Implementing Site Policies for SAS® Scheduling with Platform JobScheduler

The Power to Know®

A SAS White Paper

This paper was co-authored by Allen Tran, Software Development Manager, Platform Computing Inc. and Randy Williams, Principle Systems Developer, SAS Institute.

Table of Contents

Introduction

There are many factors that can affect job scheduling efficiency within an enterprise. For example, usually, the people who create the jobs are not the people who administer and control the resource usage. In addition, the jobs that are created might have different priorities. Although most of the people who create the jobs want them to run on the biggest and fastest machines and have the highest priorities, this might not be in the best interest of the enterprise. Relying on the job-scheduling administrator to coordinate the jobs and assign the resources can, at best, be tedious and error-prone. Many enterprises find it difficult to overcome these challenges because policies have not been implemented for scheduled job submission to help control the resources that are used by various individuals or groups. These enterprises need a way to configure their systems to automatically set the resources and control job execution based on known criteria.

SAS Institute has partnered with Platform Computing Inc. to provide a robust scheduling solution. This paper explains how to modify the configuration of Platform JobScheduler and Platform LSF, both offered by Platform Computing, in a SAS environment to automatically assign the proper resource to all jobs in a flow submission. How to change and validate the configuration is also discussed, and two common user scenarios show how it all works.

This paper is written for job-scheduling administrators and applications developers who build scheduled jobs. It is assumed that the reader has a basic understanding of scheduling servers and how to use the SAS Management Console Schedule Manager plug-in.

The SAS Scheduling Solution

An effective scheduling solution consists of three major groups of components that enable an enterprise to configure and set up their environment, create jobs and schedule flows, and then execute the flows. The following components are required in order to take advantage of the scheduling capabilities of the SAS Scheduling Solution.

- SAS configuration is handled by SAS Management Console and stored in the SAS Metadata Server. Metadata for servers, flows, and jobs that are deployed from SAS applications is captured in the metadata server. The scheduling and batch servers are defined in the Server Manager, which is a SAS Management Console plug-in. The scheduling servers are third-party software applications; the 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.

- Flow creation and management is handled by the Schedule Manager plug-in component in SAS Management Console, along with Scheduling Integrated SAS Applications such as SAS ETL Studio and SAS Marketing Automation Campaign Manager.

  A *flow* contains one or more jobs that are deployed for scheduling by the Scheduling Integrated SAS Applications. Flows are created and maintained in Schedule Manager. A deployed job is associated to a batch server.

- Flow execution is handled by scheduling servers. In SAS 9.1, the only scheduling server that is available is Platform JobScheduler, which includes Platform LSF as part of the installation and configuration of the scheduling server.

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. A batch submission flow using the batch submission (`bsub`) command is shown in Figure 1.

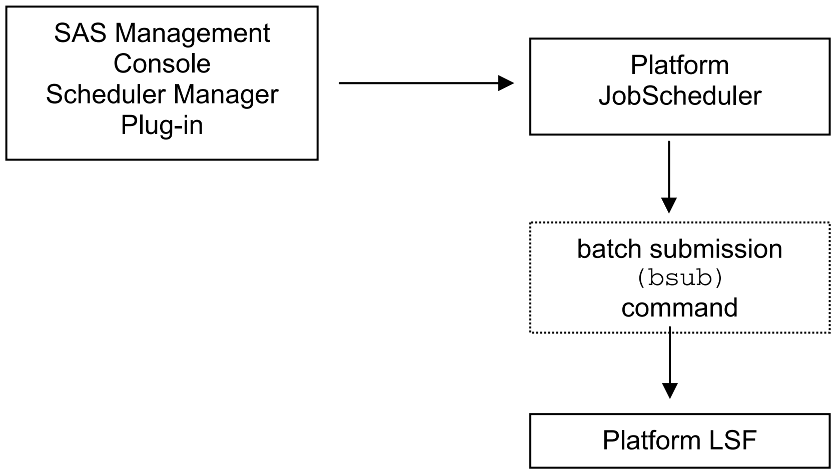
SAS Management Console Scheduler Manager Plug-in
Platform JobScheduler
batch submission (bsub) command
Platform LSF

Figure 1. Batch Submission Flow

The SAS Management Console Schedule Manager plug-in gets metadata about the flow from the 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. (Dependencies are the criteria that must be met so that the job will run.) After dependencies are defined for each job, the jobs in the flow can be executed. Platform JobScheduler executes the jobs in a flow by issuing a `bsub` command to Platform LSF.

