areas are described in the following three sections.

### Metadata Configuration

SAS configuration is handled by SAS Management Console and stored in the SAS Metadata Server. Metadata for servers, users, groups, flows, and jobs that are deployed from SAS applications is captured in the SAS Metadata Server. Scheduling and batch servers are defined in the Server Manager plug-in to SAS Management Console. Scheduling servers are third-party software applications, and batch servers are templates to command-line interfaces to SAS applications. The current types of batch servers are: the SAS Data Step Batch Server, the SAS Java Batch Server, and the SAS Generic Batch Server.

## Creation and Management

Flow creation and management tasks are performed by the Schedule Manager plug-in to SAS Management Console, along with scheduling integrated SAS applications, such as SAS ETL Studio and SAS Marketing Automation Campaign Manager. Some SAS applications, such as SAS Web Report Studio, can perform flow creation and management from within their own application.

## Execution

A *flow* contains one or more jobs that are deployed for scheduling by the scheduling integrated SAS applications. Flows are created and maintained in the Schedule Manager. A deployed job is associated with a batch server. Flow execution is handled by scheduling servers. In this paper the scheduling server is Platform JobScheduler, which uses Platform LSF. Platform JobScheduler is responsible for managing inter-job, time, and file dependencies for your jobs. It submits jobs (whose dependencies are satisfied) to Platform LSF for execution.

To schedule SAS jobs, as used by SAS applications, the jobs must be deployed to a SAS batch server in a scheduling-participating SAS application. Figure 1 illustrates how a SAS application fits into the overall architecture.

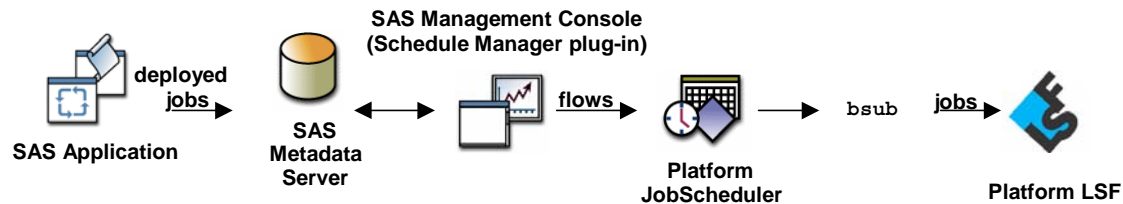


Figure 1. Overall Architecture

As seen in Figure 1, SAS applications must deploy jobs for scheduling, which saves information about the jobs in the SAS Metadata Server. These jobs are loaded into SAS Management Console. Flows are created in SAS Management Console using Schedule Manager, and are then scheduled in Platform JobScheduler, which submits the jobs to Platform LSF for execution.

Schedule Manager gets metadata about the flow from the SAS Metadata Server, converts that metadata to metadata that the underlying scheduler (Platform JobScheduler) understands, and submits the information to the scheduling server. Each job can have combinations of time, other jobs, or file dependencies. After dependencies are defined for each job, the jobs in the flow can be executed.

## SUPPORTED HOST OPERATING SYSTEMS AND CONFIGURATION

### OEM Host Operating Systems

SAS redistributes Platform JobScheduler for SAS (an OEM version of Platform JobScheduler) for the intersection of supported hosts that are part of the SAS Business Intelligence (BI) platform. Table 1 lists the mutually supported hosts on Windows, UNIX, and Linux.

Microsoft	UNIX/Linux
Windows Server 2003 (including 64-bit Itanium-based systems)	64-bit enabled Solaris - Solaris 8 and 9
Windows XP (including 64-bit Itanium-based systems)	64-bit enabled AIX - AIX (64-bit) 5.1 and 5.2
Windows 2000	HP-UX 11.22 (IPF version)
Windows NT and Windows NT Server	64-bit enabled Linux for Itanium, RHEL 3
	32-bit enabled Linux for Intel, glib 2.2 and 2.3
	HP-UX 64-bit enabled PA-RISC

Table 1. Supported Host Operating Systems

A typical install places SAS Management Console (including the Schedule Manager plug-in), Platform JobScheduler, and Platform LSF components on a single host. However, this is not the required configuration and these components can be placed on one or more hosts.

### Non-OEM Host Operating Systems

There are two options for how SAS supports a host environment in which SAS does not distribute an OEM version of Platform JobScheduler.

