Paper SAS2155-2018

SAS® Grid Manager for High Availability: Implementation Best Practices

Edoardo Riva, SAS Institute Inc.

ABSTRACT

SAS customers appreciate SAS® Grid Manager for the many benefits that it provides to their SAS® environment. Enhancing the availability of their critical services is certainly one of the most popular benefits. This paper presents the best practices for implementing highly available services and leveraging the new failover orchestration capabilities provided by the latest release of SAS Grid Manager.

INTRODUCTION

SAS Grid Manager provides you many benefits, including:

- workload balancing
- parallel processing
- job scheduling
- flexibility with scaling the architecture
- high availability

Every customer's requirements are unique. However, almost all take advantage of the high-availability capabilities. In fact, in SAS Grid Manager environments, services are distributed among multiple machines, which increases the importance of keeping them under automated monitoring and management. The shared nature of a centralized grid environment makes it more business critical and increases the potential damage caused by any failure. For this reason, services must be always available, even if some of the machines hosting them become unavailable.

SAS GRID MANAGER AND HIGH AVAILABILITY

SAS Grid Manager can increase the availability of your SAS environment components at different levels.

- Architecture and hardware that make up the grid.
- Software that operates the grid.
- Critical services that are running in the grid.
- SAS programs and applications that are running in the grid.

The lack of any of these components can result in a user's inability to work at a stated level.

Architecture

Multiple machines eliminate single points of failure from a hardware perspective. SAS Grid Manager decouples the infrastructure from applications: users submit jobs to the grid, not to host_42. Because of this, if a node becomes unavailable, other nodes can take over the workload. It becomes possible to shut down one compute server for maintenance without disrupting business operations.

Grid Software

The software at the core of SAS Grid Manager—LSF—is inherently highly available. During the deployment, you can configure a list of master candidates. In the case of failure of the current master host, another running LSF master candidate's host automatically becomes the new master, as depicted in Figure 1.

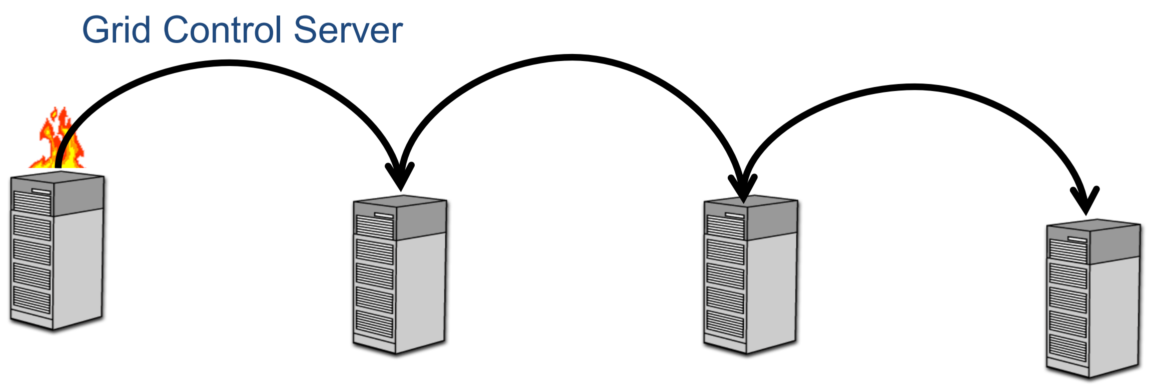


Figure 1. LSF master failover

Critical Services

The capability to monitor critical services has always been one of the most sought-after benefits that SAS Grid Manager provides. SAS Grid Manager can detect if a service fails or if the machine on which services are running fails. It automatically starts the services on a failover host. This ensures that critical services remain available to clients without any manual intervention.

SAS Programs

Even if the entire hardware and software infrastructure is highly available, your users will complain if their jobs do not complete for any reason.

The SAS programs must complete in a timely manner, even if something happened that caused them to fail. For a SAS program that takes a long time to run, this means that the program cannot be required to restart from the beginning if it ends prematurely.

SAS Grid Manager provides options to specify that SAS programs submitted to the grid are automatically restarted from the point where they stopped if they end before completion.

For the rest of this paper, the sole focus is on the third of the above topics—high availability for critical services.

SERVICES AND DEPENDENCIES

The list of services that could benefit from high availability depends on the products or solutions that you have deployed with SAS Grid Manager. In a typical SAS® Enterprise BI Server scenario, you might have the following services:

- SAS® Metadata Server
- SAS Object Spawner
- SAS® OLAP Server
- SAS® Web Infrastructure Platform Data Server
- SAS Distributed In-Process Scheduler Job Runner
- Platform Process Manager
- Platform Grid Management Service
- Web application tier components, such as:
    - SAS Cache Locator
    - SAS JMS Broker

  - SAS Web Server
  - SAS® Web Application Server
  - SAS® Environment Manager Server
- Agents, such as:
  - SAS Environment Manager Agent
  - SAS Deployment Agent

Once you configure your grid to provide high availability to services, their start-up and shutdown is managed by the grid as well. Therefore, it is important to enforce the proper sequence in operating them and to respect service dependencies.

