Paper BI222012

SAS® Grid: Grid Scheduling Policy and Resource Allocation

Adam H. Diaz, IBM Platform Computing, Research Triangle Park, NC

ABSTRACT

This paper will discuss at a high level some of the options for tuning IBM Platform LSF® which is the technology that powers SAS Grid. This will include some options for taking SAS Grid "beyond the defaults" with discussion about less commonly used scheduling options. The paper will deal with SAS Grid for SAS 9.3, Platform Suite for SAS 6.1 and Platform LSF 7.06. The audience should have a working knowledge of SAS infrastructure and a basic knowledge of Platform LSF.

INTRODUCTION

SAS Grid is a wonderful merger of SAS technology and world class grid management and workload technology created by Platform Computing. Platform LSF powers many of the world's largest and most complex scientific and analytical High Performance Computing sites in the world as well as some of the largest SAS using enterprise globally. This powerful technology has many option and creative ways to configure behavior that many times the process is more are than computer science. The merger of experience in workload management and a consultative approach to various needs of industry verticals are required to obtain a configuration that is appropriate for an individual site. What follows is a description of options less known and utilized by many SAS users. Often users and administrators interact with SAS Grid via the available graphical user interface called Platform RTM for SAS. The far more flexible method involves the use of the command line unleash the power of LSF. This will be an introduction to some of the important files, resources and scheduling policies used to move beyond a default installation with regard to job placement on a grid.

WHAT IS SAS GRID?

SAS Grid is a product from SAS that "grid enables" SAS deployments. This means that this product enables an enterprise infrastructure to distribute the compute portion of your SAS work to a networked collection of computers. While SAS Grid has a variety of methods for job submission, the user experience from an interface perspective for most products remains very similar to a non grid infrastructure. For Enterprise Guide, Data Integration Studio or Enterprise Miner the change in workflow is one of some initial configuration during product setup after which the use of the product is very similar. SAS Grid then is a way to allow many users access to a collection of compute resources and SAS applications while providing workload management, high availability and performance benefits. SAS Grid also provides a highly valued ease of expansion due to its scalable architecture. This feature allows businesses to easily expand their investment in the value of the analytics produced by their SAS code in an affordable and metered way. This single benefit is well received by system administrators and CFOs alike. Expanding a SAS Grid doesn't require reinstallation of SAS but rather an incremental investment in additional hardware and software resources.

WHAT IS PLATFORM SUITE FOR SAS?

Platform Suite for SAS (aka, PSS) is a collection of software products from Platform Computing used as the core scheduling and process management tools that power the larger SAS Grid offering. PSS is the result of a long time partnership between SAS and Platform extending back over 10 years. PSS is current at version 6.1 with SAS 9.3. The products in Platform Suite for SAS include Platform LSF, Platform Process Manager, Platform MPI and Grid Management Service.

Platform LSF provides a robust scheduling and workload management capability across the compute nodes in the cluster. Platform Process Manager provides the ability to schedule workflows across the cluster. Many times Process manager is considered "the scheduler" in that it logically provides the ability to submit a workflow to the grid. In the strictest sense Process Manager is used to create a workflow and define the dependencies around that workflow (including items like launching time, workflow logic and failure exception handling). Platform LSF technically is eventually passed the defined workflow from Platform Process manager for execution on the nodes. In this scenario Platform Process Manager is a workflow engine while Platform LSF is actually the scheduler. This is simply a case of overloaded terminology.

Platform MPI is an implementation of MPI or Message Passing Interface originally created by a now acquired company called Scali. This technology was eventually purchased and improved by Platform. Platform MPI is a core technology used in a number of alternate technology offerings not used by SAS at this time. While Platform MPI is installed along with PSS, it is technically not used at this time but may be leveraged in the future to enable parallelization of underlying technology in SAS Grid or use by advanced users who may be creating their own SAS PROCs including calls to C level code in SAS programming.

Finally Grid Management Service is a monitoring daemon that allows SAS users access to the information provided by Platform LSF including job status and load information within a SAS context. This includes the passing of information to the SAS Management Console to the Schedule Manager plug-in installed with SAS Grid.

SAS Grid then includes PSS but also other components including the licensing and additional software necessary to connect a user to grid itself. This includes the SAS Management console Schedule Manager plug-in, SASGSUB a command line submission client as well as the code included in individual SAS clients such as SAS Enterprise Miner, SAS Data Integration Studio and SAS Enterprise Guide that include both configuration options and the ability to submit jobs to the grid via SAS MetaData Server. This work is the culmination of both client and combined working experience with SAS and Grids by both SAS R&D as well Platform Computing for last decade.

