Paper 1726-2018

High-availability services in enterprise environment with SAS® Grid Manager

Andrey Turlov, Allianz Technology SE; Nikolaus Hartung, SAS

ABSTRACT

Many organizations, nowadays, rely on services for the upkeep of critical day-to-day business operations. Consequently, these services must be constantly available and accessible, even when underlying infrastructure experiences failures. SAS® Grid Manager provides a high-availability SAS® Enterprise Business Intelligence platform, out of the box, based on a multi-machine architecture with LSF® and service failover. In a large multinational organizational setting, the base installation and configuration of LSF® and SAS® should be extended to provide HA services according to company requirements (e.g. server topology, resource management, multi-tenancy). This paper presents such cross-component configuration samples, tips and tricks on all layers of SAS®, LSF®, EGO® that come together to provide better and reliable HA services for global enterprises.

INTRODUCTION

High-availability (HA) is a very important topic in today's enterprises. Core operational systems need to be online and downtimes, scheduled or unscheduled, almost always incur additional cost. HA eliminates a single point of failure by adding redundancy. It applies methods to detect failures and switchover from one system to another without or with minimal service interruption (basic principles and further reading related to HA can be found here [1]). Furthermore, when enterprises deploy their whole systems or move parts to distributed (cloud) architectures, these systems should be configured to take full advantage of the existing multi-layer HA capabilities provided by distributed architectures.

High-availability on the SAS® Enterprise Business Intelligence (EBI) Platform can be seen from different perspectives:

- Metadata Server HA
- Web Application Server HA
- WIP Server (PostgreSQL) HA
- SAS® Services (Object Spawner, OLAP Server, Connect Spawner) HA

For each of these services there are already many techniques available to provide HA or fault tolerance. Techniques such as Metadata Server Cluster (for more information see [2]), Web Application horizontal or vertical cluster or Postgres "hot" standby (for more information see [3]) have already been explained in similar papers. The focus of this paper, however, is HA for two SAS® Services the SAS® ObjectSpawner and SAS® OLAP Server used in a SAS® Grid computing environment. In addition, many other services, like Connect Spawner or additional 3rd party services, can be managed in a similar way.

A SAS® Grid computing environment allows distribution of workloads of various SAS® tasks and services among multiple servers in a cluster (for more information see [4]). To provide efficient resource allocation, policy management, and load balancing SAS® Grid computing platform depends on multiple components of which this paper focusses on:

- IBM® Platform™ LSF® (Load Sharing Facility) is a workload management platform for distributed environments. The platform manages resource requirements and performs load balancing across machines in a grid environment by dispatching jobs submitted to a cluster and returning the status of each job. It provides a comprehensive set of intelligent, policy-driven scheduling features that enable optimal utilization of compute infrastructure resources and ensure optimal application performance.

1 https://en.wikipedia.org/wiki/High_availability
2 http://go.documentation.sas.com/?docsetId=bisag&docsetTarget=n0a51gzoaey35en16vcx47li88xb.htm&docsetVersion=9.4&locale=de
3 https://support.sas.com/resources/papers/Managing-WIP-DataServerforHA.pdf
4 http://support.sas.com/resources/papers/proceedings16/11562-2016.pdf