SAS 9.4 Intelligence Platform: System Administration Guide, Fourth Edition lists the required dependencies in the chapter "Operating Your Servers." They are summarized in Table 1.

Start Order	Server or Service	Tier	Dependencies
1	SAS Metadata Server	Server tier	
2	SAS Web Infrastructure Platform Data Server	Server tier	
3	SAS OLAP Server	Server tier	SAS Metadata Server
4	SAS Object Spawner	Server tier	SAS Metadata Server
5	SAS/SHARE® server	Server tier	SAS Metadata Server
6	SAS/CONNECT® spawner	Server tier	SAS Metadata Server
7	SAS® Deployment Tester server	Server tier	SAS Metadata Server
8	SAS Distributed In-Process Scheduler Job Runner	Server tier	SAS Metadata Server
9	SAS JMS Broker	Middle tier	
10	SAS Cache Locator	Middle tier	
11	SAS Web Server	Middle tier	
12	SAS Web Application Server	Middle tier	SAS Cache Locator, SAS Web Infrastructure Platform Data Server, SAS Metadata Server
13	SAS Environment Manager Server	Middle tier	SAS Web Infrastructure Platform Data Server, SAS Web Application Server
14	SAS Environment Manager Agent	Server and middle tier	
15	SAS Deployment Agent	Server and middle tier	

Table 1. Service dependencies for a typical SAS Enterprise BI Server deployment

The original table does not explicitly include a dependency for the SAS Web Application Server on the SAS Metadata Server, because that is implied in the machine startup order which calls for operating the Server tier before the Middle tier. Here we wrote it explicitly for better clarity.

SAS GRID MANAGER MODULE FOR SAS ENVIRONMENT MANAGER

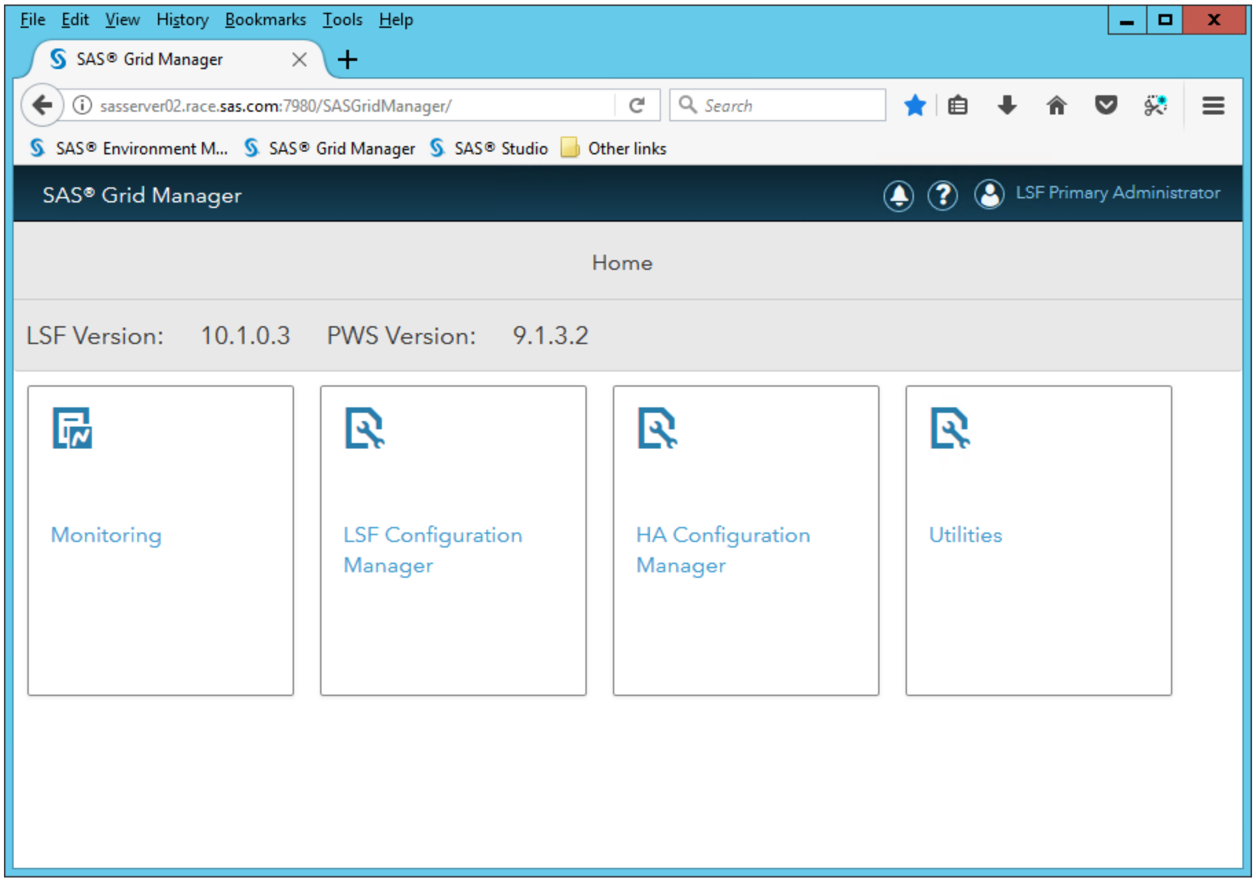
You can use the SAS Grid Manager module for SAS Environment Manager to manage and configure resources on a SAS grid, including high-availability applications.

