Paper 11562-2016

Beyond Best Practice: Grid Computing in the Modern World

Jan Bigalke, Allianz Managed Operation & Services SE

& Gregory S. Nelson, ThotWave Technologies

Architects do not see a single architectural solution, such as SAS® Grid Manager, satisfying the varied needs of users across the enterprise. Multi-tenant environments need to support data movement (such as ETL), analytic processing (such as forecasting, predictive modeling) and reporting which can include everything from visual data discovery to standardized reporting. SAS® users have a myriad of choices that may seem at odds with one another - such as in–database vs. in–memory or data warehouses (tightly structured schemes) versus data lakes and event stream processing. Whether fit for purpose or for potential, these force us as architects to modernize our thinking about the appropriateness of architecture, configuration, monitoring and management.

This paper will discuss how SAS® Grid Manager can accommodate the myriad use cases and the best practices used in large scale, multi-tenant SAS environments.

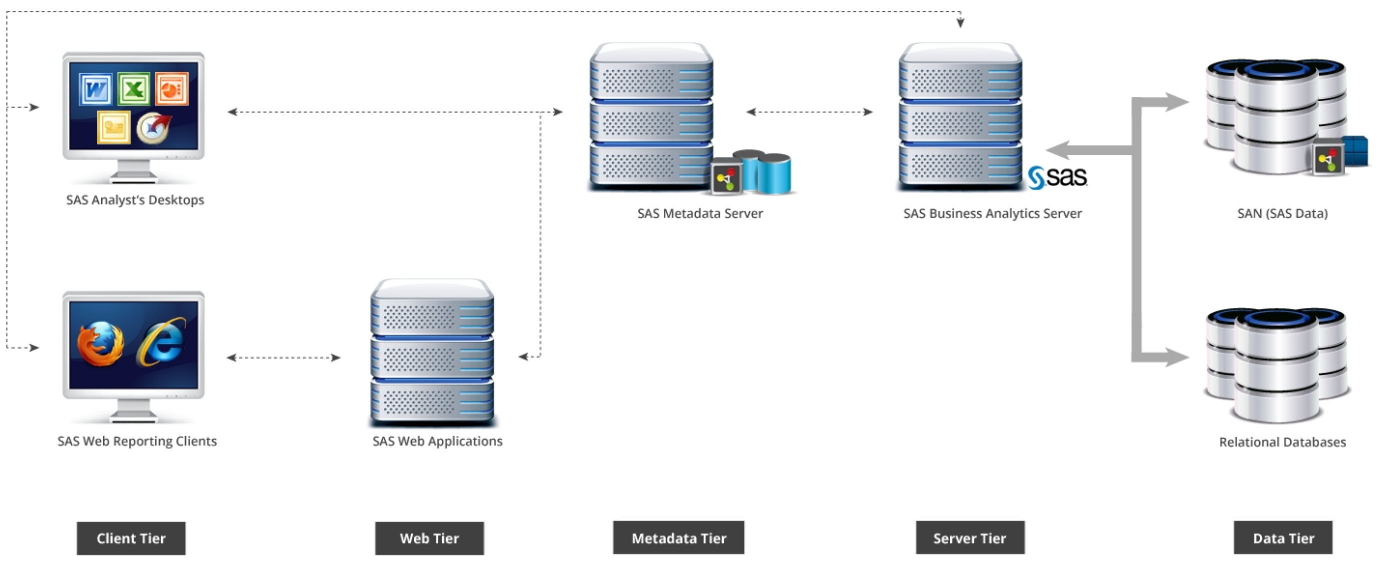
Table of Contents

Introduction

SAS Grid Manager is a SAS® product offering designed to improve the scalability and flexibility of SAS environments. Given the proliferation of commodity-based infrastructures (often using x86 technology) an architecture is needed that can follow the principle of start small and grow as the needs of the user community grow. This paper will take readers beyond the initial installation of SAS Grid and explore topics such as optimizing grid environment based on different types of workloads. Some of the important questions discussed are - how use cases can inform the way people think about the theoretical limit of their technical environment and how monitoring should be an input to the ongoing architecture review and change. Detailed topics will include discussion of workload management and prioritization based on different types of both planned and actual workloads; active monitoring and high availability; load balancing algorithms; and grid design including host, queues and grid option sets. Finally, the paper will cover implications of using Kerberos security and ticket delegation across SAS components and with external data sources like databases and Hadoop.

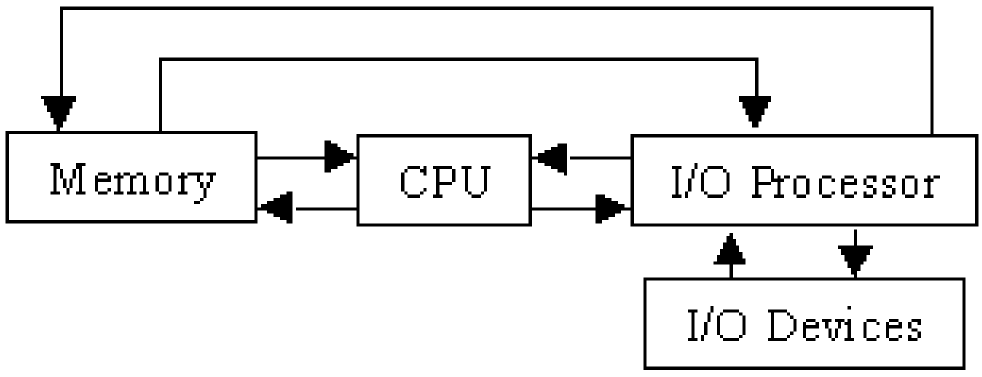
Understanding Types of Workload in an Enterprise SAS Environment

The modern SAS environment, such as the SAS® Enterprise BI Server, includes several SAS components from clients that are installed on personal computers like Enterprise Guide® (EG), SAS® Studio and Data Integration (DI) Studio to those that run in server-based environments and include components such as the Pooled Workspace Server, OLAP Server, Stored Process Server and the Workspace Server. The server components are the foundation of other services that are built on top of the SAS execution servers. The Stored Process and the Pooled Workspace Servers are used from the web applications whereas the Workspace Servers are utilized mostly by the development clients (such as EG and DI Studio.)



Each component in the SAS architecture has a fundamentally different runtime characteristic. Online (real-time, interactive) workload that supports reporting or other web based presentation forms need fast response times and are usually based on quick analytic or summary tasks. For batch execution, the ability to parallelize jobs is more important than the immediate response time. Developers, on the other hand, often will iterate as they do code development and is characterized by iterative executions that require both fast response time (with smaller datasets) and prioritization so that development activities can occur.

Moving beyond traditional SAS “BI” applications, there is a need to consider evolving strategies such as in-memory and in-database technologies which serve to complicate the role of the architect in ensuring service level agreements (SLAs) and planning for optimal (and evolving) architectures. The following diagram illustrates the competing forces of memory, CPU and I/O.