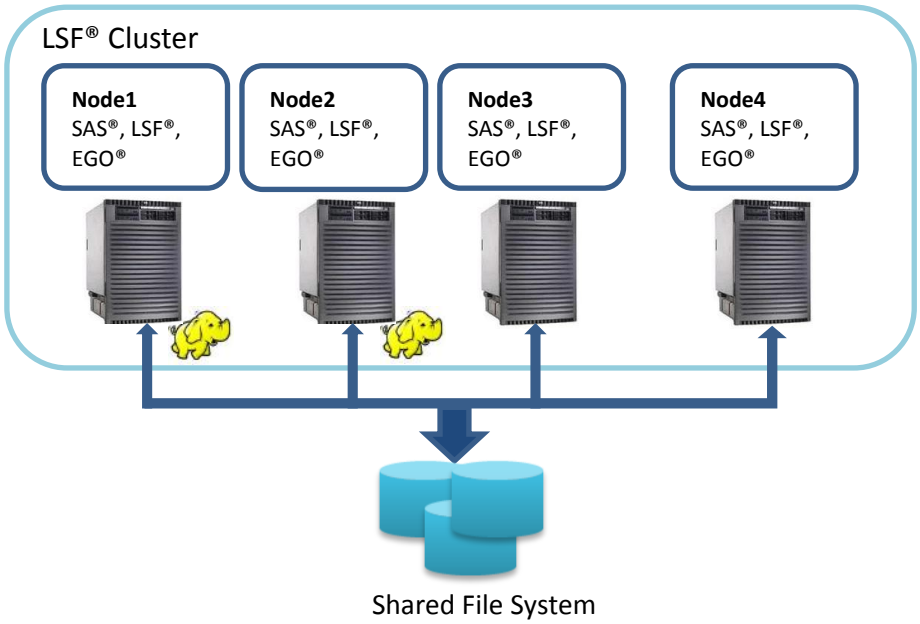
- IBM® EGO® (Enterprise Grid Orchestrator) allows developers, administrators, and users to treat a collection of distributed resources on a distributed computing infrastructure (cluster) as parts of a single virtual computer. EGO® monitors and supports critical service failover in a HA environment; restarting services if they stop, and starting services on a failover host when needed.
- The SAS® Metadata Server is a centralized resource for storing, managing, and delivering metadata for SAS® applications across the enterprise. It is a multi-user server that serves metadata from one or more SAS® Metadata Repositories to all of the SAS® EBI Platform client applications.

Although, SAS® Grid computing environment is installed as a fully integrated platform, configuration is distributed among different layers and files, that must be kept in sync and in compliance with each other. The following use case illustrates these challenges and highlights possible solutions.

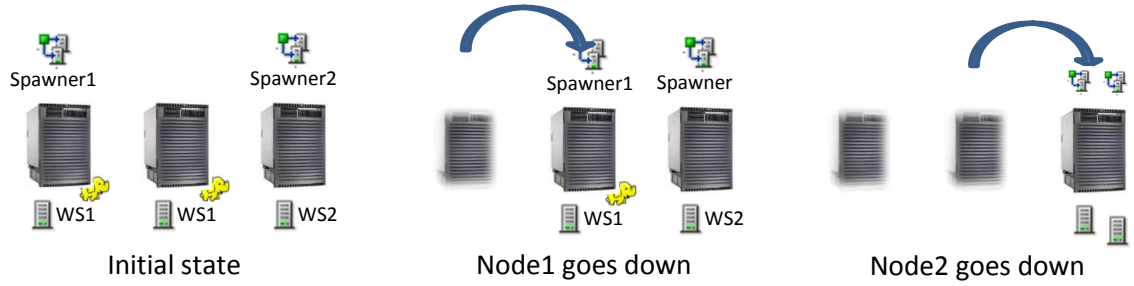
USE CASE

An enterprise with multiple independent business units would like to create a flexible and high-available SAS EBI platform, that is able to deliver services in case of scheduled (updates, patches, e.t.) or unscheduled downtimes. To isolate the resource usage (CPU, memory, disk I/O, etc.) and to enable separate business units to use different versions of interfaces like Hadoop interfaces, database client versions, and 3d party products, they need to operate on different compute servers.

In the case of a failure of one of these servers the related services must be automatically restarted preferably on a server with the same software configuration to allow users to retain a fully functional access to the SAS environment and the connected systems. To accomplish these goals the following example system architecture is proposed.



The LSF® cluster contains a metadata server managed by the EGO® service controller on Node4 and a pool of 3 compute nodes. The three compute nodes are used by two SAS® Server contexts which represent different business units. Node1 and Node2 contain a Hadoop client installation and are assigned to business unit 1 and Node3 to business unit 2. Context 1 has one instance of an Object Spawner, but allows to start SAS sessions (interactive or batch) on Node1 or Node2. If one of the nodes goes down and the Object Spawner process fails, EGO® will start a new Spawner process on the other available node. In worst case, Node1 and Node2 are down together, limited access is provided to the SAS® system (in this case without Hadoop access) as the Spawner will be started on Node3. The Object Spawner for context 2 is started on Node3 and submits SAS® sessions to the same node. In a failover case it can use any compute node in the cluster.