- *SAS/CONNECT Option.* SAS supports this host environment, but Platform Computing does not (for example, z/OS). This scenario can be addressed by leveraging SAS technology within your SAS program to handle remote submission of SAS statements to the desired host.
- *Platform LSF Option.* Both SAS and Platform Computing support this host environment; however, an OEM version of Platform JobScheduler is not provided (for example, Tru64). This scenario can be addressed by purchasing additional licenses from Platform Computing.

#### SAS/CONNECT Option

You can use SAS/CONNECT technology to remotely submit a SAS statement to the host while scheduling the initial job/flow on a BI platform (see Figure 2). The SAS statement could exploit other SAS technology to accomplish the same task of executing SAS statements on a remote host. Figure 2 illustrates configuring SAS/CONNECT technology for remote submission.

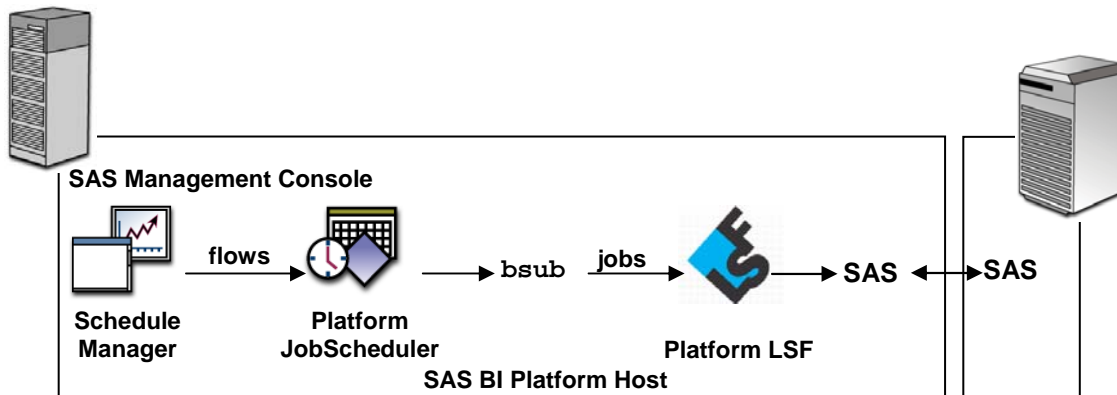


Figure 2. The SAS/CONNECT Option

## Platform LSF Option

You can use Platform LSF to provide scheduling on non-OEM hosts, but it requires purchasing an additional Platform LSF license for that host. The additional Platform LSF needs to work with Platform JobScheduler in a mixed cluster environment. Platform JobScheduler and Platform LSF must both be installed on one of the supported BI platform hosts. Next, the additional Platform LSF must be installed on the new host in the mixed cluster environment. You can purchase a license from Platform Computing for the additional Platform LSF or extra host and configure your original Platform JobScheduler to work with the additional Platform LSF. Figure 3 illustrates how execution on the non-OEM host is achieved by creating a mixed cluster environment.

