Paper 001-2009

Achieving High Availability in a SAS® Grid Environment

Daniel Wong, Platform Computing Inc., Markham, Canada

ABSTRACT

With increasingly complex analytics, growing data volumes and tightening batch windows, businesses are facing an ever greater need for guaranteed uptime and continuity of services commonly referred to as high availability. At the heart of such a system is resilient infrastructure that provides the architecture for successfully running SAS applications without fail - scheduling and managing the execution of a wide variety of daily computing tasks across shared and networked hardware. For SAS applications, this infrastructure is based on SAS Grid Manager – grid middleware deployed specifically for SAS applications and providing critical services such that all SAS computing tasks running in the grid can be completed optimally.

SAS Grid Manager now includes high availability capabilities for all of the services that are critical to your SAS environment. This has three immediate benefits to customers who require a high availability SAS environment:

- eliminates operating system dependencies
- eliminates the requirement of a hot-standby for failover
- eliminates the expense of purchasing a third-party tool to provide high availability.

The high availability capability discussed in this paper is part of the Platform Suite for SAS Version 4.1. This is shipped as part of SAS Grid Manager 9.2 and is available for download through SAS Technical Support for use with SAS 9.1.3. This paper focuses on a SAS Grid deployment, however the concepts also apply to an environment using the Platform Suite for SAS Version 4.1 to perform enterprise scheduling on a single server.

This paper discusses the necessary steps to implement high availability for critical services in a SAS Grid environment using the SAS Metadata Server as an example. Details are also given on the built-in failover capabilities of the Platform Suite for SAS itself. Finally, the last discussion will focus on current research efforts in leveraging SAS 9.2 checkpoint support to further improve availability in a SAS Grid environment.

INTRODUCTION

Before delving into the details of configuring a highly available environment for SAS servers, let's clarify some terminology of what components are required. The terms cluster, grid and high availability often mean different things to different people and require further discussion to clarify. Wikipedia defines cluster as "a small group or bunch of something". More specifically, with respect to computing, Wikipedia defines cluster as "a group of loosely coupled computers that work together closely". But even this definition is not specific enough; we must define the purpose of the computers working together closely. The two purposes that are relevant to this paper are to provide computing resources and to provide high availability.

A compute cluster or grid is a collection of compute, storage, network and data resources that can be shared by multiple users to run multiple applications. One of the promises of a grid architecture is increased availability of the underlying compute resources due to inherent system redundancy provided by multiple nodes in the grid. Also, because the applications running on the grid are not bound to specific components in the grid, hardware can be removed or introduced to the grid without disruption to the business users running the applications. SAS Grid Manager integrates SAS technology with components from Platform Computing to provide load balancing, policy enforcement, efficient resource allocation and prioritization for SAS products and solutions running in a shared grid environment. In addition, it de-couples the SAS applications and the infrastructure used to execute the applications, which allow hardware resources to transparently grow or contract as needed and provides tolerance of hardware failures within the grid infrastructure. SAS Grid Manager includes the Platform Suite for SAS, which allows you to create a

managed, flexible and shared environment to efficiently process multiple users, parallel workloads and enterprise scheduling.

But what happens if a machine in the grid goes down and that machine is running a SAS server or process that is critical to the continued operation of the overall environment? High availability ensures that all of the SAS applications and services are available nearly all of the time. A typical SAS deployment on a grid will include components such as the SAS Metadata Server, object spawner, SAS Workspace Server and SAS Stored Process Server. Failure of any one of these components can have a cascading affect on the clients and other applications running in the environment. The SAS Metadata Server is a central component to any SAS deployment. It contains information that is used by all of the other SAS products and components running in the environment. For example, it contains user information that is used to provide authentication and authorization as well as definitions of all of the SAS servers that have been configured for use. Because the SAS Metadata Server is such a critical component of the SAS environment, the examples in this paper will focus on providing high availability for this server. However, the concepts and processes can be applied to all components in your environment that require maximum availability.

The following diagram illustrates the multiple tiers in a SAS Business Analytics grid deployment.