Based on these general assumptions the following principals can be formulated:

- For online processing avoid everything that increases the latency (I/O Processor and I/O Devices). In most cases also the data has to be stored in optimized structures to support short response times
- Analytical workload needs more computing capacity and can benefit from parallelization (CPU)
- Understanding mixed workloads and characterizing the impact to CPU, memory and I/O are a critical function of the architect[1]

Understanding these generalized principles is important to evaluate how SAS Grid can be tuned specifically to address the challenges and tradeoffs. The following sections will introduce the principles of SAS Grid and what it does well out of the box and then address how elements of the grid environment can benefit from additional review and optimization.

Grid Principles

SAS Grid was originally designed for large-scale batch workloads. (Think of an environment where lots and lots of jobs are scheduled to run overnight.) What’s needed in this case is the system to become its own traffic cop, in terms of which resources are allocated to the jobs, understanding which resources are required based on the characteristics of the jobs and the ability to prioritize the jobs as they execute within the schedule window.

The core capability to parallelize the processing of the data allows the computation of huge amount of data in relatively short times. When SAS Grid Manager was launched, they initially supported only SAS Sessions. Since then, SAS has added the SAS Workspace Server, Stored Processes, OLAP, Enterprise Miner, Data Integration jobs and so on.[2]

The following table gives a quick overview about the SAS versions and the introduced capabilities.

SAS Version	**new capabilities (key points)**
9.4 M2[2]	Grid Manager plug-in for the Environment Manager
9.4 M1[2]	stored process servers ,

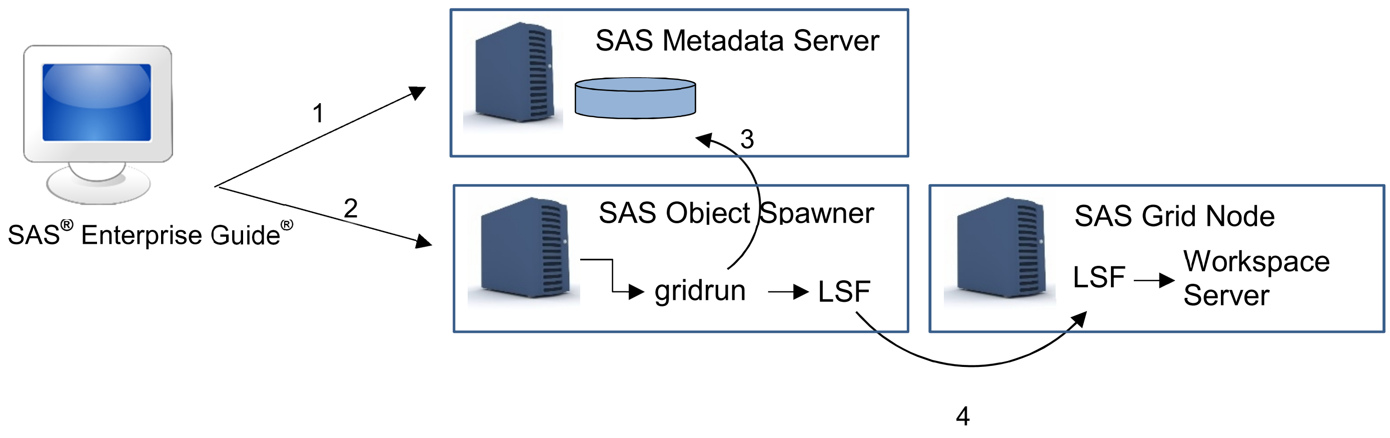
[1] https://en.wikipedia.org/wiki/Software_architect
[2] http://support.sas.com/documentation/cdl/en/gridref/67371/PDF/default/gridref.pdf

	pooled workspace servers grid-launched
9.4 M0[2]	grid options, grid-launched workspace servers
9.3[3]	load balancing for stored process servers, OLAP servers and pooled workspace servers
9.2[4]	SAS code analyzer grid-launched batch SAS jobs load balancing for SAS Workspace Servers

SAS Grid Manager is a SAS product being integrated with Platform Suite for SAS[5]. Here, the Platform Suite provides scheduling across distributed servers and load balancing features. The product included is the Platform LSF[6] (Load Sharing Facility) from IBM. The following flow chart shows the process of submission to Grid as an example with the Enterprise Guide[7] starting a Workspace Server.

How Grid Processing Works

Consider, for example, the following grid submission of a Workspace Server: (Note: for a video demonstration of how this works, please visit: http://thotwave.com/portfolio-item/serving-sas-a-visual-guide-to-sas-servers/)



The user is starting the Enterprise Guide and connects to the backend server (1+2 in the diagram). The backend processes are started by the SAS Object Spawner. In the case of a grid submitted session the SAS Object Spawner creates a process called gridrun, because in a grid scenario the SAS session can be placed on any of the grid nodes and not only direct on the server where the Object Spawner is started. The "gridrun" process manages the communication with the LSF components. First gridrun receives the settings from the metadata (via Grid Options Sets, number 3 in the figure)[8] and passes the setting to LSF. In the example below the OPTION "project=SASApp" is passed from the metadata server. In the next step (4 in the figure) LSF starts the Workspace Server on the target grid node. The information about the program to execute is passed to LSF via the COMMAND tag (see the log below).

[3] http://support.sas.com/documentation/cdl/en/gridref/64808/PDF/default/gridref.pdf
[4] http://support.sas.com/documentation/cdl/en/whatsnew/62580/PDF/default/whatsnew.pdf
[5] http://support.sas.com/rnd/scalability/platform/
[6] http://support.sas.com/rnd/scalability/platform/PSS9.1/lsf9.1.3_admin.pdf
[7] http://support.sas.com/resources/papers/proceedings14/SAS375-2014.pdf
[8] http://support.sas.com/documentation/cdl/en/gridref/67371/PDF/default/gridref.pdf

With the trace log option activated this process is visible in the console log of the Object Spawner.