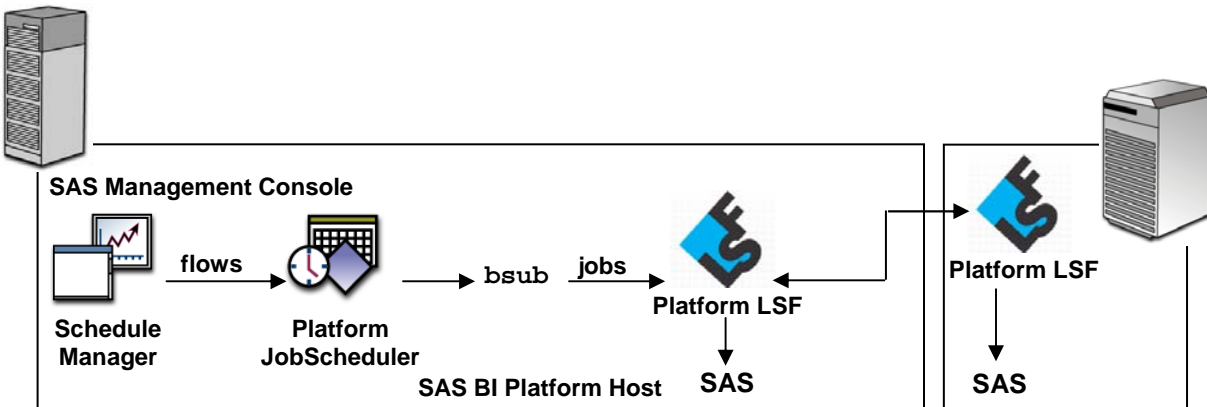


Figure 3. The Platform LSF Option

## CONFIGURING USERS AND GROUPS

To configure users and groups, it is recommended that you create a user account for administering and scheduling flows. This scheduling user account is a system account that can authenticate on all the machines in the cluster. In addition, it is made a member of the **Scheduling Admins** group and is an administrator for both Platform JobScheduler and Platform LSF. This user account should be used to schedule all production flows. A site might have multiple scheduling user accounts per department areas, or have a single, enterprise-wide scheduling user account. Whether there is one scheduling user account or many, all scheduling user accounts should be members of the **Scheduling Admins** group. By default, the scheduling user account to schedule flows make jobs run as the scheduling user account. If a particular job must run as another user account, then the scheduling administrator can set the **RunAs** value for the job on the **Advanced Properties** tab. The scheduling user account does not have the same user name that is used to run the Platform JobScheduler and Platform LSF server daemons. The user name for running the Platform JobScheduler and Platform LSF server daemons should be a local account on the machine and should be used across all machines in the cluster. For example, you would create local account **lsfadmin** to execute the server daemons, and create user account **scheduleAdmin** for scheduling all flows.

## LEVERAGING THE SAS SCHEDULING SOLUTION

The SAS Scheduling solution can address many common enterprise problems. This section describes three features that are provided by Platform JobScheduler and Platform LSF to solve common enterprise challenges in your SAS environment.

### TUNING PLATFORM LSF TO OPTIMIZE YOUR MACHINES

#### How Platform JobScheduler and Platform LSF Work

Platform JobScheduler manages job dependencies and submits jobs to Platform LSF for execution when all job dependencies are satisfied. Platform LSF does not execute the jobs immediately, but puts the jobs in the specified queue or in the default queue if no queue is specified. Platform LSF matches the requirements of the job (if defined) with the resources of the host as defined by the administrator, and decides when jobs should be picked up from the

queue and executed on a host. Several factors affect this decision, such as the load on the host and the requirements of the job.

Platform LSF waits for a configured number of dispatching intervals before sending another job to the same host. The waiting time is configured by the parameter **JOB\_ACCEPT\_INTERVAL** in the **lsb.params** file (as defined in **\$LSB\_CONFDIR/<cluster\_name>/configdir**<sup>1</sup>). The amount of time between dispatching intervals is specified by the parameter **MBD\_SLEEP\_TIME**, which is defined in the **lsb.params** file. In a default installation of Platform LSF, the **JOB\_ACCEPT\_INTERVAL** is 1 and the **MBD\_SLEEP\_TIME** is 20 (seconds). Therefore, Platform LSF dispatches one job to a particular machine and waits 20 seconds before dispatching another job to the same machine.

Typically, the default configuration is sufficient for most workload types. However, you can tune Platform LSF to optimize your machine configurations and workload types to obtain better overall performance.

## Leveraging your n-way Machine

Assume that Platform JobScheduler and Platform LSF are installed on one 16-CPU machine. There are many jobs with varying execution times from minutes to days. Despite having the 16-CPU machine, there are always jobs in the queue waiting to be executed, so you need to get the most computing out of your 16-CPU machine. All jobs are submitted to the **normal** queue. To leverage your machine, perform the following steps:

Step 1: In your **lsb.params** file, set the **JOB\_ACCEPT\_INTERVAL** parameter to **0**. This directs Platform LSF to submit more than one job on the same host during the same dispatch turn.

Step 2: Limit the number of jobs that can be executing on this 16-CPU machine by using the one-minute run queue length (**r1m**) variable. Set **r1m** for the **normal** queue to **0.7/2.0** in the **lsb.queues** file (as defined in **\$LSB\_CONFDIR/<cluster\_name>/configdir**).

```
Begin Queue
QUEUE_NAME = normal
...
r1m=0.7/2.0
...
End Queue
```

The **r1m** setting stops scheduling jobs to the host when **r1m** exceeds 0.7, and suspends jobs executing on the host when **r1m** exceeds 2.0. When **r1m** drops below 2.0, suspended jobs (if any) are resumed, and when **r1m** drops below 0.7, jobs can again be scheduled to the host. By default, the **normal** queue is assigned all hosts; therefore, this queue configuration applies to your 16-CPU machine.

Step 3: Reconfigure the cluster by issuing the **badadmin reconfig** command.