WHAT ARE THE BASIC COMPONENTS OF PLATFORM LSF?

Platform LSF as the core component of the Platform Suite for SAS is the piece of software included in SAS Grid that is the traffic cop for job placement. LSF is consulted in a variety of scenarios either to simply provide information to SAS via API on the compute node with the lowest load for workspace placement (workload balancing) or placement of full LSF jobs in form of a grid session. There is an additional component for controlling your SAS Grid called Platform RTM for SAS. This is a graphical interface for controlling most options related to your grid including nodes, users and scheduling policy to name a few. This system is available for use with SAS Grid as a separate download on the SAS website. The balance of this paper will not deal with Platform RTM for SAS but rather the command line manipulation of the Platform LSF system for configuration of resources and daemons that control the system as well as the scheduling policy central to the actual placement of jobs on nodes.

There are a number of basic daemons in use when you look at LSF itself. In the context of the overall SAS Grid architecture these daemons only run on the Grid Control Server (aka the head node) and the Grid nodes. The exception is the installation of what is called the LSF client on the Grid Client node. In this case there are no daemons running per se but the binaries for the client utility are installed to allow calls via the command line to LSF daemons installed on the Grid Control Server.

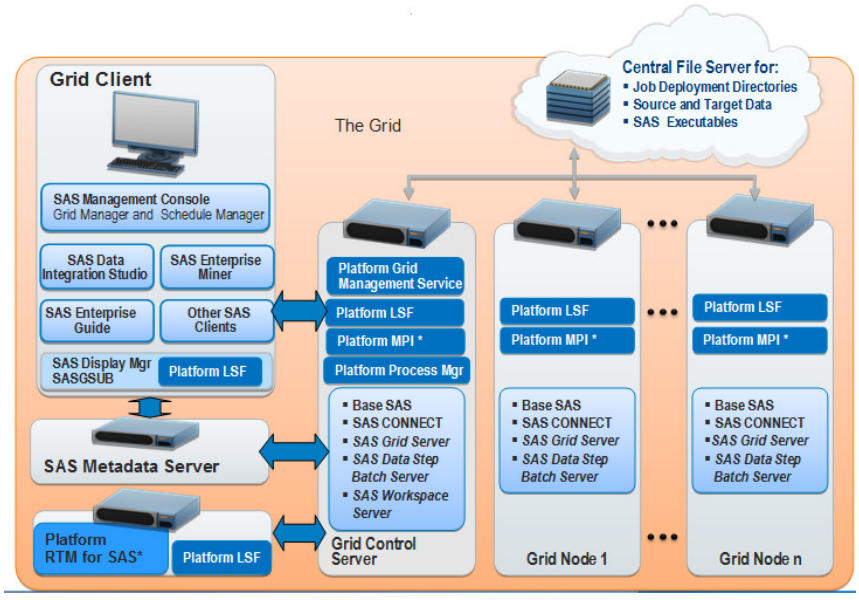


Figure 1 SAS Grid Architecture

When considering the daemons that power LSF there are two important use cases to consider. They both revolve around the use of a very specific LSF feature called EGO which stands for Enterprise Grid Orchestrator. In short EGO is a way to ensure that the services that run your grid are available. If they become unavailable for any reason

EGO can setup to restart them where they should reside normally and if not to an alternate failover location. The implication here is that different daemons run LSF when EGO is enabled. Furthermore when you want to understand not only how to manipulate the LSF options at a command line it is important to have a high level understand of what is running and how that maps to files used to control those daemons and their options.

BASIC DAEMONS IN USE WITHOUT EGO

- mbatchd –Master Batch Daemon (MBD) on LSF Master; a single running daemon per cluster; running on the Grid Control Server; understands state of all jobs and communicates load information to LIM; receives job dispatch requests from mbschd and pushes jobs to sbatchd.
- mbschd – Master scheduler will run on LSF Master; a single running daemon per cluster; running on the Grid Control Server; receives resource info from mbatchd; makes scheduling decisions and issues scheduling requests to mbatchd
- sbatchd – Slave Batch Daemon, managed by EGOSC, runs on all nodes in the cluster; receives and manages job requests from the mbatchd (aka MBD).
- res – Remote Execution Server, managed by EGOSC, runs on every node in the cluster unless it is disabled. Provides the ability to directly run interactive tasks.