First introduced with SAS 9.4M2 with a limited set of capabilities, it has evolved over its releases to become the tool of election for grid administrators.

Display 1 shows the home page of the SAS Grid Manager module. The four tiles group its capabilities.

When you log on with the LSF Primary Administrator user credentials, you can:

- Perform actions on grid resources (such as opening and closing hosts and activating and deactivating queues).
- View information for the cluster, grid hosts, queues, and jobs.
- Modify the configuration information for the LSF cluster, including definitions for hosts, queues, users, administrators, resource limits, and cluster parameters.
- Manage high-availability applications.
- Modify the configuration information for high-availability applications.
- Review records in the LSF audit log database.
- Update the LSF license.

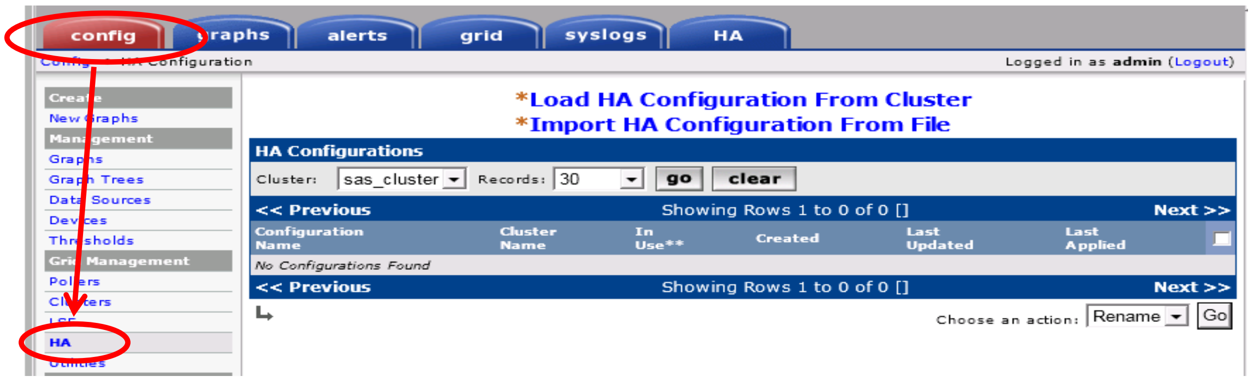


LSF Version: 10.1.0.3 PWS Version: 9.1.3.2

Display 1. SAS Grid Manager module for SAS Environment Manager home page

HA CONFIGURATION MANAGER

You use the HA Configuration Manager module to create and maintain HA configurations. For example, these could include sets of definitions of how to start, stop, and monitor services. Initially delivered with SAS9.4M3, in that release, it is nearly equivalent to the **HA Configurations** page available in the configuration pane of RTM. You can compare the home pages for both applications in Display 2 and Display 3.



Config -> HA Configuration

Logged in as **admin** (Logout)

Create
New Graphs
Management
Graphs
Graph Trees
Data Sources
Devices
Thresholds
Grid Management
Pollers
Clusters
LSF
HA
Utilities