By setting the **JOB\_ACCEPT\_INTERVAL** parameter to 0 in step 1, Platform LSF dispatches multiple jobs per dispatch turn to your 16-CPU machine. This can have the effect of loading the machine with jobs quickly. Setting the **r1m** variable in step 2 tempers this effect by specifying job scheduling and suspending conditions. Several indices can be used to define load in Platform LSF. These indices can be used in place of, or in conjunction with, **r1m**. Other load indices can be found in *Administering Platform LSF*.

### A Different Step 2

Assuming the same requirements as in example 1, you can define **job slots** to limit the number of jobs executing on the 16-CPU machine. In example 1, substitute Step 2 with the following:

---

<sup>1</sup> **LSB\_CONFDIR** is the directory containing all batch configuration files (such as details about the hosts, queues, and batch parameters in your cluster). Typically, this directory is located in **<LSF\_TOP>/conf/lisbatch**, where **<LSF\_TOP>** is the location of your Platform LSF installation.

Step 2: Limit the number of jobs executing on the 16-CPU machine to be one job per CPU.

In your **lsb.hosts** file (as defined in **\$LSB\_CONFDIR/<cluster\_name>/confidir**), create a column for **MXJ** and put **!** in the **default** host line:

```

Begin Host
HOST_NAME      MXJ  rlm    pg    ls      tmp  DISPATCH_WINDOW # Keywords
default        !   ( )    ( )    ( )    ( )    ( )              # Example
End Host

```

**MXJ** is the job slot variable and **!** indicates that the value is the number of CPUs on the machine. The **default** machine indicates that all hosts should be configured with the number of **job slots** equal to the number of CPUs they have.

**Job slots** give you the flexibility to limit the number of jobs a host will execute at any time. Note that job slots do not control the number of CPUs that are being used; that is generally determined by the host's operating system.

### IMPLEMENTING SITE POLICIES

You can modify the configuration of Platform JobScheduler and Platform LSF in a SAS environment to automatically assign the proper resources to all jobs in a flow submission. This section describes changing and validating the configuration, and includes two user scenarios. This information is directed toward job-scheduling administrators and applications developers who build scheduled jobs.

### Using `esub` to Validate, Modify, or Reject Job Submissions

In the SAS Management Console (via the Schedule Manager plug-in), you can specify advanced parameters for your jobs, such as whether to receive e-mail when the job begins execution, and on which host to run the job. When submitting the job for execution to Platform LSF, Platform JobScheduler translates these advanced parameters into a batch submission (`bsub`) command.

For example, if you specify a job to be submitted to the **normal** queue, as an exclusive job, as part of the project **my\_project**, and with the number of processors to use being **2**, the corresponding `bsub` command would be:

```
% bsub -q normal -x -P my_project -n 2 sleep 10
```

The external submission (`esub`) command enables an administrator to programmatically modify the `bsub` command before submitting jobs to Platform LSF. Figure 4 illustrates a job submission flow that uses the `esub` command.

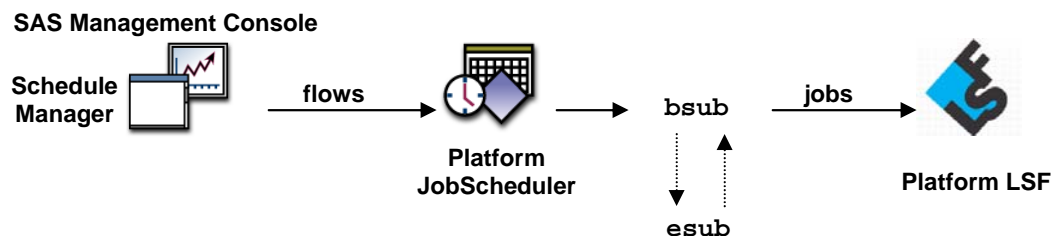


Figure 4. Batch Submission Using an `esub` Command

An `esub` command is an administrator-written executable (binary or script) that can validate, modify, or reject jobs. The `esub` command is put into the `LSF_SERVERDIR`<sup>2</sup> directory, where Platform LSF checks for its existence when a job is submitted, re-started, or modified. If Platform LSF finds an `esub` command, the command is run by Platform LSF. Whether the job is submitted, modified, or rejected depends on the logic that is built into the `esub` command.

Messages that need to be sent to the user should be directed to the standard error (`stderr`) stream, and not the standard output (`stdout`) stream.