The Platform LSF configuration contains a cluster of machines that represents a virtual machine. Within a cluster there can be multiple queues, and each queue has its own associated resources and properties. Before you can validate, modify, or reject job submissions, you need to develop the job-scheduling criteria that will be used at your site. Developing the criteria involves understanding the current Platform LSF configuration, knowing what needs to be accomplished, and identifying the available resources. The two examples given in the “Examples” section in this paper explain how to set the priority of a job based on the user who submits the job, and how to channel the job to the correct machine based on the stage of development.

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

The external submission (`esub`) command, enables an administrator to programmatically modify the `bsub` command before submitting jobs to Platform LSF. A job submission flow using the `esub` command is shown in Figure 2.

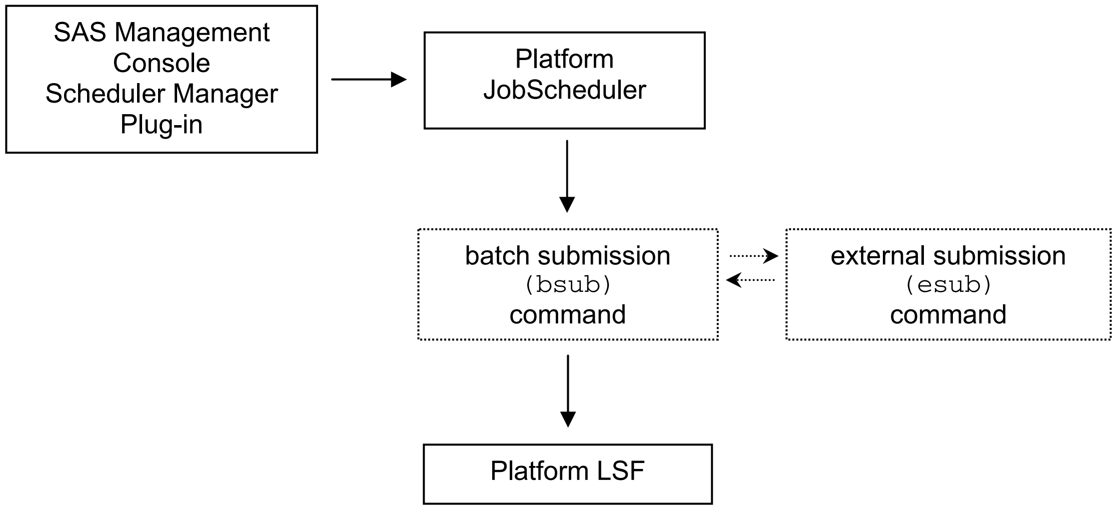
SAS Management Console Scheduler Manager Plug-in
Platform JobScheduler
batch submission (bsub) command
external submission (esub) command
Platform LSF

Figure 2. Batch Submission Using an `esub` Command

An `esub` command is an administrator-written executable (binary or script) that can be used to validate, modify, or reject jobs. The `esub` command is put into LSF_SERVERDIR[1] 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's built into the `esub` command.

Any 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

Platform LSF provides the following 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*. (Table 1 lists and describes the supported options.)

For example, if a user submits the following job:

```
% bsub -q normal -x -P my_project -R "r1m rusage[dummy=1]" -n 90 sleep 10
```

[1] 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. Usually, LSF_SERVERDIR 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. For example, if you installed Platform LSF 5.1 on a Solaris 7 64-bit host under "/usr/share/lsf", LSF_SERVERDIR would be "/usr/share/lsf/5.1/sparc-sol7-64/etc".
Note: You would 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.

then the contents of the LSB_SUB_PARM_FILE will be:

```
LSB_SUB_QUEUE="normal"
LSB_SUB_EXCLUSIVE=Y
LSB_SUB_PROJECT_NAME="my_project"
LSB_SUB_RES_REQ="r1m rusage[dummy=1]"
LSB_SUB_NUM_PROCESSORS=90
LSB_SUB_MAX_NUM_PROCESSORS=90
LSB_SUB_COMMAND_LINE="sleep 10"
```