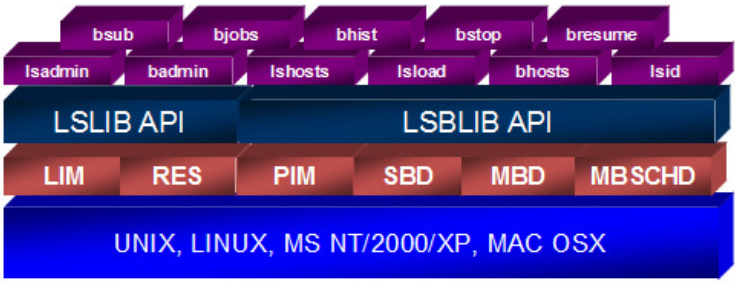


Figure 2 Structure of LSF without EGO

If Ego is enabled:

- LSF's **sbatchd** and **res** daemons are maintained by EGOSC
- LIM and PIM are part of the EGO structure

EGO DAEMONS

- VEMKD – Virtual Execution Machine Kernel Daemon; runs on the master host (Grid Control Server) when EGO is enabled. Functions as a Resource Broker; LSF mbatchd is still the workload manager.
- PEM – Process Execution Manager; started by LIM on every node; Runs to control activities as directed by VEMKD. Starts sbatchd and RES as EGO services.
- EGOSC – EGO Service Controller – the main EGO daemon; provides daemon fault tolerance.
- LIM – Load Information Manager – runs on every node in the cluster to understand the load in terms of resources being placed on that node.

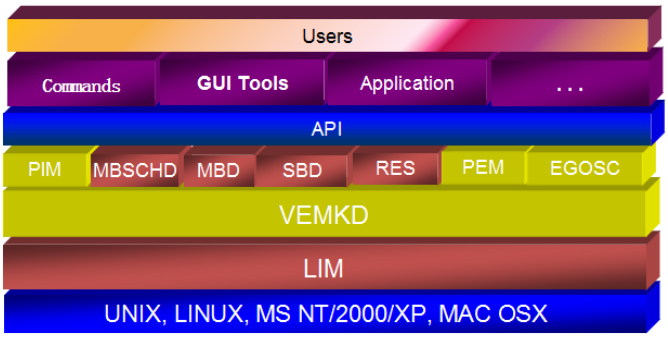


Figure 3 Structure of LSF with EGO

- PIM – Process Information Manager – runs on every node in the cluster to understand and manage the processes running on the server

IMPORTANT FILES AND LOCATIONS

There are a number of files that control not only the various daemons that run LSF but also the batch system that is contained within it. LSF stands for load sharing facility. Its original job was to understand the load running on a system (LSLIB API). The ability to then run jobs in many creative ways is a part of "batch" system (LSBLIB API). That is why in Figure 2 above LIM and RES are considered part of the LSLIB API and the balance of the daemons are listed under the LSBLIB API. LSBLIB API is the batch portion of the system that actually places jobs on nodes, runs them and returns the results. Why is this important? It is directly related to where you find the files to configure those systems in the file structure underlying LSF. As shown in Figure 4 below, the LSF_ENVDIR directory contains the files for configuring the overall cluster environment including EGO, LIM and some extent MBD and mbsched. The balance of the configuration files control the MBD behavior and are located below the LSF_ENVDIR. There are additional "lsb" files available and not all of them are listed here.

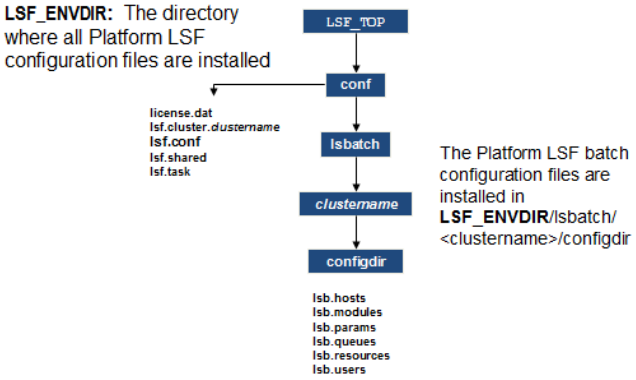


Figure 4 Distilled view of LSF file structure with important configuration file locations highlighted.

FILES THAT AFFECT THE LIM, MBD AND MBSCHD:

- lsf.conf – created at installation time; defines LSF environment; some EGO parameters allow if EGO is disabled.
- ego.conf – created at installation time; defined EGO behavior; no LSF parameters allowed.