*Load HA Configuration From Cluster
*Import HA Configuration From File

HA Configurations

Cluster: sas_cluster Records: 30 go clear

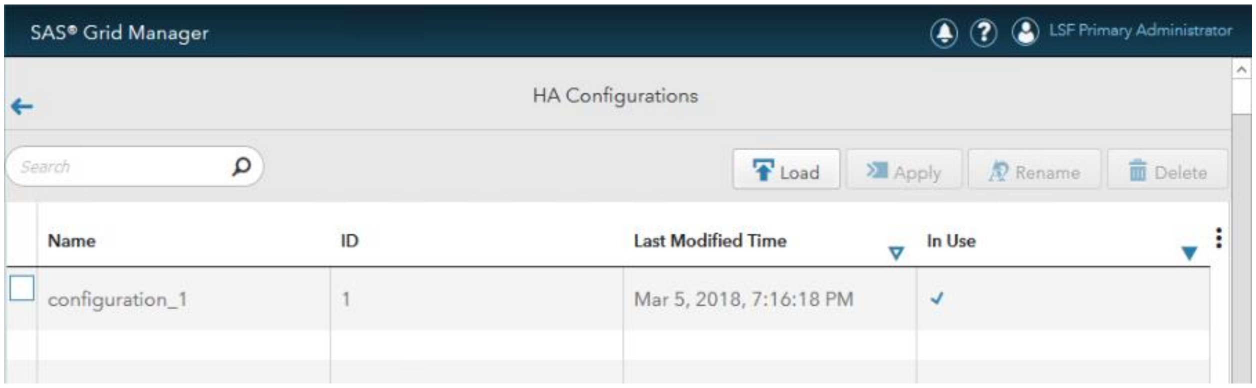
<< Previous Showing Rows 1 to 0 of 0 [] Next >>

Configuration Name	Cluster Name	In Use**	Created	Last Updated	Last Applied
No Configurations Found					

<< Previous Showing Rows 1 to 0 of 0 [] Next >>

Choose an action: Rename Go

Display 2. RTM home page to manage HA configurations



Name	ID	Last Modified Time	In Use
configuration_1	1	Mar 5, 2018, 7:16:18 PM	✓

Display 3. SAS Grid Manager home page for HA Configurations

The referenced paper SAS® Grid Administration Made Simple describes in detail how to use it.

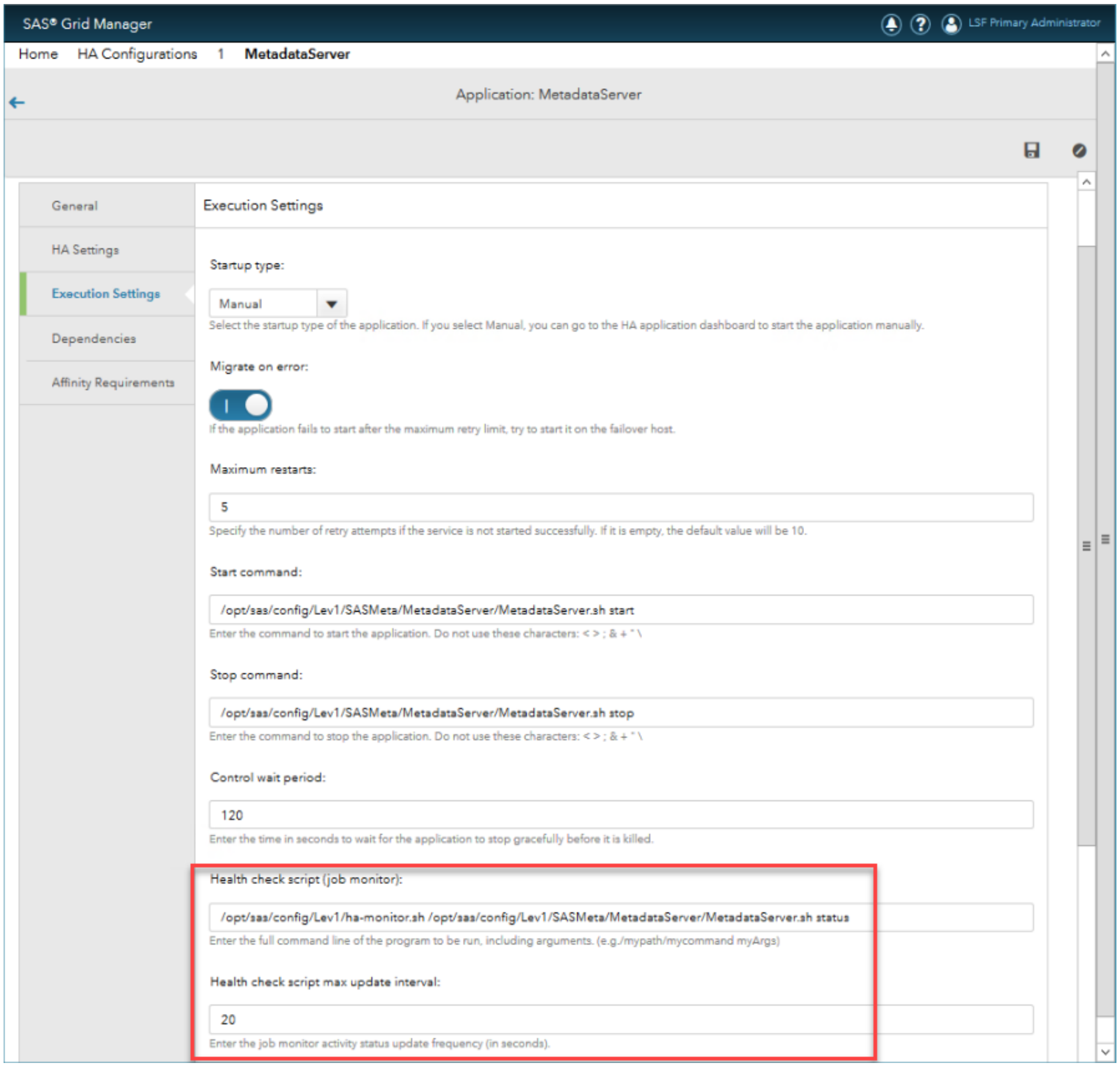
With SAS 9.4M5, the HA Configuration Manager has been enhanced to support new capabilities delivered by LSF 10, such as:

- custom service monitoring via a health check script
- enhanced management of service dependencies
- application affinity

HEALTH CHECK SCRIPTS

When you define a highly available service, the configuration page requires you to enter the scripts to start and stop the service. These scripts are used by the grid to control the service on your behalf.

The SAS 9.4M5 release includes support for an optional third script, which is used to monitor the health status of the service as shown in Display 4.



Execution Settings

Startup type: Manual

Select the startup type of the application. If you select Manual, you can go to the HA application dashboard to start the application manually.

Migrate on error: I

If the application fails to start after the maximum retry limit, try to start it on the failover host.

Maximum restarts: 5

Specify the number of retry attempts if the service is not started successfully. If it is empty, the default value will be 10.

Start command: /opt/sas/config/Lev1/SASMeta/MetadataServer/MetadataServer.sh start

Enter the command to start the application. Do not use these characters: < > ; & + " \

Stop command: /opt/sas/config/Lev1/SASMeta/MetadataServer/MetadataServer.sh stop

Enter the command to stop the application. Do not use these characters: < > ; & + " \

Control wait period: 120

Enter the time in seconds to wait for the application to stop gracefully before it is killed.

Health check script (job monitor): /opt/sas/config/Lev1/ha-monitor.sh /opt/sas/config/Lev1/SASMeta/MetadataServer/MetadataServer.sh status

Enter the full command line of the program to be run, including arguments. (e.g./mypath/mycommand myArgs)

Health check script max update interval: 20

Enter the job monitor activity status update frequency (in seconds).

Display 4. The new health check script parameters on the Execution Settings page

The purposes of this script are the following:

- If you leave the field empty, the grid behaves just like it did in the previous releases. It monitors the service at the operating-system level (i.e., it asks the operating system whether the main process of the service is still there). Although this method is effective in detecting processes that die or machines that fail, it cannot detect the case in which a service is still there but has become unresponsive.

- If you specify a script, the grid runs it immediately after starting the service. The script is expected to output a status code once every *N* seconds, as specified in the max update interval field. Table 2 shows the supported codes and their meaning. This new capability can be more effective in detecting services that have become unresponsive, but it might cause unwanted failover if the max update interval parameter is not properly configured. If, for example, the service is busy servicing user requests, it can simply take longer than usual to respond to status requests. Is this a case of failure or should the time-out simply be increased? It is your call to decide case by case. What's important is that the software gives you the tools to implement whatever decision you make.

Status code	Meaning	How does the grid react
TENTATIVE	The service is still starting or initializing.	Wait.
READY	The service is up and running.	Mark the service as ACTIVE and, if configured, start dependent services.
ERROR	The service has failed.	Issue the service stop command, then restart it.
END	The monitor script exits and will stop monitoring the service.	The service remains in the current state.

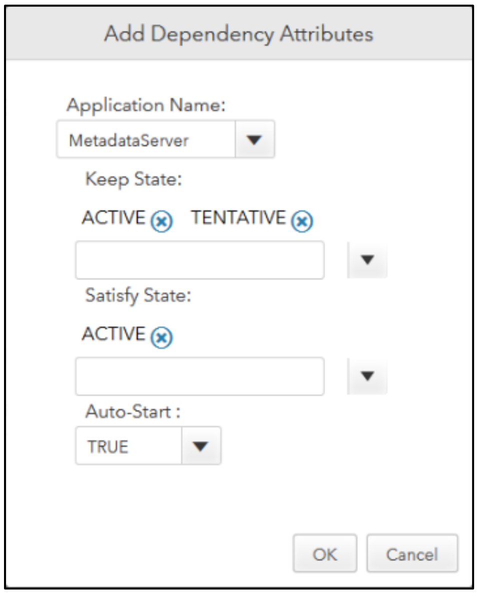
Table 2. Health check script (job monitor) status codes

DEPENDENCIES

Basic dependency management is usually not sophisticated enough to be used with SAS services. It is not enough to initialize services in dependency order. Some services require time to fully initialize. For example, when the SAS Metadata Server is configured as a dependency for the SAS Object Spawner, the grid will always start the SAS Metadata Server before the SAS Object Spawner. If the SAS Metadata Server takes time to initialize before it begins listening for connections, when the SAS Object Spawner is started, it might not be able to connect to the SAS Metadata Server as expected.