OPTION	DESCRIPTION
LSB_SUB_COMMAND_LINE	Job command
LSB_SUB_ERR_FILE	Standard error filename
LSB_SUB_EXCLUSIVE	Y specifies exclusive execution
LSB_SUB_HOST_SPEC	Host specifier
LSB_SUB_HOSTS	List of execution host names
LSB_SUB_IN_FILE	Standard input filename
LSB_SUB_LOGIN_SHELL	Login shell
LSB_SUB_MAIL_USER	E-mail address used by Platform LSF for sending job e-mail
LSB_SUB_MAX_NUM_PROCESSORS	Maximum number of processors requested
LSB_SUB_NOTIFY_BEGIN	Y specifies e-mail notification when job begins
LSB_SUB_NOTIFY_END	Y specifies e-mail notification when job ends
LSB_SUB_NUM_PROCESSORS	Minimum number of processors requested
LSB_SUB_OTHER_FILES	Always SUB_RESET if defined to indicate that a `bmod` command is being issued to re-set the number of files to be transferred **Note:** For more information about the `bmod` command, see *Platform LSF Reference*.

LSB_SUB_OTHER_FILES_	A value is the specified file transfer expression. For example, for `bsub` -f "a > b" -f "c < d", the following is defined:
	LSB_SUB_OTHER_FILES_0="a > b"
	LSB_SUB_OTHER_FILES_1="c < d"

LSB_SUB_OUT_FILE	Standard output filename
LSB_SUB_PRE_EXEC	Pre-execution command

continued

Table 1. Supported Option Names for Job Parameters

OPTION	DESCRIPTION
LSB_SUB_PROJECT_NAME	Project name
LSB_SUB_QUEUE	Submission queue name
LSB_SUB_RERUNNABLE	Y specifies the job is rerunnable
LSB_SUB_RES_REQ	Resource requirement string
LSB_SUB_RLIMIT_CORE	Core file-size limit
LSB_SUB_RLIMIT_CPU	CPU limit
LSB_SUB_RLIMIT_DATA	Data-size limit
LSB_SUB_RLIMIT_FSIZE	File-size limit
LSB_SUB_RLIMIT_RSS	Resident-size limit
LSB_SUB_RLIMIT_RUN	Wall-clock run limit
LSB_SUB_RLIMIT_STACK	Stack-size limit
LSB_SUB_USER_GROUP	User group name

Table 1 (continued). Supported Option Names for Job Parameters

LSB_SUB_ABORT_VALUE

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

LSB_SUB_MODIFY_ENVFILE

specifies the file that the `esub` command should write any job environment variables changes to. The variables that will be 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.

LSB_SUB_MODIFY_FILE

specifies the file that the `esub` command should write any submission parameter changes to. The job options that will be 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 the `esub` command is issued, Platform LSF runs the following checks (see Figure 3).

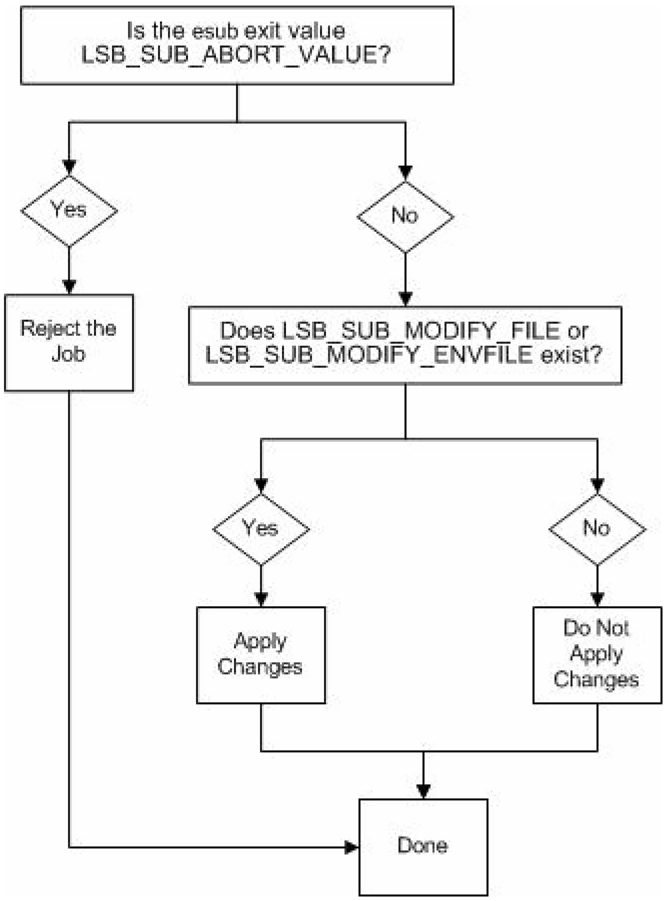
Is the esub exit value LSB_SUB_ABORT_VALUE?
Yes
No
Reject the Job
Does LSB_SUB_MODIFY_FILE or LSB_SUB_MODIFY_ENVFILE exist?
Yes
No
Apply Changes
Do Not Apply Changes
Done