This configuration setup enables the system to remain available, even in the event of simultaneous failure of two nodes.

HA CONFIGURATION WITH SAS GRID

LSF® AND EGO® CONFIGURATION

LSF® and EGO® can be configured to start different services on separate predefined groups of servers. The separation of components on nodes can be at the SAS® EBI architectural level, for e.g.

- Metadata Server is started on the metadata server node and Spawner is started on the application servers pool

or defined by the enterprise business topology for e.g.

- different server contexts are separated on different application servers.

To manage and accomplish such segregations, resource groups have to be defined and mapped in the cluster host configuration. The following simplified sample shows the definition and mapping within the LSF® configuration files.

The resource names are strings defined in the Resource section of the lsf.shared file. Recommendation is to keep the names simple, no space, nor underscores or dashes. They identify a group, in this example there are groups for:

- mdh = metadata host group
- ch = compute host group
- context1 and context2 = context groups

each definition is of the type Boolean which will identify a host is available or not.

```
Begin Resource
RESOURCENAME  TYPE    INTERVAL INCREASING  DESCRIPTION         # Keywords
mdh           Boolean ()       ()          (Metadata hosts)
ch            Boolean ()       ()          (Compute hosts)
context1      Boolean ()       ()          (Context1 hosts)
context2      Boolean ()       ()          (Context2 hosts)
End Resource
```

The mapping from a resource to host is done in the lsf.cluster.<cluster_name> file:

```
Begin    Host
HOSTNAME  model  type  server r1m  mem  swp  RESOURCES    #Keywords
Host4     !      !     1      3.5   ()   ()   (mdh)
Host1     !      !     1      3.5   ()   ()   (ch context1)
Host2     !      !     1      3.5   ()   ()   (ch context1)
Host3     !      !     1      3.5   ()   ()   (ch context2)
End      Host
```

For more information on the format and limitations of LSF® configuration files see IBM® configuration resources[5].

[5] https://www.ibm.com/support/knowledgecenter/en/SSETD4_9.1.3/lsf_config_ref/part_files.html

The third part to the full picture is EGO® that controls the start and stop of HA services. Each service is defined in an EGO® service definition xml file. The important part of the configuration and usage of the resource group is shown here:

```
<sc:MaxInstancesPerHost>1</sc:MaxInstancesPerHost>
……
<ego:ResourceRequirement>select('context1' || 'ch') order (0-'context1' +
ut)</ego:ResourceRequirement>
```

During the start of an EGO® service, the service definition xml files with the configured directives are processed and interpreted. The MaxInstancePerHost and ResourceRequirements attributes of the EGO® service xml will force EGO® to start one instance of context1 services on one of the available nodes in context1 resource group (Node1 or Node2). In the case of a failover event the instance is restarted in the same group. In the worst case, when there is no server available in the requested group, the instance is restarted in any of the available compute nodes in the cluster, because the ch-group is configured on any compute host. The select statement determines only nodes with a resource configured for "context1" or compute hosts "ch" and if they are available for the cluster. Additionally, the order directive sorts the result set by the node with the lowest utilization running context1. (more information to the select syntax can be found here [6])

To apply the latest changes to LSF®, the system need to reconfigure. To do so, execute the following commands:

- *lsadmin ckconfig*
- *lsadmin reconfig*

To view the new configuration, use the command:

- *lshosts -w*

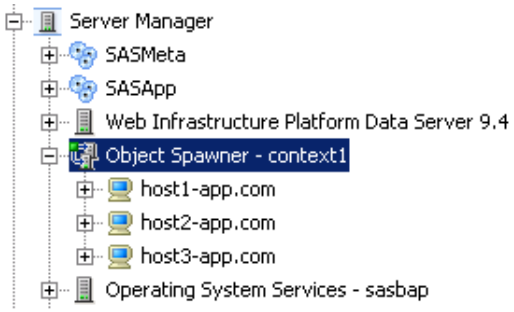
which outputs this list:

```
HOST_NAME     type     model    cpuf ncpus maxmem maxswp server RESOURCES
host1-app.com  X86_64  DEFAULT 1.0  8     64G    15.9G  Yes (ch context1)
host2-app.com  X86_64  DEFAULT 1.0  4     64G    15.9G  Yes (ch context1)
host3-app.com  X86_64  DEFAULT 1.0  4     64G    15.9G  Yes (ch context2)
host4-app.com  X86_64  DEFAULT 1.0  4     64G    15.9G  Yes (mdh)
```

The configuration above allows dynamic spread of workload into the cluster and actively switch the services from one node to another in case of failover is initiated.

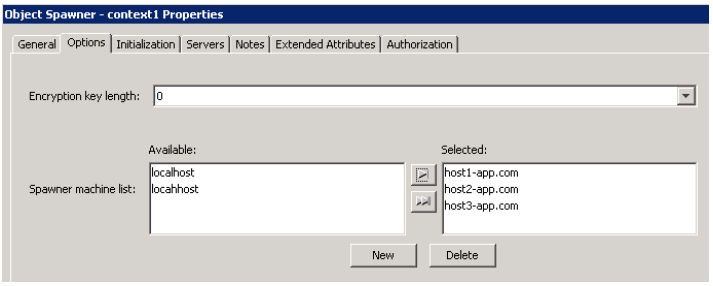
SAS® METADATA

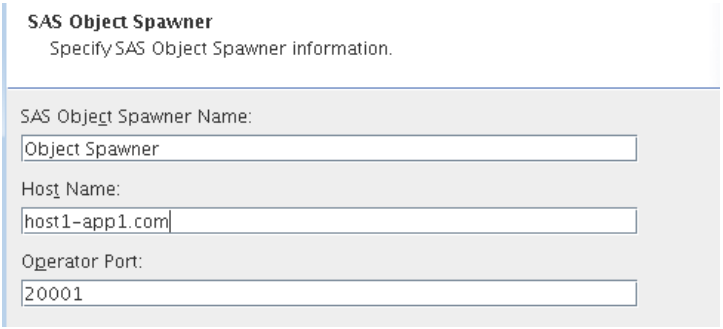
The above configuration of LSF® and EGO® start services on predefined group of hosts and can dynamically move around on different hosts. So far, so good. The management instance, however, the Metadata Server, doesn't know anything about this new dynamic service configuration. Metadata Server registers all its Spawners and server context during installation or right after when they are configured with SAS Deployment Wizard (SDW) for their purpose.



[6] https://www.ibm.com/support/knowledgecenter/SSGSMK_6.1.1/reference_sym/-R_res_req.html

After configuration the registered hostname and ports are statically assigned within the metadata repository. That applies to Object Spawner, OLAPServer Stored Process Servers (STP), Workspace Servers (WS) and Pooled Workspace Servers (PWS). More so STP, WS and PWS are statically assigned to active machines already during the installation phase.





The metadata configuration does allow all registered nodes to be added to the service configuration. At first thought this seems to solve the issue, until the first host goes down and EGO® according to the dynamic configuration starts the Object Spawner on another host. Systems are all up and everything looks good until a user tries to start an Enterprise Guide session and observers that this process can take a considerable time. According to observed behavior, some clients circle though the registered servers and try to reach them until a time out is reached. Looking back at the registered servers in the worst case a user might have to wait over a minute until EG connects to a server. The situation is even worse in DI Studio when a connection is initiated to OLAP and WS, the application does not even connect to these services throwing an hard error in the user interface.

Metadata in the Server Manager is quite static from the point of the installation and configuration of the system. But with some additional effort, it can be highly dynamic and modified based on events. The solution to the issue can be added in the process while EGO® is trying to get the Object Spawner running on another host. A small SAS® program is triggered during the execution of the service startup. During service startup it connects to the metadata server and updates the server's registration with the updated hostname. For OLAP server the executed program is applying a Hide ACT to show or hide OLAP configurations during start and stop with metadata security. DI Studio will only see the available services and the other instances are hidden.