## Understanding Environment Variables to Bridge `esub` and Platform LSF

There are four environment variables in the `esub` execution environment:

### **LSB\_SUB\_PARM\_FILE**

points to a temporary file that contains the job parameters that the `esub` command reads when the job is submitted. The submission parameters are name and value pairs on separate lines specified in the form: ***option\_name=value***.

For example, submit the following job:

```
% bsub -q normal -x -P my_project -n 2 sleep 10
```

The contents of **LSB\_SUB\_PARM\_FILE** are:

```
LSB_SUB_QUEUE="normal"  
LSB_SUB_EXCLUSIVE=Y  
LSB_SUB_PROJECT_NAME="my_project"  
LSB_SUB_NUM_PROCESSORS=2  
LSB_SUB_MAX_NUM_PROCESSORS=2  
LSB_SUB_COMMAND_LINE="sleep 10"
```

The list of all possible job parameters that are written to the **LSB\_SUB\_PARM\_FILE** can be found in the SAS white paper, *Implementing Site Policies for SAS Scheduling with Platform JobScheduler*.

### **LSB\_SUB\_ABORT\_VALUE**

specifies the exit value that `esub` should return if Platform LSF is to reject the job submission.

### **LSB\_SUB\_MODIFY\_ENVFILE**

specifies the file that `esub` should write any job environment variables changes to. The variables modified should be written to this file in the same format that is used in **LSB\_SUB\_PARM\_FILE**. The order of the variables does not matter. After `esub` runs, Platform LSF checks **LSB\_SUB\_MODIFY\_ENVFILE** for changes. If changes are found, Platform LSF applies them to the job's environment variables.

---

<sup>2</sup> **LSF\_SERVERDIR** is the machine-dependent directory that contains all the server daemon binaries, scripts, and other utilities that are shared by all hosts of the same type. Typically, this directory is located in `<LSF_TOP>/<version>/<arch>/etc`, where `<LSF_TOP>` is the location of your Platform LSF installation, `<version>` is the Platform LSF version number, and `<arch>` is the host's architecture. You will have an **LSF\_SERVERDIR** for every different machine architecture that you install, although some machine architectures are similar enough to share the same binaries and, therefore, the same **LSF\_SERVERDIR** directory.

## LSB\_SUB\_MODIFY\_FILE

specifies the file that `esub` should write any submission parameter changes to. The job options modified should be written to this file in the same format that is used in `LSB_SUB_PARM_FILE`. The order of the options does not matter. After `esub` runs, Platform LSF checks `LSB_SUB_MODIFY_FILE` for changes. If changes are found, Platform LSF applies them to the job.

### Understanding General `esub` Logic

After `esub` is issued, Platform LSF checks if the script exited with `LSB_SUB_ABORT_VALUE`. If it did, the job is rejected. Otherwise, Platform LSF applies the changes found in the `LSB_SUB_MODIFY_ENVFILE` and `LSB_SUB_MODIFY_FILE` files, if they exist.

### Rejecting, Validating, and Modifying Job Submissions

Depending on your site policies, you can reject a job. To reject a job, your `esub` command should return with `LSB_SUB_ABORT_VALUE`. If a job is rejected, the `esub` command should not write to either `LSB_SUB_MODIFY_FILE` or `LSB_SUB_MODIFY_ENVFILE`. Your `esub` command can make use of the `LSB_SUB_ABORT_VALUE` to validate a job submission by defining conditions to reject jobs.

An `esub` command can modify the job-submission parameters and the job environment variables before the job is actually submitted by writing the appropriate values to `LSB_SUB_MODIFY_FILE` and `LSB_SUB_MODIFY_ENVFILE`.

The following example illustrates a basic `esub` command that rejects, validates, and modifies the job submission based on certain criteria. In this example, **userA** can submit jobs only to the queue **queueA**; **userB** must use Bourne shell (`/bin/sh`); and **userC** has no authority to submit a job:

```
#!/bin/sh
. $LSB_SUB_PARM_FILE

# redirect stderr to stdout so echo can be used
# for error messages
exec 1>&2
#Note: In this example, queueA must exist or be added to the lsb.queue file.

# ensure userA is using the correct queue, that is, 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

# ensure userB is using the correct 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
```

## Setting the Priority of Jobs Based on the User

Assume that a job in a work environment is given a priority based on the user who initiated it. **User1** and **user2** have the same priority. **User3** and **user4** have the same priority, and their priority is higher than **user1** and **user2**. **User5** has the highest priority, and **user5** jobs should execute before any other jobs. No other users should be allowed to execute jobs in the work environment's cluster. To set job priorities, perform the following steps:

Step 1: Define three queues that have different priorities.