Also, if the start-up sequence is the only type of dependency that can be defined, you lack a way to address the case where the failover of a service requires an automatic restart of a dependent service. For example, during an automatic restart of a failed SAS Metadata Server, other dependent services have to be stopped and restarted as well.

The SAS 9.4M5 release supports this traditional dependency—named "onStart"—and then adds a new one named "Conditional". Display 5 shows how to configure this new "Conditional" dependency.


Add Dependency Attributes

Application Name: MetadataServer

Keep State: ACTIVE TENTATIVE

Satisfy State: ACTIVE

Auto-Start : TRUE

Display 5. Conditional dependency

Here, **Keep State** and **Satisfy State** are defined as follows:

- The service being defined will be started only when the one that it depends on is in one of the satisfy states. For example, while the SAS Metadata Server is initializing, its status can be

TENTATIVE before becoming ACTIVE. Your grid will wait and start this dependent service only when the SAS Metadata Server has become ACTIVE.

- The service will be kept running only when the one that it depends on is in one of the keep states. For example, as soon as the SAS Metadata Server is not ACTIVE or TENTATIVE (i.e., it is probably stopped or dead), then your grid will also stop this dependent service.

Based on preliminary testing, these new capabilities are still not enough to cover all the dependency requirements of SAS services, as detailed in High Availability Services with SAS Grid Manager. However, they are additional tools that you can leverage to simplify the design of a correct orchestration.

APPLICATION AFFINITY

This configuration setting lets you define service affinities or anti-affinities. Application affinity specifies whether the grid should start the selected service on the same machine or on different machines with respect to other services.

- The selected application must run on the same server as the other applications. This is known as "hard affinity."
- The selected application must run on a different server than the other applications. This is known as "hard anti-affinity."
- The selected application should run on the same server as the other applications, if possible. This is known as "soft affinity."
- The selected application should run on a different server than the other applications, if possible. This is known as "soft anti-affinity."

Affinity and anti-affinity are not required by default when configuring high availability for SAS services. They could be useful for other services or to address specific use cases or business requirements.

Here is one example. You have defined a virtual IP to redirect client connections to a specific service (for example, the SAS Metadata Server in an active-passive scenario). A hardware load-balancer monitors the back-end cluster to assign that virtual IP to the machine where the grid has started the service. If you configure additional services to use that virtual IP, then they should all be moved to the failover machine as soon as the first service—the SAS Metadata Server, in this example—fails over. Previously, this required custom scripting. Now, you can configure these additional services to have a hard affinity with the service monitored by the load-balancer (i.e., the SAS Metadata Server). The grid will take care of moving them to the failover machine all at once.

ADDITIONAL TIPS

LICENSING CONSIDERATIONS

Before designing and implementing high availability in your environment, be sure to verify your licensing terms. The current SAS Grid Manager licensing prescribes that you should count and include in the license the processor cores of any machines used to run services, such as the SAS Metadata Server, for which SAS Grid Manager will provide high availability.

Usually, this has no implications for services running on compute nodes because these have already been included in your SAS Grid Manager license.

Instead, if you are planning to use SAS Grid Manager to provide high availability to SAS Metadata Servers deployed on dedicated machines or to services deployed on middle-tier machines, you should contact your SAS representative to verify the compliancy to your licensing terms.

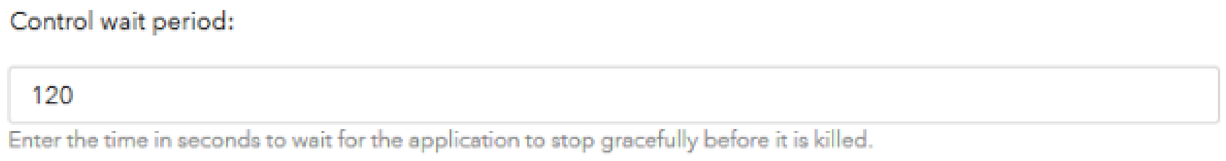
CONTROL WAIT PERIOD

On the **Execution Settings** page of the high-availability service definition form, there is one parameter called **Control wait period**. This specification is often overlooked, but it can play a critical role in the health of a monitored service. In fact, when the grid issues a stop command, it waits for the monitored service to properly shut down up to the control wait period. After that, if the controlled application is still running, it kills it. For some services, such as the SAS Object Spawner, that is not an issue. If it is still running after 30 seconds—the default value for this option—it probably means that the process is stuck and that it can be killed.