The modification script is added in the startup process for OLAP Servers (OLAPServer_usermods.sh) and Object Spawner (ObjectSpawner.sh) in each of them a SAS® session is stared that executes code to add the new active machine (host) to the Spawner definition and also dependent server definitions (Stored Process Server, Workspace Server and Pooled Workspace Server). The script is called e.g.

- *change_spw.sh "$SPWNNAME" "$HOSTNAME" $COMMAND*

and starts a SAS® session with the code:

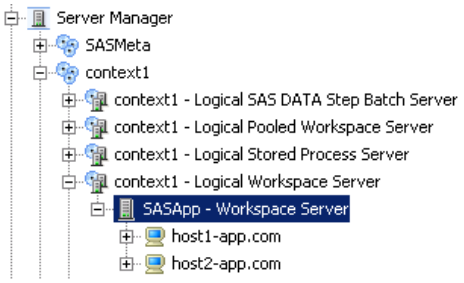
```
data _null_;
 length uri treeUri $200;
 rc=metadata_getnobj("omsobj:ServerComponent?@Name='&name.'",1, uri);
 if rc > 0 then do;
  treeRC = metadata_getnasn(uri,"SoftwareTrees", 1, treeUri);
  if treeRC > 0 then do;
   if "%upcase(&command.)" eq "STOP" then do;
     metadata_setassn(treeUri,"Members", "REMOVE", "&machineUri.");
   end;
   else do;
    metadata_setassn(treeUri,"Members", "APPEND", "&machineUri.");
   end;
  end;
 end;
run;
```

To apply a hide OLAPServer ACT the following code is executed:

```
data _null_;
 length uri treeUri $200;
 rc=metadata_getnobj("omsobj:ServerComponent?@Name='&name.'",1, uri);
 if rc > 0 then do;
  if "%upcase(&command.)" eq "STOP" then do;
    metadata_setassn(uri,"AccessControls","REPLACE","&hideact.");
  end;
  else do;
   metadata_setassn(uri,"AccessControls","REMOVE","&hideact.");
  end;
 end;
run;
```

Security relevant variables (e.g. username or password) should not be used in the UNIX scripts (e.g. change_spw.sh), as they will be propagated to the users SAS® session. This is because a SAS® session on UNIX is a child process of the Object Spawner process and inherits environment variables from its parent.

Workspace Server definition has to be excluded from dynamic changes and must contain all hosts defined for the group (context1 or context2), as the list of defined hosts is forwarded to LSF® during the start of the SAS® session.



According to the load on the servers in the list LSF® decides where to start the session. At log on the Object Spawner contains following:

```
2018-01-12 INFO: user - Request made to cluster context1 - Logical
Workspace Server
2018-01-12 INFO: user - Redirect client in cluster context1 - Logical
Workspace Server to server context1 - Workspace Server at host1-
app.com:8561
2018-01-12 DEBUG: user - Allowing redirection for grid launch server to
proceed due to object spawner being available on redirection host1-
app.com
2018-01-12 DEBUG: user - Launching server context1 - Workspace Server
(child id 10) in grid on hosts host1-app.com, host2-app.com
```

CONCLUSION

The SAS® Grid solution offers many valuable features to ensure HA within the SAS® application layer. The configuration examples described in this paper allows to close the gap between a dynamic LSF® platform and the rather static SAS® metadata server concept. This achieves a higher services availability of your SAS® environment. However, this is a step in the right direction, for full flexibility, many more techniques can be applied. Options such as Grid Option Sets, LSF® queues and host groups, internal and external sensors and time scheduling. Further information can be requested or also shared with the authors.

ACKNOWLEDGMENTS

We thank Jan Bigalke (Allianz Technology) and Nitin Nair (Allianz Technology) for critically reviewing the manuscript.

CONTACT INFORMATION

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

Andrey Turlov, Allianz Technology SE

andrey.turlov@allianz.de

Nikolaus Hartung, SAS

nikolaus.hartung@sas.com