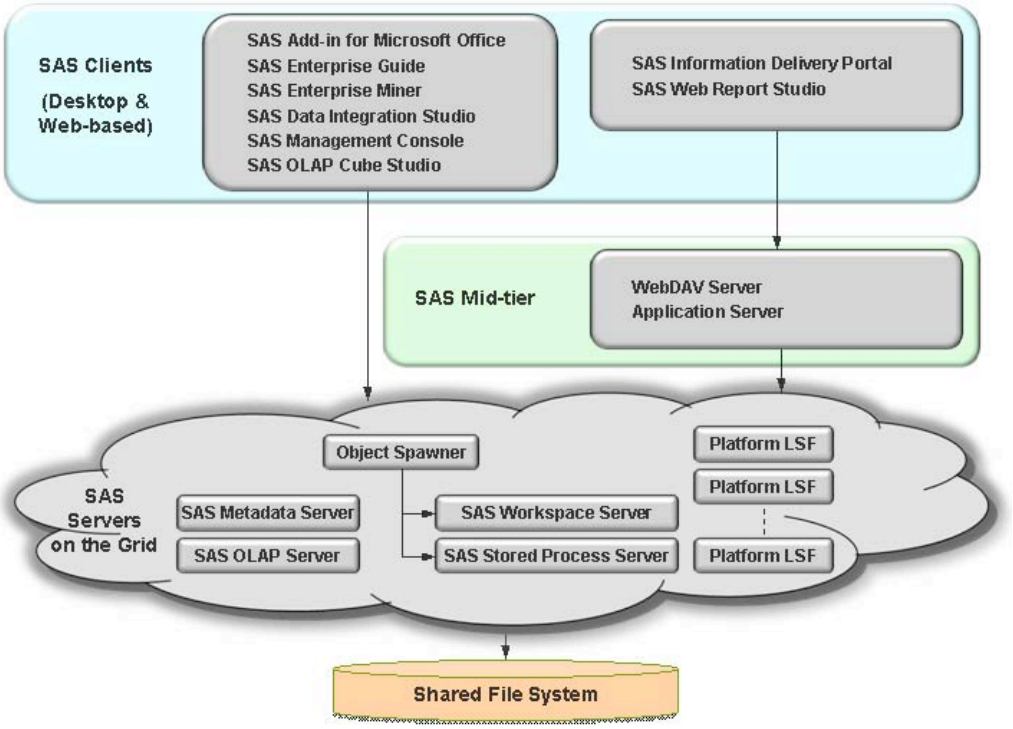


Figure 1. Sample SAS Business Analytics Architecture

WHAT IS HIGH AVAILABILITY?

High availability is a system design protocol and associated implementation that ensures a certain absolute degree of operational continuity during a given measurement period. Availability refers to the ability of the user community to access the system, whether to submit new work, update or alter existing work, or collect the results of previous work. If a user cannot access the system, it is said to be unavailable[1].

Common examples of unplanned downtime as a result of hardware or software failures include the following:

- Power outages
- Failed CPU, RAM, Disks NICs, …
- Over-temperature related shutdown
- Security breaches

[1] Wikipedia – High Availability

- Application, middleware, and operating system failures

Many solutions have been implemented to deal with specific types of failure of specific components. These solutions typically provide high availability through the following processes:

- Monitor – discover service (hardware or software) failures
- Recover – restart services when failures are detected
- Synchronize – ensure components in the environment are aware of the new configuration
- Reconnect – enable clients and applications to access the service again

Our discuss will follow a similar process for implementing high availability for critical system services in a SAS Grid environment.

SAS GRID ENVIRONMENT

Grid computing is, simply, the linking together of multiple computing resources – including processors and storage devices – to provide a much larger, more powerful single virtual computer. Grid computing offers a variety of benefits, including:

- Making individual applications run much faster.
- Allowing multiple applications to share computing resources.
- Increasing the use of all available computing resources.
- Removing knowledge of IT infrastructure from applications.
- Providing flexibility to add more resources to the grid as needed.

Because SAS is so analytically powerful, many SAS applications tend to be very data intensive, compute intensive or both. The performance of these SAS applications often can be improved dramatically by running in a grid environment. In addition, no business runs on a single application. A SAS grid environment provides a way to efficiently manage multiple SAS applications being run by multiple users and possibly many business units within an organization.