FILES THAT AFFECT LIM:

- lsf.shared – common definitions share by all hosts; lists of cluster names; host types and models and special resources.
- lsf.cluster.<clustername> - defines options for a specific cluster name in 4 sections ClusterAdmins, Host, Parameters and ResourceMap.

FILES THAT AFFECT MBD

- lsb.params – general parameters used by MBD
- lsb.hosts – host related configuration information
- lsb.users – defines user groups and hierarchical fairshare for users and groups
- lsb.queues – defines queues in LSF
- lsb.resources – optional file concerning resources, limits and allocation
- lsb.modules – optional file concerning the information for the LSF scheduler module- "Plugin Modules"
- lsb.applications – defines application profiles for common parameters used in applications

HOW ARE RESOURCES DEFINED AND CONTROLLED?

In LSF there are variety of strategies for properties of nodes and how jobs land on those nodes. In LSF terminology those properties are called resources. This is an area of LSF that could easily consume an entire chapter in a full textbook. What follows is an extremely brief discussion of what types of resources you can control in the files we discussed in the previous section.

Resources are classified with four basic characteristics including a value, an assignment, a scope and finally a definition. These four characteristics together provide a picture of what a resource is and how it is used. A resource value includes a number of types of value including numeric, string or Boolean values. The assignment classification refers to type of change a resource might undergo including dynamic or static. Some resources such as the hostname may never change and would be classified as static whereas a resource like swap space maybe fluctuating rapidly based upon the workload. The scope of a resource deals with the level of sharing a resource undergoes. Many resources cannot be shared between nodes such as a CPU. A shared resource on the other hand would be owned collectively by the entire cluster. Some great examples include floating license for software running on the grid. Finally a resource can also be characterized by what is called classification. This means a resource can be termed "built-in" such as the amount of memory on a node or external. The externally defined resource are of particular interest due to the fact this resource type allows one to extend the capability of the system without major changes or requests to developers for source code changes. This allows for extension of LSF to include resources that might be most important to SAS users like IO or that all important amount of temporary space available on a shared directory.

Resource	Value Type	Assignment	Definition	Scope
Memory	Numeric	Dynamic	Built-in	Host-based
Maximum memory	Numeric	Static	Built-in	Host-based
Tape drive	Boolean	Static	Site-defined	Shared
Host type	String	Static	Built-in	Host-based
Host status	String	Dynamic	Built-in	Host-based
Computeserver	Boolean	Static	Site-defined	Host-based
verilog	Numeric	Static	Site-defined	Shared
verilog	Numeric	Dynamic	Site-defined	Shared

Figure 5 Examples of Resources

DEFINING AN ELIM

A very common task in configuring LSF is defining a resource that is not included by default. This is called a site-defined resource. A common way to do this involved editing the `Resource` section of the `lsb.shared` file as well as the `ResourceMap` section of the `lsf.cluster.<clustername>` file. If this resource is static, no other file level changes are necessary. If the resource is dynamic, there is a requirement to write a script called an "ELIM" or extended LIM. This is simply a script that performs the actions necessary to collect the required system information and returns the result in a formatted output that LSF can understand. It is common to have this script do a variety of shell level commands. This script, following a common naming convention, is placed in the LSF_SERVERDIR (typically LSF_TOP/version/platform/etc) on the appropriate hosts. Regardless of the static or dynamic nature after editing the appropriate file to create the resource a "reconfigure" of the cluster is required. This is a set of administrative commands to cause the required configuration files to be re read (usually involving `lsadmin reconfig` and `badmin reconfig`). To verify the collection of the dynamic resource that was created by an elim one would use `lsload` (see bold columns below). This again allows for the extension of LSF without major source code changes.

`lsload -l`

```
HOST_NAME status r15s r1m r15m ut pg  io ls it  tmp  swp  mem lic_X simpton rtprc
training1   ok   0.1  0.2 0.1  7% 0.8  1  1  0 412M 414M 152M  15       3     56
training5   ok   1.2  3.2 2.6 54% 1.5  2  8  0 183M  74M  56M  15       -     21
training8   ok   1.4  0.8 0.2 40% 1.3  3 10  0 232M 109M  45M  15       5     49
training2   ok   2.0  1.9 2.4 29% 2.1  5  5  0 209M  92M  47M  15       -     18
training3   ok   2.6  1.1 1.2 23% 3.4  3  2  0  80M  90M  92M  15       5     23
```

This resource can then be scheduled against in LSF. This means that one could submit jobs using this resource as a criterion for node placement. In LSF speak one would pass options to the bsub command such as:

```
bsub -R "rusage[lic_X=1:duration=1]" app_X
```

In SAS however the ability to pass options to LSF is slightly more complex depending upon the client. In most cases the easiest way to pass options to LSF from SAS that would encompass all jobs submitted to the system would be in the Logical Grid Server definition within the SAS Management console via the “Additional Options” line in the Grid Server Properties dialog. This most commonly is used to pass all jobs to specific queues for example. These types of options can also be passed via the JOBOPTS= option in the GRDSVC_ENABLE function but this is well beyond the scope of this paper.