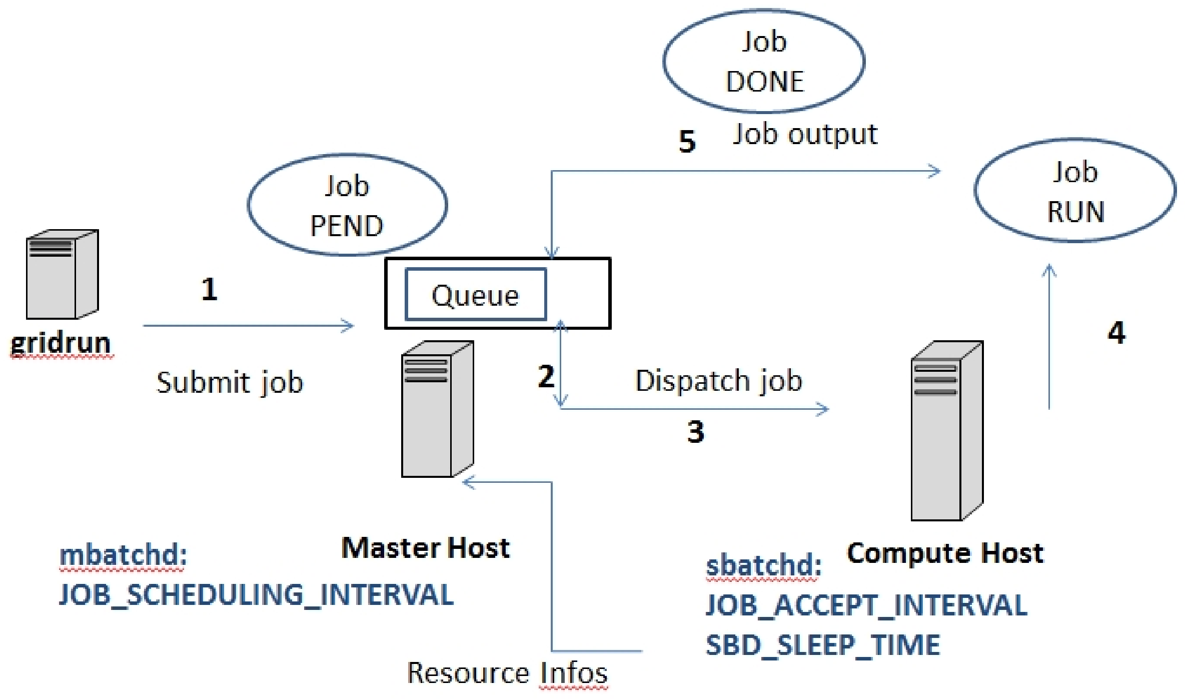
Output 1 shows the output from the ObjectSpawner_console.log with the Trace option.

```
2015-10-07T21:09:23,798 INFO  (gridrun.c:232) -
2015-10-07T21:09:23,798 INFO  (gridrun.c:233) - TKEGRID Proxy Grid Job Runner,
build date: Jul 23 2014 @ 20:33:03
2015-10-07T21:09:23,798 INFO  (gridrun.c:234) - Copyright (C) 2013, SAS Institute
Inc., Cary, NC, USA. All Rights Reserved
2015-10-07T21:09:23,798 INFO  (gridrun.c:235) -
2015-10-07T21:09:23,798 INFO  (gridrun.c:242) - GRIDRUN: Running as testuser
2015-10-07T21:09:23,798 INFO  (gridrun.c:243) -
2015-10-07T21:09:23,800 INFO  (gridrun.c:467) - commHandler: attempting to
connect to master
2015-10-07T21:09:23,800 INFO  (gridrun.c:558) - commHandler: letting master know
initialization is done
2015-10-07T21:09:23,801 INFO  (gridrun.c:590) - commHandler: command received is
>[INIT] [PROVNAME]:"Platform" [MODNAME]:"" [SRVHOST]:"sas94-app1-syst.testdomain"
[SRVPORT]:"0" [USERNAME]:"" [PASSWORD]:"" [TIMEOUT]:"0"
[OPTIONS]:<project=SASApp><.
2015-10-07T21:09:23,836 INFO  (gridrun.c:609) - commHandler: command response is
>[DONE]<.
2015-10-07T21:09:23,837 INFO  (gridrun.c:590) - commHandler: command received is
>[STARTJOB] [JOBNAME]:"SAS Enterprise Guide_SASApp - Workspace Server_4101A009-
6340-2B46-8E08-8DB2933E8182" [RESOURCES]:""
[COMMAND]:</var/opt/data/sas/sas94/configAPP/Lev1/SASApp/WorkspaceServer/Workspac
eServer.sh> [ARGUMENTS]:<-noterminal -noxcmd -netencryptalgorithm AES -metaserver
sas94-meta-syst.testdomain -metaport 8561 -metarepository Foundation -locale
en_US -objectserver -objectserverparms "delayconn sph=hosta.testdomain
protocol=bridge spawned spp=42449 cid=0 pb classfactory=440196D4-90F0-11D0-9F41-
00A024BB830C server=OMSOBJ:SERVERCOMPONENT/A5ZI7NU4.AY0000WN cel=everything lb
recon grid "keepalive=30"" -METAUSER '"testuser@!*(generatedpassworddomain)*!"' -
METAPASS 49944139d506b727d1555D7b1d8E6162 > [OPTIONS]:<> [ARMCORR]:"" [FLAGS]:"0"
[INFILES]:"" [OUTFILES]:"" [HOSTS]:"sas94-app1-syst.testdomain,sas94-app2-
syst.testdomain" [MPIPROCS]:"0" [PROCSHOST]:"0"<.
Job <33707> is submitted to queue <qiSASApp>.
```

Output 1. Output Object Spawner Console Log

Processing Inside LSF

The last two lines of the console log above show the passing of the parameters to LSF and the JobID "33707" that is created from LSF to process this job. The following picture describes what happens inside LSF to create this job.



1. **Submit the job**

   gridrun submits the job to a queue. When submitting without a queue, the job is submitted to the default queue. Jobs are held in a queue waiting to be scheduled are in the PEND state. LSF assigns each job a unique job ID for each submitted job.

2. **Schedule the job**

   On the master host runs the master batch daemon (mbatchd).

   The master batch daemon (mbatchd) looks at jobs in the queue and sends the jobs for scheduling to the master batch scheduler (mbschd). The look up time interval is defined by the parameter JOB_SCHEDULING_INTERVAL). The mbschd evaluates jobs and makes scheduling decisions based on: 1) Job priority, 2 Scheduling policies, and 3) Available resources. The resource information is updated from the compute hosts in the interval defined by the SDB_SLEEP_INTERVAL.

   mbschd selects the best host where the job can run and sends its decisions back to mbatchd. Resource information is collected at preset time intervals by the master load information manager (LIM) from LIMs on server hosts. The master LIM communicates this information to mbatchd, which in turn sends it to mbschd to support scheduling decisions[9].

3. **Dispatch the job**

   When mbatchd receives scheduling decisions, it dispatches the jobs to hosts.

4. **Run the job**

   On the hosts the slave batch daemon (sbatchd) receives the request from mbatchd. The sbatchd creates a child, the execution environment and starts the job using the remote execution server (res).

5. **Return output**

   After the execution the status is set DONE for a job without any problems and EXIT if errors occur.

   If JOB_ACCEPT_INTERVAL is set to non-zero value this host will not accept a new job within JOB_ACCEPT_INTERVAL interval

[9] http://support.sas.com/rnd/scalability/platform/PSS9.1/lsf9.1.3_admin.pdf

Based on this information the response time can be calculated. In the worst case, if job slots are available (less jobs running than the maximum slots defined) and no jobs with higher priority are in the PEND state.

```
Response time = 1 (* MDB_SLEEP_TIME)
```

If MDB_SLEEP_TIME is 5 seconds then the wait time can be 5 seconds. This type of scheduling is not designed for short response times. Web requests need response times within seconds for the whole process chain; they cannot spend seconds in the scheduling process.