Figure 3. Platform LSF `esub` Command Logic Flow

Rejecting Jobs

Depending on your site policies, you can choose to 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.

For example, the following Bourne shell `esub` command rejects all job submissions by returning LSB_SUB_ABORT_VALUE.

```
#!/bin/sh

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

# reject the submission
    echo "LSF is rejecting your job submission..."
    exit $LSB_SUB_ABORT_VALUE
```

Validating Job Submission Parameters

Validation can be used to support project-based accounting. This means that the user may request that the resources that are used by a job be charged to a specific project. Because projects are associated with a job at the time the job is submitted, Platform LSF accepts any arbitrary string for a project name. An `esub` command can be used to ensure that only valid projects are entered, and that the user is authorized to charge to that project.

For example, the following Bourne shell `esub` command validates the job-submission parameters.

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

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

# check valid projects
if [ “$LSB_SUB_PROJECT_NAME” != "proj1" -a
“$LSB_SUB_PROJECT_NAME” != "proj2" ];
then
  echo "Incorrect project name specified."
  exit $LSB_SUB_ABORT_VALUE
fi

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
```

Modifying Job-Submission Parameters

An `esub` command can be used to modify the job-submission parameters and the job environment variables before the job is actually submitted. The next example writes modifications to LSB_SUB_MODIFY_FILE for the parameters LSB_SUB_QUEUE and SHELL.

In this example, user “userA“ can submit jobs only to the queue “queueA”; user “userB” must use Bourne shell (/bin/sh); and user “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
```

Examples

Example 1 Setting the Priority of Jobs Based on the User

In this example, the jobs in the work environment are given different priorities based on the user who initiates the job. User1 and user2 have the same level of priority. User3 and user4 have the same level of priority, but their level of priority is higher than that of user1 and user2. User5 has the highest priority, and this user's jobs should execute before any other jobs. No other users should be allowed to execute jobs on the cluster.

Step 1. Define three queues that have different priorities.

In your cluster's lsb.queues file (defined in $LSB_CONFDIR/<*cluster_name*>/configdir),[2] set up three distinct queues. Define a different level of 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.[1]

```
#!/bin/sh

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

[2] LSB_CONFDIR is the directory that contains all the batch configuration files (such as details about the hosts, queues, and batch parameters in your cluster). Usually, this directory is located in "<*LSF_TOP*>/conf/lsbatch" where <*LSF_TOP*> is the location of your Platform LSF installation.

```
# 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
```

Example 2 Implementing a `dev/test/prod` Environment

Usually, 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 may be used, but jobs that run in the production stage should always have higher priority. Because users progress at their own pace, it’s up to the user to specify what stage they are at.

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

In your cluster's lsb.queues file (defined in $LSB_CONFDIR/<*cluster_name*>/configdir)[2], you must set up

1. three queues: "dev", "test", "prod"

2. the hosts for each queue as described above

3. 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 (badmin reconfig).

Step 2. Define an `esub` command to detect which stage the user is at and 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.[1]

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 "logs in" 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 into the Platform JobScheduler server host and submitted the job to Platform LSF.

Summary

A site policy for scheduled job submission and use of resources varies from enterprise to enterprise. The person who submits the jobs for scheduling might not be aware of the overall use of resources and is usually not equipped to determine the start time, the machine, and the external dependencies on input data. The IT Department is often understaffed and is looking for ways to automate the workload and reduce manual intervention. Manually scheduling resources is costly to the enterprise in terms of increased errors and increased workload for the IT staff.

Automation of site policies in scheduling is an integral part of SAS Scheduling with Platform Computing scheduling services. The examples in this white paper show how to configure Platform JobScheduler and Platform LSF in a SAS environment to automatically assign the proper resource to all jobs in a flow submission. The examples and information in this paper can help you leverage the flexibility of the SAS Scheduling Solution to your specific policies.