On the contrary, the SAS Metadata Server during the shutdown procedure must perform some activities before actually stopping (such as synchronize its memory to disk, commit pending journal entries, close repositories, update indexes, and so on). This means that on a loaded production system, it might take more than the default 30 seconds to stop. You do not want anything to kill it in the meantime because this introduces the risk of metadata repository corruption!

The same holds true for the SAS Web Infrastructure Platform Data Server. It's a transactional database and you would not kill it while it's flushing its logs during shutdown.

For these services, you should increase the **Control wait period** from the default, even if there is no magic number to put in that field. An approach that has proven successful is to consult the past logs of the service. For example, the SAS Metadata Server writes in its logs how long it takes to shut down each time. After getting a ballpark estimate from there, add some additional safety time to this.


Control wait period:

120

Enter the time in seconds to wait for the application to stop gracefully before it is killed.

Display 6. Control wait period increased to two minutes

ONLY ONE CONTROLLER!

After your grid takes control of a service, there will be conflicts if the same service is managed by any other means, such as using the sas.servers script, Windows services, or UNIX INIT scripts.

It's usually a good practice to adjust anything that was in control of the services that are now under grid control to avoid overlapping commands.

Services initially configured to automatically start at boot time.

An example of this type of service is the Platform Process Manager or the Platform Grid Management Service. In a normal deployment, they are configured for auto-start as part of the machine start-up.

On a Linux system, you can disable their INIT scripts with:

```
sudo rm /etc/rc?.d/*jstartup
sudo rm /etc/rc?.d/*lsfwm
```

Services included in the sas.servers script.

After the grid takes control of a SAS service, you want to modify how the default sas.servers script manages it. There are two possible options.

1. Modify the service start, stop, or status subroutines to call EGO start, stop, or status commands instead of the actual service commands. This solution requires careful planning and implementation. It is described in High Availability Services with SAS Grid Manager under "SAS Service Startup Scripts."
2. Modify the sas.servers script to disable any call to the EGO-managed services as documented in SAS(R) 9.4 Intelligence Platform: System Administration Guide. After that, any start, stop, or status commands should be issued using the EGO command-line tool EGOSH or using the SAS Grid Manager module for the SAS Environment Manager web interface.

As an example of implementing the second method, you can disable the SAS Object Spawner with the following steps:

1. Log on as the SAS Installer user (in this example, sasinst).
2. Change the keyword ENABLE to DISABLE in the file <*SAS-configuration-directory*>/Lev1/ObjectSpawner/ObjectSpawner.srv.
3. Regenerate the sas.servers script.

You can use the following commands to do it:

```
cd /opt/sas/config/Lev1/
sed -i.bak 's/^ENABLE/DISABLE/' ./ObjectSpawner/ObjectSpawner.srv
./generate_boot_scripts.sh
```

Output 1 shows the expected result of the execution of the above commands:

```
[sasinst@sasserver01 ~]$ cd /opt/sas/config/Lev1/
[sasinst@sasserver01 Lev1]$ sed -i.bak 's/^ENABLE/DISABLE/'
./ObjectSpawner/ObjectSpawner.srv
[sasinst@sasserver01 Lev1]$ ./generate_boot_scripts.sh
./generate_boot_scripts.sh: The SAS server configuration file
"/opt/sas/config/Lev1/ObjectSpawner/ObjectSpawner.srv"
specifies the DISABLE keyword, so this SAS server instance
will be ignored.

...

2 SAS servers were explicitly disabled in their .srv files.
0 SAS servers were disabled by an apparent error in their .srv files.
```

Output 1

CONCLUSION

Maintaining your SAS environment that is up and running is a task that can be greatly simplified with SAS Grid Manager high-availability capabilities. The SAS 9.4M5 release introduces new options to cover even more use cases and reduce the requirement to implement custom scripting.

REFERENCES

Doninger, Cheryl, Zhiyong Li, and Bryan Wolfe. 2014. “Best Practices for Implementing High Availability for SAS 9.4.” *Proceedings of the SAS Global Forum 2014 Conference.* Cary, NC: SAS Institute Inc. Available at https://support.sas.com/rnd/scalability/grid/305_Doninger_FinalPaper.pdf.

Jackson, Robert, and Scott Parrish. 2014. “High Availability Services with SAS Grid Manager.” Cary, NC: SAS Institute Inc. Available at http://support.sas.com/rnd/scalability/grid/HA/GridMgrHAServices.pdf.