But for workload with longer execution time and with the need to find free capacity inside the grid this approach provides advantages to a round robin distribution of the jobs.

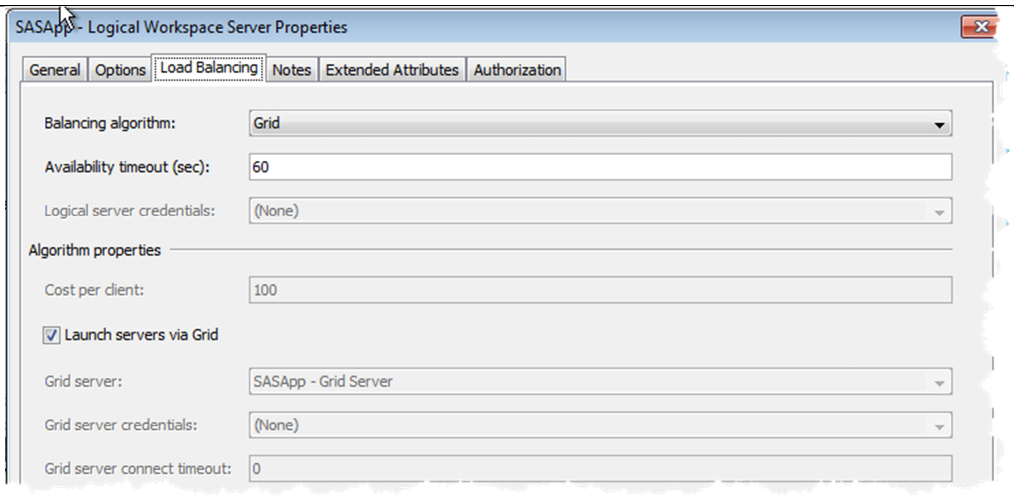
Best Practices

Grid Configuration related to the characteristic of the workload

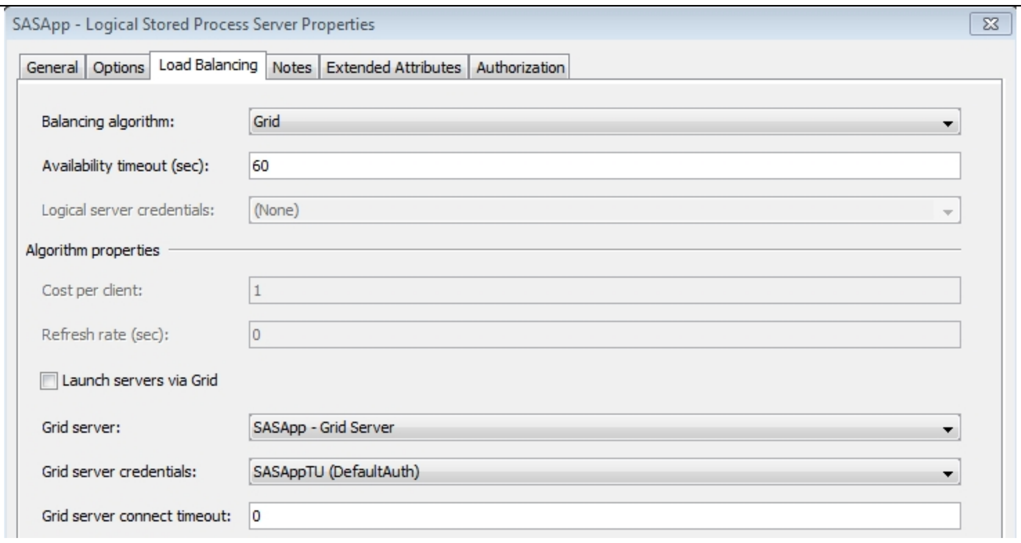
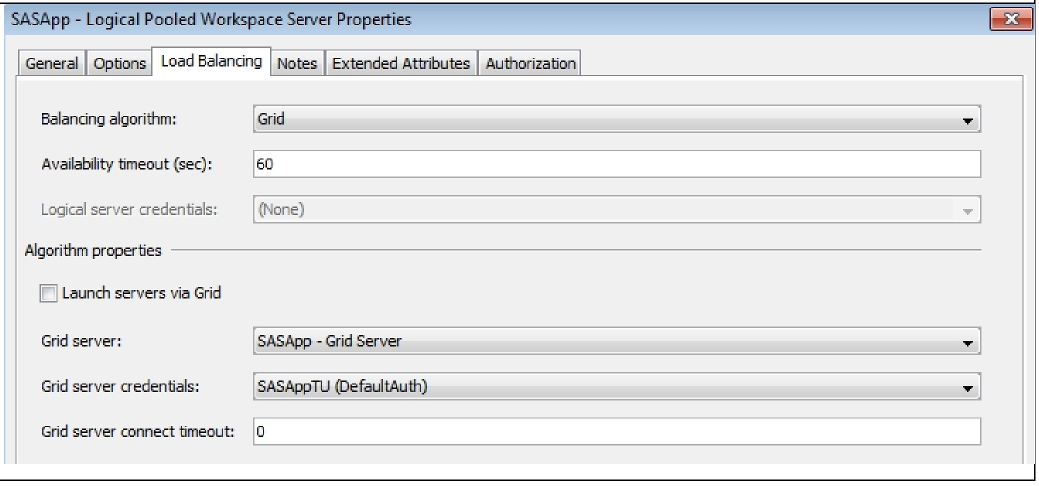
As mentioned previously, there are a number of use cases (workloads) which illustrate the importance of having different configurations to handle those various workloads. The grid scheduling process, introduced in the previous chapter, takes some time and fits well for Batch workload and interactive analytical sessions like the Workspace Server where the initial session can take a few seconds but is then persisted for the entire interactive user session (e.g., SAS Enterprise Guide[10].) Online workload like STP's is directly submitted using a pool of services that are persistent across user sessions.

Configuration of the Workspace Server and the Stored Process Server:

The picture below shows a configuration for an analytical session with the workspace server and an online session with the Stored Process Server.