In your cluster's **lsb.queues** file (as defined in **\$LSB\_CONFDIR/<cluster\_name>/configdir**), set up three distinct queues. Define a different priority for each queue. For example:

```
Begin Queue
QUEUE_NAME = highpriority
PRIORITY   = 40
DESCRIPTION = for high priority users
End Queue

Begin Queue
QUEUE_NAME = medpriority
PRIORITY   = 30
DESCRIPTION = for medium priority users
End Queue

Begin Queue
QUEUE_NAME = lowpriority
PRIORITY   = 20
DESCRIPTION = for low priority users
End Queue
```

Remember to reconfigure your cluster after making these changes (**badmin reconfig**).

Step 2: Define an **esub** command to change to the appropriate queue based on the user. Place the **esub** command in your **\$LSF\_SERVERDIR** directory.

```
#!/bin/sh

# source in the parameter file so that they can be treated as environment
variables. $LSB_SUB_PARM_FILE

# redirect stdout to stderr so echo can be used for error messages
exec 1>&2

if [ "$USER" = "user1" -o "$USER" = "user2" ];
then
    # "user1" and "user2" jobs run on the low priority queue
    if [ "$LSB_SUB_QUEUE" != "lowpriority" ];
    then
        # let the user know the queue has been changed
        echo "Changing from queue <$LSB_SUB_QUEUE> to
        <lowpriority>."
        echo 'LSB_SUB_QUEUE = "lowpriority"' >
        $LSB_SUB_MODIFY_FILE
    fi
elif [ "$USER" = "user3" -o "$USER" = "user4" ];
then
    # "user3" and "user4" jobs run on the medium priority queue
    if [ "$LSB_SUB_QUEUE" != "medpriority" ];
    then
        # let the user know the queue has been changed
```

```

        echo "Changing from queue <${LSB_SUB_QUEUE} to
        <medpriority>."
        echo 'LSB_SUB_QUEUE = "medpriority"' >
        $LSB_SUB_MODIFY_FILE
    fi

elif [ "$USER" = "user5" ];
then
    # "user5" jobs run on the high priority queue
    if [ "$LSB_SUB_QUEUE" != "highpriority" ];
    then
        # let the user know the queue has been changed
        echo "Changing from queue <${LSB_SUB_QUEUE} to
        <highpriority>."
        echo 'LSB_SUB_QUEUE = "highpriority"' >
        $LSB_SUB_MODIFY_FILE
    fi

    else
        # unknown user. reject job for this reason
        echo "Unknown user $USER."
        exit $LSB_SUB_ABORT_VALUE
    fi
fi

```

### Implementing a dev/test/prod Environment

Typically, users develop, test, and then run their flows in production. Therefore, three different environments must be supported: development, testing, and production. For example, at the development stage, only **host1** should be used. For the testing and production stages, **host2**, **host3**, and **host4** can be used, but jobs that run in the production stage should always have higher priority. Because users progress at their own pace, it is up to the user to specify what stage they are at. To set up different environments, perform the following steps:

Step 1: Define three different queues to support the different stages.

In your cluster's **lsb.queues** file (as defined in **\$LSB\_CONFDIR/<cluster\_name>/configdir**), set up three queues: **dev**, **test**, **prod**; the hosts for each queue as described above; and the **prod** queue to have a higher priority than the **test** queue. For example:

```

Begin Queue
QUEUE_NAME    = prod
PRIORITY      = 35
HOSTS         = host2 host3 host4
DESCRIPTION   = for production jobs
End Queue

Begin Queue
QUEUE_NAME    = test
PRIORITY      = 20
HOSTS         = host2 host3 host4
DESCRIPTION   = for testing jobs
End Queue

Begin Queue
QUEUE_NAME    = dev
PRIORITY      = 20
HOSTS         = host1
DESCRIPTION   = for development jobs
End Queue

```

Remember to reconfigure your cluster after making these changes (**badadmin reconfig**).