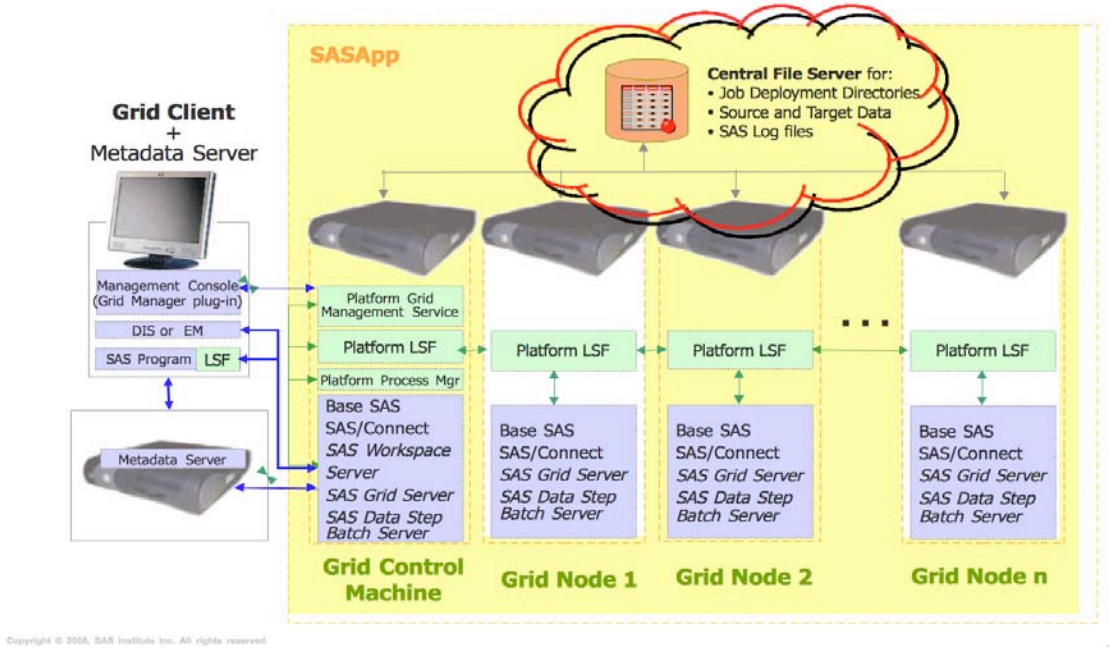


Figure 2. SAS Grid Topology

With the above topology, we highlight the key availability challenges as follows:

Components	Key Functions	High Availability Challenges
Metadata Server	SAS Metadata Server • Provides access to metadata used by all SAS products and solutions	• Must appear to exist at a constant host name which can be resolved to the IP address of the new failover host
Grid Control Server	SAS Workspace Server • Executes SAS programs submitted from clients such as SAS Enterprise Guide and SAS Data Integration Studio Platform LSF • Receive job requests from Grid clients and dispatches them to best available grid node Platform Process Manager • Schedule job flows with LSF Platform Grid Management Service • Provide status info	• Must recover from previous state after failover • Must restart SAS workspace server • Must make Grid Nodes aware of new configuration after failover • Only one Grid Control Server exists per Grid
Grid Nodes	• Receive requests from Grid Control Server and execute SAS workload	• Re-run job on a different Grid Node • Re-run job from previous checkpoint on the same or a different Grid Node
Grid Clients	• Client applications / GUI frontend for accessing Grid services	• Knowing new service arrangement after failover • Re-establish connection with servers
Central File Server	• A shared file repository across all Grid machines	• Critical dependency, Grid will not function without a central file server • Numerous solutions exist for implementing HA for a file server (this topic is beyond the scope of this paper)

In addition, monitoring services must be in place to automatically discover failure of critical services and trigger the corresponding recovery process.

The rest of our discussion will address the above challenges through the following sections:

- High availability for the SAS Metadata Server
- Eliminating the need for a hot stand-by during failover
- Failover of Platform Suite for SAS
- High availability for running SAS jobs with SAS checkpoint integration

HIGH AVAILABILITY FOR THE SAS METADATA SERVER

This section discusses how to configure the SAS Metadata Server to achieve high availability such that it can be monitored and recovered automatically. The same approach can be applied to other critical SAS services such as the SAS Object Spawner for example.

This section will outline the desirable failover behavior for the SAS Metadata Server, and provide step-by-step configuration instructions for setting up high availability for the SAS Metadata Server. These steps apply to all supported Windows, Unix and Linux platforms.

A DESIRABLE FAILOVER BEHAVIOR FOR SAS METADATA SERVER

1. The SAS Metadata Server is configured to be started by EGO on one of the hosts within the grid
2. SAS grid clients can lookup the host address where the SAS Metadata Server is. For example:

```
$nslookup
$server SASMeta.ego.sas.com
Server:         172.25.241.134     <-- corporate DNS server
Address:      172.25.241.134#53

Name:   SASMeta.ego.sas.com
Address: 172.25.235.50
```

   It is important that clients must be configured to use the hostname (SASMeta.ego.sas.com) instead of the IP address for accessing the server since the IP address will change during failover.