SAS Server	**Grid Configuration Properties**
SAS Workspace Server	 SASApp - Logical Workspace Server Properties; Load Balancing tab: Balancing algorithm: Grid; Availability timeout (sec): 60; Logical server credentials: (None); Algorithm properties: Cost per client: 100; ☑ Launch servers via Grid; Grid server: SASApp - Grid Server; Grid server credentials: (None); Grid server connect timeout: 0

[10] http://support.sas.com/resources/papers/proceedings14/SAS375-2014.pdf

SAS Stored Process Server	 SASApp - Logical Stored Process Server Properties General | Options | Load Balancing | Notes | Extended Attributes | Authorization Balancing algorithm: Grid Availability timeout (sec): 60 Logical server credentials: (None) Algorithm properties Cost per client: 1 Refresh rate (sec): 0 ☐ Launch servers via Grid Grid server: SASApp - Grid Server Grid server credentials: SASAppTU (DefaultAuth) Grid server connect timeout: 0
Pool Workspace Server	 SASApp - Logical Pooled Workspace Server Properties General | Options | Load Balancing | Notes | Extended Attributes | Authorization Balancing algorithm: Grid Availability timeout (sec): 60 Logical server credentials: (None) Algorithm properties ☐ Launch servers via Grid Grid server: SASApp - Grid Server Grid server credentials: SASAppTU (DefaultAuth) Grid server connect timeout: 0

First the workspace server: The settings for the “grid launched” can be set on the logical workspace server on the tab load balancing in the SAS Management Console (SMC). Similarly, the Grid algorithm on the stored process server can be specified so that the distribution of jobs is based on the information passed back from the SAS Grid Manager. The balancing algorithm is configured on the logical stored process also on the tab ‘load balancing’ but the Stored process servers are not launched using grid because of the latency in startup times.

If the online workload should also be able to run on every server in the grid, every server needs a running instance of the Object Spawner. The Object Spawner instances then distribute the workload based on the selected balancing algorithm between the servers.

Queue definitions:

Grid jobs are submitted into queues based on the characteristics of a job. If no queue is specified, then the default queue is used. For the different workload types queues can be configured to handle the specific workload for that particular use case. That approach has clear advantages. For example, if DISPATCH_ORDER=QUEUE is set in the master queue, jobs are dispatched according to queue priorities. The following suggestions can help to create this configuration.

The queue for the Enterprise Guide should have a higher priority than the batch jobs. This is, in part, because of the interactive and immediate nature of interactive development activities. If these have lower priorities, then due to serious contention between batch jobs and interactive users may never get their session started.

Another configuration example is the “stop” and “resume” condition based on thresholds[11]. In the described environment the online workload has absolute priority. For such situations a stop and resume condition support the priority for the online workload. If the workload on the nodes increases then jobs with lower priority will be suspended until the resume condition for this job is met. The parameter “qjob_limit” can limit the number of jobs per queue; this setting is helpful if one queue should not be able to use all available job slots in the grid.

The following table highlights some of the most common configurations that can be set for queues. Note, the examples provided are based off a grid that has 16 cores and 64 job slots.

Parameter	**Example (Interactive Queue)**	**Example (Batch Queue)**	**Definition**
PRIORITY	`PRIORITY=50`	`PRIORITY=20`	The relative priorities as compared to other queues
NICE	`NICE=20`	`NICE=10`	Specifies the execution priority change, based on Linux “nice” values.
CPULIMIT	`CPULIMIT=5`	`CPULIMIT=15`	a time limit applied to jobs
UJOB_LIMIT	`UJOB_LIMIT=5`	`UJOB_LIMIT=2`	the maximum job slots per user in a queue
PJOB_LIMIT	`PJOB_LIMIT=10`	`PJOB_LIMIT=5`	the maximum job slots per processor in a queue.
QJOB_LIMIT	`QJOB_LIMIT = 120`	`QJOB_LIMIT = 60`	the maximum jobs in a queue.
HJOB_LIMIT	`HJOB_LIMIT = 4`	`HJOB_LIMIT = 4`	Maximum number of job slots that this queue can use on any host
CHUNK_JOB_SIZE	`CHUNK_JOB_SIZE = 4`	`CHUNK_JOB_SIZE = 4`	Specifies the maximum number of jobs allowed to be dispatched together in a chunk.
r1m	`r1m=0.3/1.5`	`r1m=0.3/1.5`	1-minute CPU run queue length (alias:cpu)
Ut	`ut=0.2`	`ut=0.2`	1-minute CPU utilization (0.0 to 1.0)
r15s	`r15s=0.3/1.5`	`r15s=0.3/1.5`	15 second CPU run queue length (alias:cpu)
It	`it=10/1`	`it=10/1`	Idle time (minutes) (alias: idle)

[11] https://www-01.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_admin/suspend_conditions_queue_set.html

Multi-tenant considerations

A multi-tenant configuration for the grid follows the same principles as the multi-tenant configurations for non-grid installations[12]. Each customer has its own SAS application Server context in the SAS metadata. The running services for the customer on the compute nodes are under EGO (Platform Enterprise Grid Orchestrator) control. If the customers need different settings for the queue definitions like possible job slots, separate queues for each customer make sense.

The queues can be protected with additional security so that only the assigned customers can use their queues. In the queue definition the attribute users can define a group of allowed users. For the group resolution a custom program called egroup can be created.

Custom queues should than also used from the build in features of the SAS clients. For the EG for example a macro variable with the queue name in the `autoexec_usermods.sas` from the workspace server configuration should be set.

```
%let _gridjoboptions=queue= queue name;
```

The resource management inside a shared environment is a key element of multi-tenant configuration. Each customer should be able to process his workload even if some other customers submit heavy workload to the grid. In the previous chapter the online workload was excluded from the grid management. But for the resource management the whole measurement of the workload of each customer is necessary. For LSF external load indices[13] can be defined to handle this requirement.

An option on Linux operating systems to get the resource consumption for each customer is to put all task of a customer into a dedicated CGROUP. The paper[14] from the SGF 2014 shares some thoughts on the use of CGroups.

Based on this idea the stop and resume conditions of a queue can be enhanced to the utilization of each customer cguxx. In this case a job is only suspended if the customer is using his own capacity cguxx>90 and the overall utilization cpuusg is also high (here >95).

```
stop_cond   = select[ (cpuusg > 95.0) && (cguxx > 90.0)) ]

resume_cond = select[ (cguxx < 95.0) || (cpuusg < 95.0) ]
```

Hints for the developers

One of the benefits of the SAS Grid Manager is that it should be transparent to the users – and that includes the developer community that relies on SAS “just working”.

As can be recalled from previous discussion, developers often run very short jobs in parallel. This is often orthogonal to the design of the grid environment in general as large, parallel batch jobs are often the norm. The short workload bursts that developers place on the environment generates an often unnecessary overhead for dispatching the job and instantiating the SAS environment and will often be more consumptive than the actual job itself.

Switching between compute nodes during the computation of a workload has an impact to the file cache. The bundle of work that is included in a job should recognize this behavior. Sometimes a larger job with more tasks is better than to split the work in several small jobs that then run on different nodes in the grid.

A simple recommendation to the developers: Avoid parallelization for jobs running only a few seconds.

[12] http://support.sas.com/resources/papers/proceedings13/494-2013.pdf
[13] https://www-01.ibm.com/support/knowledgecenter/SSETD4_9.1.3/lsf_admin/elim_about_lsf.dita
[14] http://support.sas.com/resources/papers/proceedings14/SAS289-2014.pdf

GRID and interaction with other systems (databases / Hadoop)

SAS jobs in the grid can use not only SAS data files stored in the shared filesystem of the servers but also data accessible via SAS/Access interfaces. The SAS interfaces should have the same configuration on all nodes in the grid. The recommendation is to store the configuration of the database clients in the shared files system as well. This approach supports a consistent configuration for the SAS access modules.