Step 2: Define an `esub` command to detect which stage the user is at and to submit the jobs to the appropriate queue. Use a specific environment variable (for example, `USER_STAGE`) with specific stage keywords that map back to the queue names `dev`, `test`, and `prod`.

```
#!/bin/sh

# source in the parameter file so that they can be treated as environment
variables. $LSB_SUB_PARM_FILE

# redirect stdout to stderr so echo can be used for error messages
exec 1>&2

if [ "$USER_STAGE" = "dev" -o "$USER_STAGE" = "test" -o "$USER_STAGE" = "prod" ];
then
    # submit the job to the appropriate queue as specified by the user
    echo "Executing job in the $USER_STAGE environment."
    echo "LSB_SUB_QUEUE = \"$USER_STAGE\" " > $LSB_SUB_MODIFY_FILE
else
    # stage is not specified or unknown; assume production job
    echo "Executing job in the prod environment."
    echo 'LSB_SUB_QUEUE = "prod"' > $LSB_SUB_MODIFY_FILE
fi
```

Place the `esub` command in your `$LSF_SERVERDIR` directory.

Step 3: Tell users to change their login scripts to set the `USER_STAGE` environment variable to the appropriate stage. For example, if a user's login shell is C shell and the user is in the development stage, this user should add the line `setenv USER_STAGE dev` to the `.cshrc` file.

**Note:** When Platform JobScheduler submits the batch job to Platform LSF on behalf of a user, Platform JobScheduler effectively logs on as that user (which picks up the user's environment), and then submits the job. By doing this, the job submission behaves as if the user logged on to the Platform JobScheduler server host and submitted the job to Platform LSF.

## IMPLEMENTING FLEXIBILITY IN YOUR FLOWS AND JOBS

Platform JobScheduler provides substitution capabilities within your job through the use of variables. When Platform JobScheduler encounters a variable, it substitutes the variable with its current value. For example, you can use variables as part of or as all of a filename to make filenames flexible, or you can use variables to pass arguments to and from scripts. You can export the value of a variable to one or more jobs in a flow (making the variable local to the flow), or to other flows that are currently running on the same Platform JobScheduler server (making the variable global to all flows).

### Using User Variables within a Flow Definition

There are two types of variables that you can set: local variables, which are available to the jobs or events within the current flow; and global variables, which are available to all the flows within the Platform JobScheduler server.

You can set the value of one or more variables in the following ways:

- Set the variable value within your job. This approach allows you to dynamically define and modify variable values during the execution of your flow.
- Trigger a flow to run with user variables through the Platform JobScheduler Flow Manager<sup>3</sup>. This approach allows you to statically define your variable values when the flow is initially triggered for execution.

---

<sup>3</sup> The Flow Manager application is provided with Platform JobScheduler. Flow Manager enables you to monitor the progress and manage the execution of your flows.

To set variable values within your job, see the “Defining Flows to Process Today’s Data” section.

To trigger a flow with user variables through the Flow Manager, perform the following steps:

1. In the tree view of the Flow Manager, expand the tree until you see the flow definition you want to trigger.
2. Right-click the flow definition, select **Trigger**, and then select **With Variables**. The **Pass Variables to Flow** dialog box appears.
3. Specify the parameters to pass in the following format:  
`variable=value;variable=value...`
4. Click **OK**. A flow is created and run.

Once a local or global variable has been defined, you can use the variable in the following job fields (available in Schedule Manager from the job’s properties using the **Advanced...** button on the **Scheduling Details** tab):

- Job Definition, General tab: Name, Command to run, Input file, Output file, Error file
- Job Definition, File Transfer tab: Local path including name, File on execution host
- File Event Definition dialog: File name

To use a variable, perform the following steps:

1. Ensure that a value for the variable is set, either within the current flow, or as a global variable within the Flow Manager server, prior to the point in the flow where the value is required.
2. In the appropriate input field, specify the variable in the format `#{variable}`.

More information about user variables (including built-in variables) can be found in *Using Platform JobScheduler*.

## Defining Flows to Process Today’s Data

Suppose you want to define a flow that processes data based on the current day’s data files. You might want to process data based on previous data files. The data files are in the format `<yyyy><mm><dd>.dat`. To use variables to define a flow, perform the following steps:

**Important:** Before you can use variables in your flow, you or your administrator must configure one or more queues in Platform LSF to support setting user variables. You can either choose an existing queue or create a new queue. In your cluster’s `lsb.queues` file (as defined in `$LSB_CONFDIR/<cluster_name>/configdir`), add the following **JOB\_STARTER** line to the queue(s) you want to support setting user variables:

```
JOB_STARTER = <JS_TOP>/<JS_VERSION>/bin/jsstarter  
  
For example:  
Begin Queue  
QUEUE_NAME = uservars  
DESCRIPTION = for jobs that need to set user variables  
# Note: <JS_TOP> is where you installed Platform JobScheduler  
#       <JS_VERSION> is your JobScheduler version  
JOB_STARTER = <JS_TOP>/<JS_VERSION>/bin/jsstarter  
End Queue
```

Step 1: Write a script that sets the **DATE** variable for the flow.