3. SAS Metadata Server goes down unexpectedly as the host crashes.
4. EGO detects that the SAS Metadata Server is gone, and tries to restart it on the next available host based on the control policy and resource requirements.
5. SAS Metadata Server is successfully restarted:

```
$nslookup
$server SASMeta.ego.sas.com
Server:         172.25.241.134
Address:      172.25.241.134#53

Name:   SASMeta.ego.sas.com
Address: 172.25.235.55
```

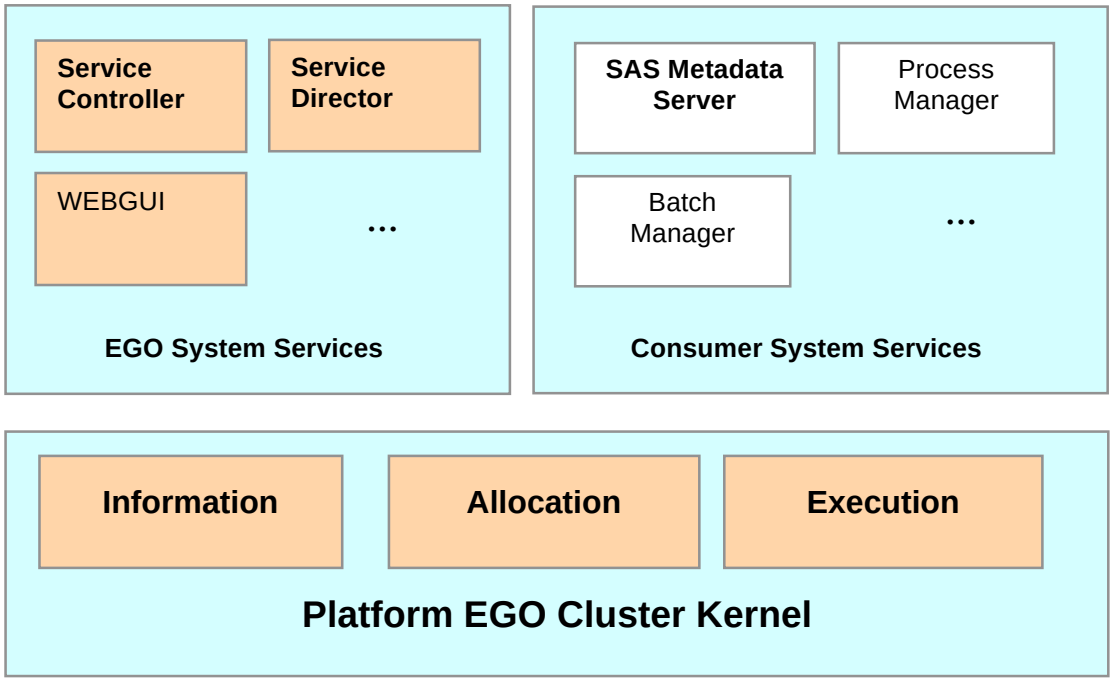
   Notice the hostname resolves into a new IP address.

6. The recovery is completed. Clients can still access the SAS service by connecting to `SASMeta.ego.sas.com`[2], even though the SAS service is restarted on a different host.

[2] Client connection state information may not be preserved after failover, client connections will eventually timeout and re-establish automatically with server again

WHAT IS PLATFORM EGO?

Platform Enterprise Grid Orchestrator (EGO) is part of the Platform Suite for SAS Version 4.1 and is provided as part of SAS Grid Manager for 9.2. It can also be obtained through SAS Technical Support for use with SAS 9.1.3. EGO is a collection of cluster orchestration software components that, among other things, provide high availability to critical services. The following figure shows the overall architecture and how these components fit within the SAS grid failover context:



Services	Role in a HA arrangement
EGO cluster kernel	EGO cluster kernel process starts automatically on grid control server. It is responsible for launching EGO service controller
EGO system services	EGO system services include EGO service controller, EGO service director, WEBGUI services, and etc.
EGO service controller	EGO service controller always starts on the grid control node. It starts EGO and consumer system services on remote grid nodes. It ensures that services are running by detecting failures and restarting service instances based on availability plans
EGO service director	EGO service director redirects client requests to the new service instance running on the failover node
Consumer system services	Consumer system services are third-party services that are registered to and controlled by EGO

Figure 3. How Different Services Fit into the EGO Architecture

HOW TO CONFIGURE HIGH AVAILABILITY FOR THE SAS METADATA SERVER

This section presents the configuration instructions to setup failover for the SAS Metadata Server with techniques discussed from the previous sections. Failover configuration for the SAS Metadata Server requires the setup of two EGO components and modifications to the Corporate DNS:

1. Register the SAS Metadata Server to EGO
   - Make sure EGO is correctly installed and configured. The Platform Suite for SAS Version 4.1 includes LSF7 and installs and enables EGO by default.
   - Write a service definition file for the SAS application, and place it under

     `%LSF_ENVDIR%\ego\<clustername>\eservice\esc\conf\services (Windows)`
     `$LSF_ENVDIR/ego/<clustername>/eservice/esc/conf/services (Unix)`

     This service definition file will be loaded by the EGO service controller when it starts. An example is provided in Figure 5.
   - Restart EGO to register and start the SAS Metadata Server.

2. Configure EGO service director
   - Make sure EGO ServiceDirector is running by running the "`egosh service list`" command (see Step 3 below)
   - Configure the default service director plug-in file under

     `%LSF_ENVDIR%\ego\<clustername>\eservice\esd\conf\esddefault.xml (Windows)`
     `$LSF_ENVDIR/ego/<clustername>/eservice/esd/conf/esddefault.xml (Unix)`

     to change the out-of-box service configuration to reflect your high availability needs. An example is provided in Figure 6.
3. Configure your Corporate DNS server by adding an NS record for the EGO service director

STEP 1: INSTALL LSF

1. Install the Platform Suite for SAS 4.1 provided with SAS Grid Manager 9.2. Make sure hostA and hostB are both server hosts. Make sure the shared configuration directory (e.g., \\hostF\LSFShare) is fully accessible by the account that runs LSF services.
2. Make sure hostA and hostB belong to the master list by adding hostA and hostB to `EGO_MASTER_LIST` in `%EGO_CONFDIR%\ego.conf` (`$EGO_CONFDIR/ego.conf` on Unix).

STEP 2: INSTALL THE SAS METADATA SERVER (NEED TO DECIDE SAS VERSION? 9.1 OR 9.2)

1. Before installing the SAS Metadata Server, proper domain IDs must have been created for administering and running the SAS Metadata Server.
2. The SAS Metadata repository and configuration information must be on a disk that is shared between fail-over cluster nodes. For example, the UNC path to the shared directory can be \\hostF\SAS. The user account under which the SAS Metadata Server runs should have full control over the shared directory.
3. Choose two hosts, for example hostA and hostB to be the primary host and failover host to run the SAS Metadata Server. The two hosts must belong to the LSF/EGO cluster that was created in Step 1: Install LSF.

4. Steps to install the SAS Metadata Server for 9.2:
    a. Start the SDW
    b. Select “Install SAS Software”; Press Next
    c. Select the order to install; Press Next
    d. Select “Perform a Deployment” and make sure the “Configure SAS Software” is checked. Also check “Install SAS Software” if SAS has not been installed. Press Next
    e. Choose a standard plan option and scroll down to the plan for “Metadata Server, one machine”. Press Next
    f. Choose ‘Custom’ install type. Press Next
    g. Set the configuration directory to the shared directory \\hostF\SAS. Make sure the configuration level is the same on both metadata server machines. Press Next
    h. Take the defaults for all configuration dialogs until you get to the Windows Options dialog.
    i. Make sure the “Run from Management Scripts” option is selected for Server Operation Type. Press Next
    j. Complete the configuration, taking the defaults where provided and filling in fields where needed.
    k. Repeat on hostB.

5. Steps to install the SAS Metadata Server for 9.1.3:
    a. Start SAS Software Navigator on hostA and select **Advanced** for deployment.
    b. Click **Select a standard plan** from the Select a deployment plan page and use the drop-down box to select **SAS Metadata Server**.
    c. At the end of installation, the SAS Configuration Wizard is launched. Specify the configuration directory location (For example: \\hostF\SAS).
    d. Specify **Run as scripts** on the SAS Server Configuration Options panel.
    e. On the Enter SAS Metadata Server Information page, enter the fully qualified physical node name of hostA, in the field SAS Metadata Server Host Name.
    f. Complete the Configuration Wizard and proceed with the instructions.html document produced by the SAS Software Navigator. Notice, the batch programs that come with the instructions will not work, as it does not recognize a UNC path. You must complete the configuration manually according to the instructions.
    g. Complete the installation and configuration of the SAS Metadata Server on hostB, using standard installation methodology. When prompted for configuration directory location, use the shared configuration directory (e.g. \\hostF\SAS). When prompted for the SAS Metadata Server Host Name, use the fully qualified physical node name of hostB.