Parrish, Scott, Linda Zeng, and Paula Kavanagh. 2016. “SAS Grid Administration Made Simple.” *Proceedings of the SAS Global Forum 2016 Conference.* Cary, NC: SAS Institute Inc. Available at http://support.sas.com/rnd/scalability/grid/SGF16_SASEV_Grid.pdf.

RECOMMENDED READING

- SAS Institute Inc. 2018. *Grid Computing in SAS 9.4, Fifth Edition*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/documentation/onlinedoc/gridmgr/index.html.
- SAS Institute Inc. 2018. *SAS 9.4 Intelligence Platform: System Administration Guide, Fourth Edition*. Cary, NC: SAS Institute Inc. Available at http://documentation.sas.com/?docsetId=bisag&docsetVersion=9.4.
- Pan, Helen. 2013. “Best Practices for Deploying Your SAS Applications in a High-Availability Cluster.” *Proceedings of the SAS Global Forum 2013 Conference.* Cary, NC: SAS Institute Inc. Available at

http://support.sas.com/resources/papers/proceedings13/469-2013.pdf.

CONTACT INFORMATION

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

Edoardo Riva
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
+1 919 531 7293
edoardo.riva@sas.com
http://www.sas.com

```
#!/bin/bash
# Job Monitor Sample - expect a status command as the first parameter
# i.e. /opt/sas/config/Lev1/ha-monitor.sh /opt/sas/config/Lev1/SASMeta/MetadataServer/MetadataServer.sh status

STATCMD=
if [ $# -ne 0 ]
then
   STATCMD="$1"
   shift
   if [ ! -e "${STATCMD}" ]
   then
      echo "${STATCMD} must be the full path to the service status command." >&2
      exit 2 # LSB compliant status; invalid or excess argument(s)
   fi
   if [ ! -x "${STATCMD}" ]
   then
      echo "${STATCMD} cannot be executed." >&2
      exit 4 # LSB compliant status; user had insufficient privilege
   fi
else
   echo "No status command was specified" >&2
      exit 2
fi
#create a descriptive name by taking the status command and stripping the final 3 characters - assuming they are ".sh"
name=`basename ${STATCMD}`
name=${name:-3}

now=`date +%Y-%m-%d@%H:%M:%S`
#DEBUG=1

debug()
{
   # When the env var DEBUG is set, log extra output to stderr
   if [ ! -z "${DEBUG}" ]
   then
      for line in "$@"
      do
         echo "${now} monitor-${name}: ${line}" >&2
#          echo "${now} monitor-${name}: ${line}" >> /opt/sas/sharedwork/monitor.log
      done
   fi
}

debug "Status cmd was specified: '${STATCMD}'"
debug "Status cmd arguments ($#):" "--$@--"
debug "EGO_ACTIVITY_PID=$EGO_ACTIVITY_PID"

#emit the first status - initially the service is "TENTATIVE"
echo "UPDATE_STATE 'TENTATIVE'"

#If we knew the service is already OK, we could simply set the service instance to ready state which triggers activity
state to RUN
#echo "UPDATE_STATE 'READY'"
```

```
#Assume the update interval is 2 second less than max
#If max update interval is not set in service profile, set updateInterval to 2 second
if [ -z $JOB_MONITOR_MAX_UPDATE_INTERVAL ] || [ $JOB_MONITOR_MAX_UPDATE_INTERVAL -le 2 ]
;then
   let updateInterval=2
else
   let updateInterval=$JOB_MONITOR_MAX_UPDATE_INTERVAL-2
fi
debug "updateInterval=$updateInterval"

#sleep once to give the service time to initialize
sleep $updateInterval
echo "UPDATE_STATE 'TENTATIVE'"

#Start loop to monitor service status
#Check if there are issues or not and update status to ERROR. Otherwise it will continue to RUN
while [ 1 ];
do
    startLoop=$(date +"%s")
    debug "Ready to run monitor"
    stat_out=`"${STATCMD}" "$@"` # NOTE: passing along additional args "$@" to status cmd
    mon_rc=$?
    debug "--Service ${name} status (${mon_rc})..."
    debug "stat_out=$stat_out"
    [ ${mon_rc} -ne 0 ] && break
    # SAS init scripts status functions exit 0 when server is stopped, must check the output
    [ `echo "${stat_out}" | grep -c "is stopped$"` -ne 0 ] && break
        echo "UPDATE_STATE 'READY'"
        debug "STATE READY"
        # if the monitor command completed faster than updateInterval, then sleep.
        endLoop=$(date +"%s")
        elapsed=$(( $endLoop - $startLoop ))
        [ $elapsed -le $updateInterval ] && sleep $(( $updateInterval - $elapsed ))
done

#if we get here, the status command failed.
echo "UPDATE_STATE 'ERROR'"
echo "END"
debug "exiting from monitor"
```