An enterprise scheduler can be integrated with the SAS grid[15]. In this case the suggestion is to start the enterprise scheduler agent with EGO. If the Server with the Enterprise Scheduler agent fails, the agent is automatically restarted on another server inside the cluster. The integration module can wait until the job inside the grid is finished so that the job results code can be send synchronously to the enterprise scheduler.

SAS provides also an access engine for Hadoop (SAS ACCESS to Hadoop.) To configure the Hadoop interface jar files from the Hadoop cluster have to be copied to the SAS installation machine(s)[16]. Instead of RDBMS clients that can support databases over version boundaries, the Hadoop interface is tightly governed and tied to versions of a Hadoop cluster. If there are several Hadoop Clusters on different versions or different distributions the only option is to install the related jars on different nodes in the grid. To steer the jobs on the required node “resources” can be used. A resource can be defined per node and in the job request the resource request has to be included. This approach is described in the grid documentation[3] (look for “Defining and Specifying Resources”).

Connections to Big Data systems like Hadoop are often configured with Kerberos authentication to support the security model inside the Hadoop system. SAS provides an Access Interface to Hadoop. A SAS session can be enabled to use the Kerberos protocol for the authentication to Hadoop[17]. In this case all the hosts for SAS/ACCES to Hadoop must be enabled for Kerberos. The request flow looks like the following request chain. The client e.g. the SAS Enterprise Guide connects to the SAS Grid via IWA (Integrated Windows Authentication). The client request and received a service ticket for the SAS servers. This ticket is send to the SAS session on the Server for authentication. The SAS session use this ticket to acquire a new ticket for the Hadoop system. To make this possible the service principal account that defines the SAS service needs the trusted for delegation right. Details about this process can be found inside the SAS documentation[18] (look for Integrated Windows Authentication).

[15] http://support.sas.com/rnd/scalability/grid/InterfaceEnterpriseScheduler.pdf
[16] https://support.sas.com/resources/thirdpartysupport/v94/hadoop/hadoopbacg.pdf
[17] http://support.sas.com/resources/papers/Hadoop_Architecture.pdf
[18] http://support.sas.com/documentation/cdl/en/bisecag/67045/PDF/default/bisecag.pdf

Capacity considerations

The number of jobs that a grid environment can support depends on the available job slots. The slots can be dynamically calculated e.g. 4 jobs per core or in the LSF configuration a maximum number of jobs per host can be defined. This configuration can be set in the file lsb.hosts. The variable MXJ defines the maximum number of jobs per host.

With these values being aggregated over the whole cluster nodes it’s possible to calculate the max number of concurrent active jobs.

$$\sum_{k=0}^{n} \text{Jobs}(k) = \text{MXJ}_1 + \text{MXJ}_2 + .. + \text{MXJ}_n$$

How many jobs can be processed in a timeframe? To calculate this, the following assumptions are necessary.

JOB_ACCEPT_INTERVAL is 1 and the MBD_SLEEP_TIME is 5 (seconds).

Therefore, Platform LSF dispatches one job to a particular machine and waits for 5 seconds before dispatching another job to the same machine regardless of how long each job takes. (JOB_ACCEPT_INTERVAL * MBD_SLEEP_TIME = 1 * 5 = 5). The highest number of jobs that can be dispatched for a single queue and a given host would be 12 per host in the example. To calculate this consider

a) Average job duration: 5 seconds

b) JOB_ACCEPT_INTERVAL: 1

c) MBD_SLEEP_TIME: 5

d) 4 cores per host

e) 2 hosts

f) 4 job slots per core

Formula AA for number of jobs per host per minute possible: 60 seconds / (B*C)

Formula BB for number of jobs possible for the grid environment: Formula AA * E

Calculations:

Formula AA = 60/ (1 * 5) = 12 jobs per host per minute

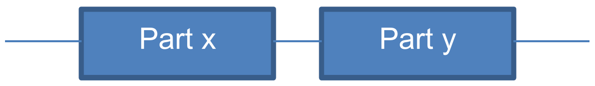
Formula BB = 12 * 2 = 24 jobs per minute for the whole cluster

High availability considerations

Before SAS 9.4 high availability deployments often had been a challenge. Recent enhancements to components such as Metadata Server made this much simpler. In SAS 9.4 all components can be clustered[19] [20]. For the cluster configuration itself a shared file system is essential. For higher availability the introduction of a high available shared file system is a prerequisite. The reason is that grid installations have a serial dependency to the shared file system and also the metadata server cluster needs a storage that can be accessed from all members of the cluster.

[19] http://support.sas.com/documentation/cdl/en/bisag/68240/PDF/default/bisag.pdf
[20] http://support.sas.com/documentation/cdl/en/bimtag/68217/PDF/default/bimtag.pdf

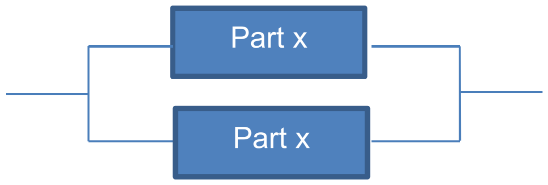
Availability in Series is computed in the following manner:



The combined availably is calculated with the formula:

$$A = A_x A_y$$

Availability in parallel with introduction of redundancy can reach higher availability rates.



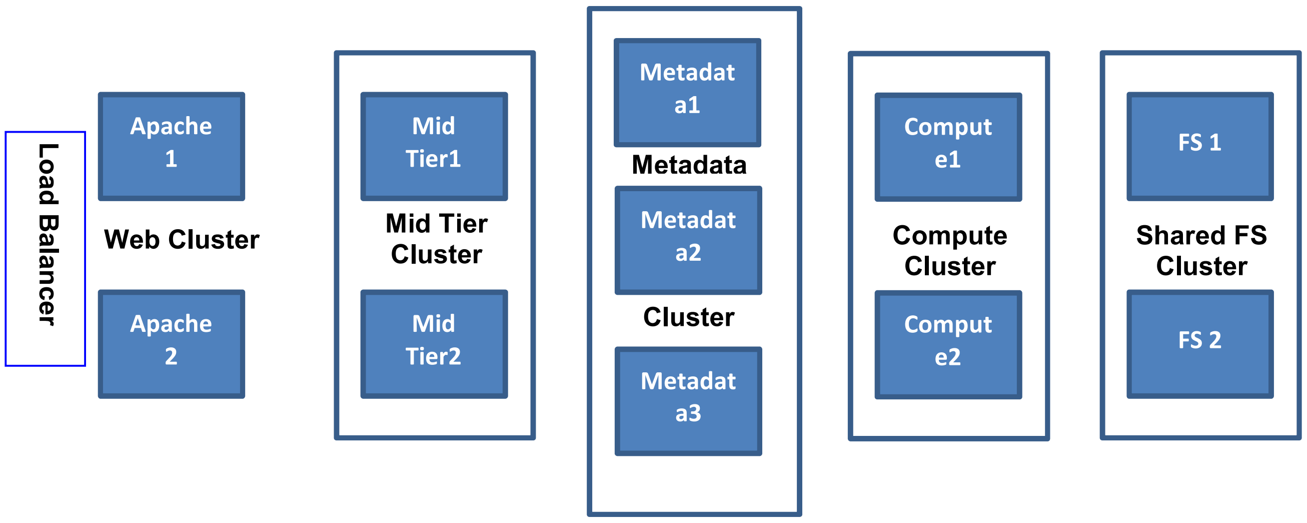
Parallel availability is calculated with the equation:

$$A = 1-(1-A_x)^2$$

The starting point for the planning of high availability is the desired availability for the overall system. Based on the introduced basic formulas and the technical restrictions for each application the number of needed cluster members for each layer like Web Application, Metadata Server and compute nodes can be calculated. If the grid system is huge and the intention is not to start all SAS services like Object Spawner, OLAP servers on every node, EGO can be used to orchestrate the services and the related node. EGO is a part of the LSF suite embedded from SAS Grid.

In EGO the number of instances of services that should run in the cluster can be defined and if one node with a running service fails, the service is started automatically on another node in the grid.

Scenario for a high availability configuration with distributed components



Enterprise Grid Orchestrator (EGO) and GRID

At a high level, SAS 9.4 Grid Manager includes Platform Enterprise Grid Orchestrator Software (EGO) that can be customized to support automatic health check monitoring and automatic restart. SAS and LSF services can be defined as EGO services for EGO to monitor and restart.

Platform Enterprise Grid Orchestrator is installed as part of LSF but needs to be configured to ensure that critical services are monitored. EGO will monitor critical services in the grid environment. EGO can start services if they are not running and restart services on other nodes if a node fails or restart the component if a service itself fails on a node.

To support a failover for the SAS components some preparation in the SAS Metadata is necessary. Each SAS component has to be assigned to all hosts in the Metadata. Afterwards an EGO definition for each service has to be created. In the definition of the service it can be defined on how many servers on the grid the service has to be started.

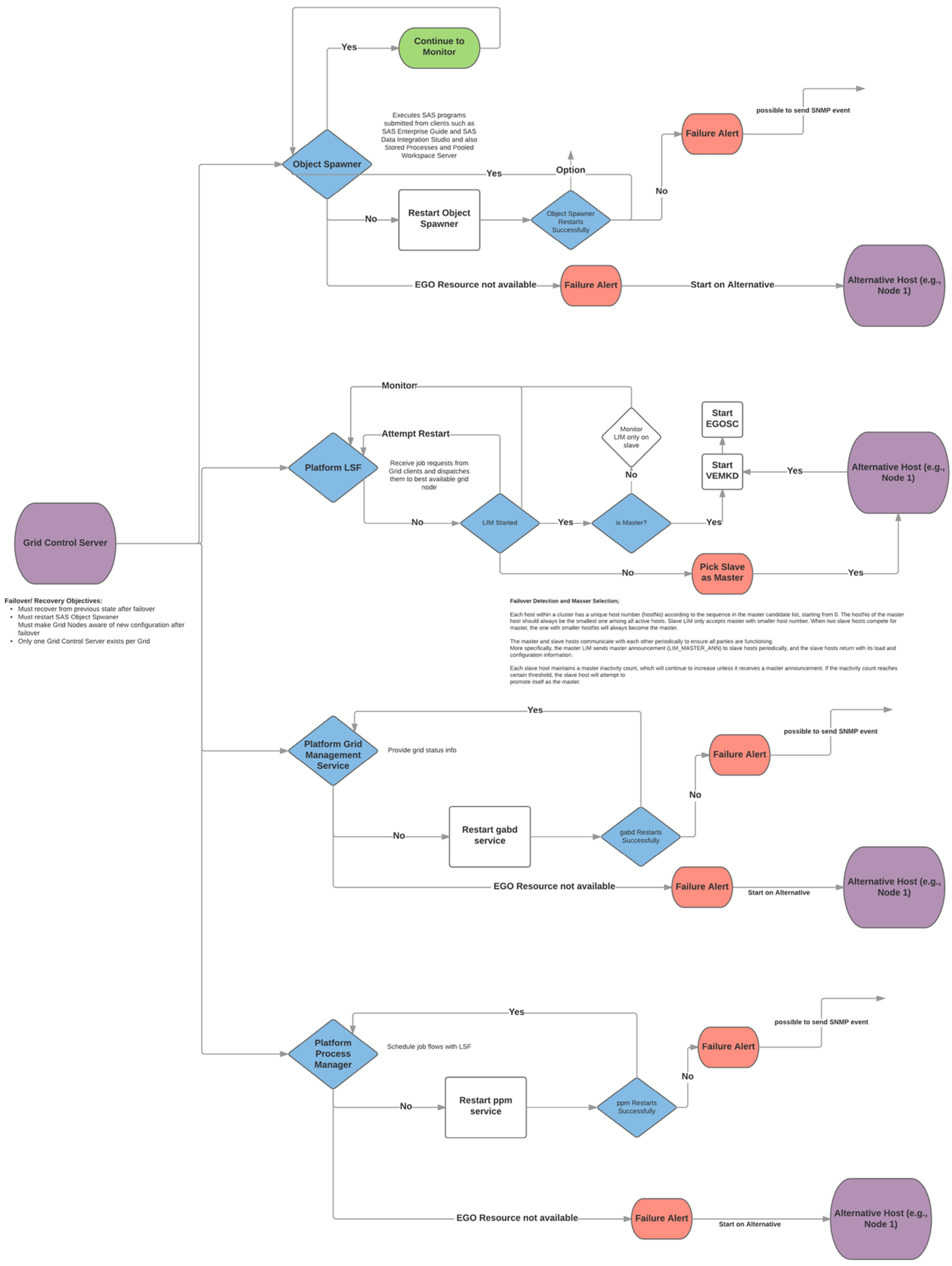
The following list gives a high level overview over the necessary steps to introduce EGO for a SAS Server like the Object Spawner.

1. Enable EGO in lsf.conf (LSF_ENABLE_EGO=Y) if not already turned on
2. Register the service to EGO (e.g., Object Spawner, OLAP Server) copy the template.xml file for a new service and adjust this file
3. Make EGO aware of the new file. Restart EGO: egosh ego restart $MASTER_HOST, when the daemons come up again the service is in the service list egosh service list
4. Test the service e.g with “egosh service start service-name” or “egosh service stop service-name”

For example, let’s say that the goal is to ensure the Object Spawner was started in case it was stopped on a given server. The flow chart below depicts what should happen in the case of an Object Spawner not starting up properly.