STEP 3: CONFIGURE THE EGO SERVICES

1. Create the service definition file for the SAS Metadata Server and modify the service definition file for Service Director

    To create the service definition file for SAS Metadata Server, copy and rename a template from `%LSF_ENVDIR%\ego\<clustername>\eservice\esc\conf\services\service.xml.TMPL` to `%LSF_ENVDIR%\ego\<clustername>\eservice\esc\conf\services\sasmeta.xml` An example is provided in Figure 5.

    To modify the service definition file for Service Director, edit `%LSF_ENVDIR%\ego\<clustername>\eservice\esc\conf\services\named.xml` as shown in the example provided in Figure 6.

In order for EGO to monitor a service, it must be registered with EGO by defining the service definition file. EGO service controller then reads this file and request the EGO kernel to launch the service on the host(s) that has been defined. It further ensures that the service will continue to run by detecting failures and restarting service instances similar to the way that init on UNIX systems and Service Control Manager on Windows system work. The key parameters that require editing are listed in the following table.

Key parameters	Description	Example
Service Name	Defines the name of the service, this is a handle for EGO	SASMeta
Service Description	Provides a service description for the end user	SAS Metadata Server
StartType	Start service automatically	AUTOMATIC
HostFailoverInterval	Threshold for EGO to trigger failover	PT2M0S (two minutes)
EGOCommand	Command line to start the service	C:\Program Files\SAS\SAS 9.1\sas.exe" -config "\\HostF\SAS\MetadataServer\Lev 1\SASMain\MetadataServer\sasv9_ MetadataServer.cfg Note: Use the appropriate Unix command to start the SAS Metadata Server if you have a Unix environment. Note: For SAS 9.2 use the 9.2 command to start the SAS Metadata Server. For example: C:\Program Files\SAS\SASFoundation\ 9.2\sas.exe" -config "\\HostF\SAS\MetadataServer\Lev 1\SASMain\MetadataServer\sasv9_ MetadataServer.cfg
ResourceRequirement	Host candidates for service	select('hostA' || 'hostB')
ExecutionUser	User ID that runs the service	LSF\sas Note: Specify the appropriate Unix account in a Unix environment.

Figure 4. Key Parameters for the EGO Service Definition File

```
<?xml version="1.0" encoding="UTF-8"?>
<sc:ServiceDefinition xmlns:sc="http://www.platform.com/ego/2005/05/schema/sc"
xmlns:ego="http://www.platform.com/ego/2005/05/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.platform.com/ego/2005/05/schema/sc ../sc.xsd
http://www.platform.com/ego/2005/05/schema ../ego.xsd" ServiceName="SASMeta">
  <sc:Version>1.2</sc:Version>
  <sc:Description>SAS Metadata Server</sc:Description>
  <sc:MinInstances>1</sc:MinInstances>
  <sc:MaxInstances>1</sc:MaxInstances>
  <sc:Priority>1</sc:Priority>
  <sc:MaxInstancesPerSlot>1</sc:MaxInstancesPerSlot>
  <sc:MaxInstancesPerHost>1</sc:MaxInstancesPerHost>
  <sc:NeedCredential>FALSE</sc:NeedCredential>
  <sc:ControlPolicy>
    <sc:StartType>AUTOMATIC</sc:StartType>
    <sc:MaxRestarts>10</sc:MaxRestarts>
    <sc:HostFailoverInterval>PT2M0S</sc:HostFailoverInterval>
  </sc:ControlPolicy>
  <sc:AllocationSpecification>
    <ego:ConsumerID>/ManagementServices/EGOManagementServices</ego:ConsumerID>
<!--The ResourceType specifies a "compute element" identified by the URI used below-->
    <sc:ResourceSpecification ResourceType="http://www.platform.com/ego/2005/05/schema/ce">
    <ego:ResourceGroupName>ManagementHosts</ego:ResourceGroupName>
    <ego:ResourceRequirement>select('hostA' || 'hostB')</ego:ResourceRequirement>
</sc:ResourceSpecification>
  </sc:AllocationSpecification>
  <sc:ActivityDescription>
    <ego:Attribute name="hostType" type="xsd:string">all</ego:Attribute>
    <ego:ActivitySpecification>
    <ego:Command>
"C:\Program Files\SAS\SAS 9.1\sas.exe" -config
"\\HostF\SAS\MetadataServer\Lev1\SASMain\MetadataServer\sasv9_MetadataServer.cfg"
      </ego:Command>
     <ego:ExecutionUser>LSF\sas</ego:ExecutionUser>
     <ego:Umask>0022</ego:Umask>
    </ego:ActivitySpecification>
  </sc:ActivityDescription>
</sc:ServiceDefinition>
```