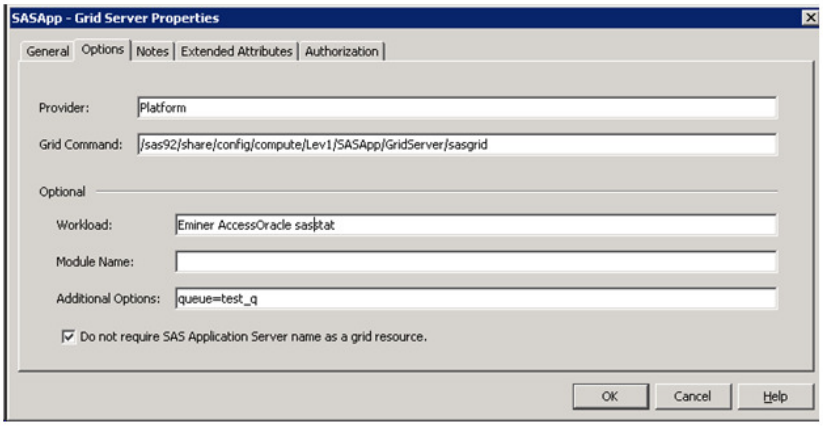


Figure 6 Logical Grid Server Definition Properties

MANIPULATING SLOTS

Another very common procedure when setting up a grid is the manipulation of slots. Slots are a relatively simple concept in LSF defined as the basic unit of processor allocation. This means that typically administrators would think of setting the number of slots equal to the number of logical cores on a node. There are of course other use cases. For example one may wish to allow fewer jobs to land on a node in order to preserve performance of an individual job especially in the case of large memory use for example. In other cases, jobs may be very small in that they use very little CPU or memory so allowing for more slots than logical CPUs may provide satisfactory performance. In the case of SAS, the workload imposed on a node from opening a SAS workspace for example tends to be intermittent so allowing more workspaces on a single node can provide decent performance. Batch workloads with consistently levels of system utilization would do better with a number equal to or less than the number of logical cores in most cases. Below is section of lsb.hosts where the number of slots can be defined in the column MXJ which stands for maximum jobs.

```
HOST_NAME MXJ   r1m     pg    ls    tmp  DISPATCH_WINDOW  # Keywords
#host1     () 3.5/4.5   15/   12/15  0      ()               # Example
#host2     !    3.5    15/18  12/    0/  (5:19:00-1:8:30 20:00-8:30)
#host4     10    ()      ()    ()     ()     ()               # Example
default    !    ()      ()    ()     ()     ()               # Example


MXJ - max job slots where an integer defines an exact value, ! mean set value equal to
ncpus and () mean unlimited.
r1m - 1 min exponentially averaged CPU run queue length
pg - memory paging rate exponentially averaged over the last min, pages per second
ls - number of logged in users
```

`tmp - amount of temp space in /tmp`

`dispatch_window - times when jobs can be dispatched to this host`

HOW IS SCHEDULING POLICY DEFINED AND CONTROLLED?

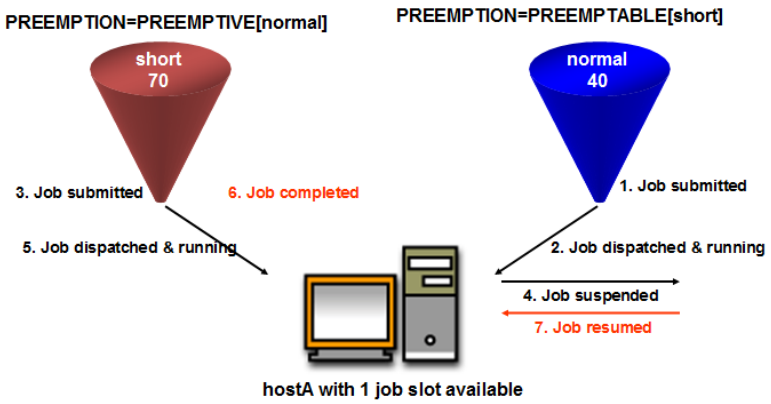
The previous section dealt with defining and using the properties important to individual jobs. This section will deal with the handling of many jobs simultaneously. This is the core function of a workload management system. This is commonly referred to as "scheduling" and should not be confused with workflow (not workload) management (aka Distributed Enterprise Scheduling) such as a workflow created for submission to SAS Grid via DI Studio or "scheduling" a workflow using SAS Management Console to run on the Grid. In this case the scheduling refers to the work completed by the mbsched daemon to determine what job will land where with assistance from MBD (and others).