GRID CONTROL FAILOVER FINAL

Greg Nelson | February 8, 2016


Continue to Monitor
Yes
Executes SAS programs submitted from clients such as SAS Enterprise Guide and SAS Data Integration Studio and also Stored Processes and Pooled Workspace Server
Object Spawner
Yes
Option
Failure Alert
possible to send SNMP event
No
No
Restart Object Spawner
Object Spawner Restarts Successfully
EGO Resource not available
Failure Alert
Start on Alternative
Alternative Host (e.g., Node 1)
Monitor
Attempt Restart
Platform LSF
Receive job requests from Grid clients and dispatches them to best available grid node
Monitor LIM only on slave
Start EGOSC
Start VEMKD
No
Yes
Alternative Host (e.g., Node 1)
No
LIM Started
Yes
is Master?
Yes
No
Pick Slave as Master
Yes
Grid Control Server
Yes
Platform Grid Management Service
Provide grid status info
Failure Alert
possible to send SNMP event
No
No
Restart gabd service
gabd Restarts Successfully
EGO Resource not available
Failure Alert
Start on Alternative
Alternative Host (e.g., Node 1)
Yes
Platform Process Manager
Schedule job flows with LSF
Failure Alert
possible to send SNMP event
No
No
Restart ppm service
ppm Restarts Successfully
EGO Resource not available
Failure Alert
Start on Alternative
Alternative Host (e.g., Node 1)

Failover/ Recovery Objectives:
- Must recover from previous state after failover
- Must restart SAS Object Spwaner
  Must make Grid Nodes aware of new configuration after failover
- Only one Grid Control Server exists per Grid

Failover Detection and Masser Selection;

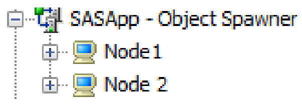
Each host within a cluster has a unique host number (hostNo) according to the sequence in the master candidate list, starting from 0. The hostNo of the master host should always be the smallest one among all active hosts. Slave LIM only accepts master with smaller host number. When two slave hosts compete for master, the one with smaller hostNo will always become the master.

The master and slave hosts communicate with each other periodically to ensure all parties are functioning.
More specifically, the master LIM sends master announcement (LIM_MASTER_ANN) to slave hosts periodically, and the slave hosts return with its load and configuration information.

Each slave host maintains a master inactivity count, which will continue to increase unless it receives a master announcement. If the inactivity count reaches certain threshold, the slave host will attempt to
promote itself as the master.

The following list provides the necessary steps that are needed to register the service and ensure its survivability in the case of failure[21].

1. Assign the possible Servers to the Object Spawner in the SAS Metadata (SMC: properties of the Object Spawner, option tab)



SASApp - Object Spawner
Node 1
Node 2

2. Check if EGO is enabled in lsf.conf (LSF_ENABLE_EGO=Y)
3. Check if the Object Spawner is working (execute `ObjectSpawner.sh start` on both servers and stop the servers if everything works as expected)
4. Create the service definition file in $EGO_CONFDIR/../../eservice/esc/conf/service. Make a copy of the service definition template file service.xml.TMPL to any name with .xml extension and adjust the file [21 (page 48)]
5. Restart EGO : `egosh ego restart all`
6. Check the Service exists in the service list: egosh service list
7. Logon to EGO with an administrative user "`egosh user logon`". Test the service `egosh service start "service-name from the xml file",`
8. The failover can be tested with the closure of a resource in EGO: `egosh resource close Node2`. Details for the failover test are available in the GridHAServices.pdf [21] document (page 46).

As with most environments, a risk assessment should be performed to determine which SAS services should be monitored. Some applications have "built-in" high availability depending on how they were configured. For example, SAS Metadata can be installed in a clustered configuration[22] which provides greater resilience against single points of failure. Similarly, the mid-tier services can also be installed in a clustered configuration[23].

The core elements of LSF have built in capability for internal failover. For example, if the master becomes unavailable, a slave node will take over. When deploying HA objective was to generally provide failover for the Object Spawners, Metadata Server, PM and GMS. The LSF master host is chosen dynamically. If the current master host becomes unavailable, another host takes over automatically. The failover master host is selected from the list defined in LSF_MASTER_LIST in lsf.conf (specified in install.config at installation). The first available host in the list acts as the master.

Running jobs are managed by sbatchd on each server host. When the new mbatchd starts, it polls the sbatchd on each host and finds the current status of its jobs. If sbatchd fails but the host is still running, jobs running on the host are not lost. When sbatchd is restarted it regains control of all jobs running on the host.[24]

For the Metadata server the recommendation is to use clustering for both Metadata and mid-tier server high availability and not EGO. The Object Spawner, OLAP Server and PPM and GMS are candidates for EGO. Beyond the Metadata Server and Mid-Tier services, candidates for high availability include:

- SAS Object Spawner
- Platform Process Manager
- Platform Grid Management Service

[21] https://support.sas.com/rnd/scalability/grid/HA/GridMgrHAServices.pdf
[22] http://support.sas.com/documentation/cdl/en/bisag/68240/PDF/default/bisag.pdf (Chapter 17)
[23] http://support.sas.com/documentation/cdl/en/bimtag/68217/PDF/default/bimtag.pdf (Chapter 16)
[24] http://www-01.ibm.com/support/knowledgecenter/SSETD4_9.1.2/lsf_foundations/failover_lsf_admin_perspective.html

Update and hotfix considerations of a SAS Grid installation

A grid installation has the same need to apply hotfixes and other needed updates like maintenance updates. For maintenance updates there is nothing that can be influenced. The update process is defined inside the SDW. The SDW updates the installation and the configuration files. In a grid environment binaries can be in the shared file system or on the local file systems on each node. The configuration, data and naturally the SAS code has to be located in the shared file system. If the goal is to install binaries local on each node, the update of the binaries can be done sequentially. In this case there is a master that is updated with the SAS update process and from this master the binaries are synchronized (e.g. with rsync) to the other members of the grid. This process combined with a stop and start of the SAS processes on the updated nodes allows always some nodes to be available for computation. As long as there are no updates to the configuration this process allows a rolling update to the grid servers.

Conclusion

The flexibility of the Grid technology allows configuration for several different use cases. Like high performance computing (HPC) configurations or also the approach of a consolidation platform in a multi-tenant approach.

The paper introduces an approach to organize the design decision based on the expected workload and requirements for capacity, performance and availability of the service. Depending on the type of the data source systems additional considerations are necessary especially Hadoop has often additional requirements like the Kerberos enablement of the cluster.

The reader should be aware that the grid technology is not only SAS. The SAS Platform Suite with LSF has a lot of options and possible configurations. Based on the decisions which use cases and which security requirements have to be supported for the grid environment the configuration has to be adjusted.

Contact Information

Comments and questions are valued and encouraged. Contact the authors at:

Jan Bigalke

Allianz Managed Operations & Services SE

jan.bigalke@allianz.com

Greg Nelson

ThotWave Technologies

greg@ThotWave.com