Figure 5. Sample Service Definition File for the SAS Metadata Server
`%LSF_ENVDIR%\ego\<clustername>\eservice\esc\conf\services\sasmeta.xml`

Note: On Unix, the location for `sasmeta.xml` is
`$LSF_ENVDIR/ego/<clustername>/eservice/esc/conf/services/sasmeta.xml`

- The `Command` to launch the SAS Metadata Server should be changed to the appropriate command for your corresponding platforms
- The `ExecutionUser` should have sufficient privilege to run the service (for example `root`)

```
<?xml version="1.0" encoding="UTF-8"?>
<sc:ServiceDefinition xmlns:sc="http://www.platform.com/ego/2005/05/schema/sc"
xmlns:ego="http://www.platform.com/ego/2005/05/schema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xsi:schemaLocation="http://www.platform.com/ego/2005/05/schema/sc ../sc.xsd
http://www.platform.com/ego/2005/05/schema ../ego.xsd" ServiceName="ServiceDirector">
  <sc:Version>1.2</sc:Version>
  <sc:Description>EGO: Service Director</sc:Description>
  <sc:MinInstances>1</sc:MinInstances>
  <sc:MaxInstances>1</sc:MaxInstances>
  <sc:Priority>10</sc:Priority>
  <sc:MaxInstancesPerSlot>1</sc:MaxInstancesPerSlot>
  <sc:MaxInstancesPerHost>1</sc:MaxInstancesPerHost>
  <sc:NeedCredential>FALSE</sc:NeedCredential>
  <sc:ControlPolicy>
    <sc:StartType>AUTOMATIC</sc:StartType>
    <sc:MaxRestarts>1</sc:MaxRestarts>
    <sc:HostFailoverInterval>PT1M0S</sc:HostFailoverInterval>
  </sc:ControlPolicy>
  <sc:AllocationSpecification>
    <ego:ConsumerID>/ManagementServices/EGOManagementServices</ego:ConsumerID>
<!--The ResourceType specifies a "compute element" identified by the URI used below--
>
    <sc:ResourceSpecification
ResourceType="http://www.platform.com/ego/2005/05/schema/ce">
      <ego:ResourceGroupName>ManagementHosts</ego:ResourceGroupName>
      <ego:ResourceRequirement>select('hostA' || 'hostB')</ego:ResourceRequirement>
    </sc:ResourceSpecification>
  </sc:AllocationSpecification>
  <sc:ActivityDescription>
    <ego:Attribute name="hostType" type="xsd:string">all</ego:Attribute>
    <ego:ActivitySpecification>
      <ego:Command>
        ${EGO_TOP}\7.0\scripts\egosrvloader.bat ${EGO_TOP}\7.0\etc\named –f
      </ego:Command>
      <ego:ExecutionUser>LSF\sas</ego:ExecutionUser>
      <ego:EnvironmentVariable name="PATH">
         ${EGO_TOP}\7.0\bin;${EGO_TOP}\7.0\lib
      </ego:EnvironmentVariable>
      <ego:EnvironmentVariable name="ESD_CONF">
         ${EGO_TOP}\conf\ego\cluster1\eservice\esd\conf
      </ego:EnvironmentVariable>
      <ego:Umask>0022</ego:Umask>
    </ego:ActivitySpecification>
  </sc:ActivityDescription>
</sc:ServiceDefinition>
```

Figure 6. Sample Service Definition File for the Service Director

`%LSF_ENVDIR%\ego\<clustername>\eservice\esc\conf\services\named.xml`

Note: On Unix, the location for `named.xml` is
`$LSF_ENVDIR/ego/<clustername>/eservice/esc/conf/services/named.xml`

The Service Director service definition file is generated automatically by EGO. You only need to modify the following two places (apply to both Windows and Unix):

- Change the `StartType` from `MANUAL` to `AUTOMATIC`
- Modify the `ResourceRequirement` to select the appropriate primary and failover host (e.g., `hostA` and `hostB`)

2. Configuring EGO and Corporate name services

Parameters	Description
ESD_EGO_NAMESERVER	The service director DNS server name. The cluster administrator must add an NS resource record into CORP DNS database, indicating that this server is used as the service director DNS server
ESD_EGO_DOMAIN	EGO sub-domain name
ESD_CORP_DOMAIN	Corporation domain name
ESD_EGO_KEY	TSIG KEY used to update the service director DNS server (optional)
ESD_CORP_KEY	TSIG KEY used to update Corporation DNS server (optional)