FCFS

First Come First Serve – This is the default scheduling method whereby LSF attempts to schedule jobs in the order they are received. This doesn't allow users with higher priority work to cut in line or "preempt" other jobs.

PREEMPTION

Preemption simply stated is the ability to allow another job to have priority to the compute resources over another job. This can take several forms. The most common form is called "Job Slot" preemption. The other two are called Advanced Reservation and License preemption. These probably deserve separate consideration and for now will be considered beyond the scope of this paper. Job slot preemption involves editing queue definitions to be either `PREEMPTIVE` or `PREEMPTABLE` in `lsb.queues`. This will allow the jobs in a preemptive queue to suspend running jobs dispatched by the preemptable queue.



FAIRSHARE

Fairshare scheduling as the name implies attempts to address the issue of sharing resources through assigning shares to users. This type of scheduling calculates a dynamic priority for each user by analyzing not only how many shares a user has but also determining the current and historical workload on a grid. This includes the number of slots available, the total number of slots, the amount of time jobs have been running and finally the run time of finished jobs. There are a number of specific types of Fairshare but the basic system as one might expect simply balances usage between all users. This is also called "Round Robin" scheduling and is setup by setting the `FAIRSHARE=` option in `lsb.queues` for a particular queue. Additional more advanced versions handle resource contention across groups of user, queues and hosts but are not covered here.

EXCLUSIVE

Exclusive scheduling very simply gives a user exclusive use of an entire host once a job has been dispatched. This is very useful where the amount of memory of a given job can vary widely or resource contention from subsequent jobs is not desired. This feature is set by enabling `EXCLUSIVE = Y` for a queue in `lsb.queues`. Normally an exclusive job cannot be preempted but this can be overridden with settings in `lsb.params` and `lsf.conf`.

BACKFILL

As the name suggests, backfill scheduling allows smaller jobs to run using slots reserved for other jobs provided they will not disrupt the start time of subsequent jobs. This by definition means the scheduler must some how know how long each job will take hence the requirement for a runlimit to be submitted with the backfill job. This can be done using "-W" option during job submission or by setting queue or application runlimits. This is a compelling feature to allow smaller jobs to complete rapidly without disturbing larger jobs. This feature increases overall cluster efficiency dramatically when properly applied.

CONCLUSION

The options to configure SAS Grid include many technologies and considerations. A simple examination of figure one demonstrates the variety of servers, services and client configurations influencing job placement. How a request is formed upon submission followed by policies set at user, queue or site wide levels along with the scheduling policy allow massive flexibility to alter workload placement. In short Platform LSF and by extension SAS Grid has the flexibility to address the most complex use cases for any size organization.

REFERENCES

- http://www.sas.com/high-performance-analytics/
- http://www.sas.com/grid
- http://support.sas.com/rnd/scalability/platform/index.html
- http://support.sas.com/rnd/scalability/grid/PSS6.1_Unix_Install.pdf
- http://www.hpcwire.com/hpcwire/2007-11-02/platform_acquires_scali_manage_business.html
- http://www-03.ibm.com/press/us/en/pressrelease/36372.wss
- http://support.sas.com/documentation/cdl/en/gridref/64808/PDF/default/gridref.pdf
- http://support.sas.com/resources/papers/SASScheduling.pdf

RECOMMENDED READING

- http://support.sas.com/rnd/scalability/grid/gridpapers.html
- http://support.sas.com/rnd/scalability/grid/griddocs.html
- http://support.sas.com/documentation/cdl/en/gridref/64808/PDF/default/gridref.pdf

CONTACT INFORMATION

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

Adam H. Diaz
IBM Platform Computing
500 SAS Campus Drive R1465
Cary, NC 27513
919-531-1644
adam.diaz@sas.com or adiaz@us.ibm.com
http://www.sas.com/grid