In the Bourne shell, the script would look like the following:

```
getdate.sh
-----
#!/bin/sh

# if DATE has already been defined, don't set the DATE variable
if [ "$1" = "#{DATE}" ]
    # the date has not been set; set it to today's date
    todays_datevar=`date +%Y%m%d`
    JS_FLOW_VARIABLE_LIST="DATE=$todays_datevar"
    export JS_FLOW_VARIABLE_LIST
elif
    # do nothing, the date has already been set
fi
```

Note that the script communicates the **DATE** variable with Platform JobScheduler through the **JS\_FLOW\_VARIABLE\_LIST** environment variable. If you want to make **DATE** a global variable, replace **JS\_FLOW\_VARIABLE\_LIST** with **JS\_GLOBAL\_VARIABLE\_LIST**.

Step 2: Create a job at the beginning of the flow that executes the script. The job would have the command line **getdate.sh #{DATE}** and must be submitted to a queue that supports setting user variables.

When Platform JobScheduler encounters an undefined variable, no substitution is made. Thus, if **DATE** is not defined, then **getdate.sh #{DATE}** is the command that gets executed.

Step 3: Define other jobs that require the **DATE** variable to be dependent on this job. For example, the command line of a dependent job might be **processdata.sh #{DATE}.dat**. This job can be submitted to any queue.

As a result, in Flow Manager, if you trigger the flow with no variables, the first job sets the **DATE** variable to today's date. If you trigger the flow that sets the **DATE** variable (as described in the "Using User Variables within a Flow Definition" section), the first job leaves the **DATE** variable alone. In either case, the subsequent jobs use the value that is specified by **DATE**.

## CONCLUSION

SAS and Platform Computing have partnered to provide an integrated, enterprise job scheduler that is specifically designed to manage your complex job flows more efficiently. An overview of the architecture and a description of the supported operating systems and configuration were described that can be leveraged to maximize your existing IT infrastructure. Three broad scheduling topics were introduced to resolve common user scenarios:

- tuning your cluster to maximize the utilization of n-way machines
- automating the implementation of site policies to automatically set job priorities based on the user and implement a dev/test/prod environment
- implementing flexibility in your flows to create a flow that processes today's data files

The topics and examples provided in the paper are a small representation of the features available and how they can be applied. A few commonly encountered use cases and customer scenarios are a fraction of all use cases and customer scenarios. More information is available from the SAS Customer Support Web site (<http://support.sas.com>) and the Platform JobScheduler for SAS Web site (<http://www.platform.com/products/JSSAS>), and you are encouraged to visit them.

## REFERENCES

Tran, Allen, and Williams, Randy. 2004. "Implementing Site Policies for SAS Scheduling with Platform JobScheduler". Available <http://support.sas.com/documentation/whitepaper/technical/JobScheduler.pdf>.

Platform Computing Corporation. 2003. *Administering Platform LSF*. Markham, Ontario: Platform Computing Corporation.

Platform Computing Corporation. 2003. *Using Platform JobScheduler*. Markham, Ontario: Platform Computing Corporation.

## ACKNOWLEDGEMENTS

The authors would like to thank Cheryl Doninger, Amy Wolfe, Diane Hatcher, Qingda Wang, and Kathy Walch for reviewing contents of this paper and making suggestions to improve it.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Allen Tran  
Platform Computing Corporation  
Markham, Ontario  
Work Phone: 905-948-4169  
[atran@platform.com](mailto:atran@platform.com)

Randy Williams  
SAS Institute Inc.  
SAS Campus Drive  
Cary, NC 27513  
Work Phone: 919-531-6865  
[Randy.Williams@sas.com](mailto:Randy.Williams@sas.com)

Alan Wong  
Platform Computing Corporation  
Markham, Ontario  
Work Phone: 905-948-4308  
[awong@platform.com](mailto:awong@platform.com)

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

Copyright © 2005 Platform Computing Corporation. ®™ Trademarks of Platform Computing Corporation. All other logo and product names are the trademarks of their respective owners, errors and omissions excepted.