Figure 7. Key Parameters for EGO Service Director

Edit
%LSF_ENVDIR%\ego\<clustername>\eservice\esd\conf\esddefault.xml
($LSF_ENVDIR/ego/<clustername>/eservice/esd/conf/esddefault.xml on Unix):

```
<?xml version="1.0" encoding="UTF-8"?>
<ESDDefaultPluginConfiguration>
  <!-- EGO DNS server name -->
  <ESD_EGO_NAMESERVER>egonameserver</ESD_EGO_NAMESERVER>
  <!-- EGO DNS domain name -->
  <ESD_EGO_DOMAIN> ego.sas.com</ESD_EGO_DOMAIN>
  <!-- Corporation DNS domain name -->
  <ESD_CORP_DOMAIN>sas.com.</ESD_CORP_DOMAIN>
  <!-- EGO DNS sub-domain TSIG key created by dnssec-keygen -->
  <ESD_EGO_KEY name="ego.platform.com.">EGOSD_DNS_TSIG_KEY</ESD_EGO_KEY>
  <!-- CORP DNS domain TSIG key created by dnssec-keygen -->
  <ESD_CORP_KEY name="platform.com.">CORP_DNS_TSIG_KEY</ESD_CORP_KEY>
</ESDDefaultPluginConfiguration>
```

Figure 8. Sample EGO Service Director Configuration File

Edit
%LSF_ENVDIR%\ego\<clustername>\eservice\esd\conf\named\conf\named.conf
($LSF_ENVDIR/ego/<clustername>/eservice/esd/conf/named/conf/named.conf on Unix):

```
…   key ego.sas.com {
                algorithm HMAC-MD5.SIG-ALG.REG.INT;
                secret "EGOSD_DNS_TSIG_KEY";
    };

…
    zone " ego.sas.com." IN {
                 type master;
                 file "db.ego.sas.com";
                 allow-update { key ego.sas.com; };
    };
…
```

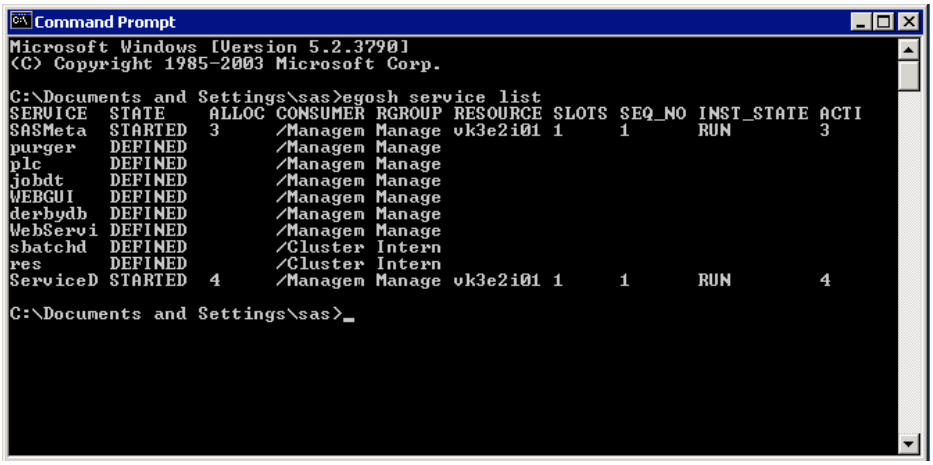
Figure 9. Sample EGO Name Service Configuration File

Rename the file `TMPL.db.EGODOMAIN.CORPDOMAIN` under `%LSF_ENVDIR%\ego\<clustername>\eservice\esd\conf\named\namedb\` (`$LSF_ENVDIR/ego/<clustername>/eservice/esd/conf/named/namedb/` on Unix) to `db.ego.sas.com`, edit `db.ego.sas.com`:

```
$ORIGIN .
$TTL 0 ; 0 seconds
ego.sas.com IN SOA egonameserver.sas.com. root.ego.sas.com. (
                        84           ; serial
                        10800     ; refresh (3 hours)
                        900         ; retry (15 minutes)
                        604800    ; expire (1week)
                        0             ; minimum (0 seconds)
                        )
               NS    egonameserver.sas.com.
               NS    egonameserver.ego.sas.com.
```

Figure 10. Sample Corporate DNS Configuration

3. Restart the cluster by running: `egosh ego restart all`
4. Check EGO service status by running: `egosh service list`



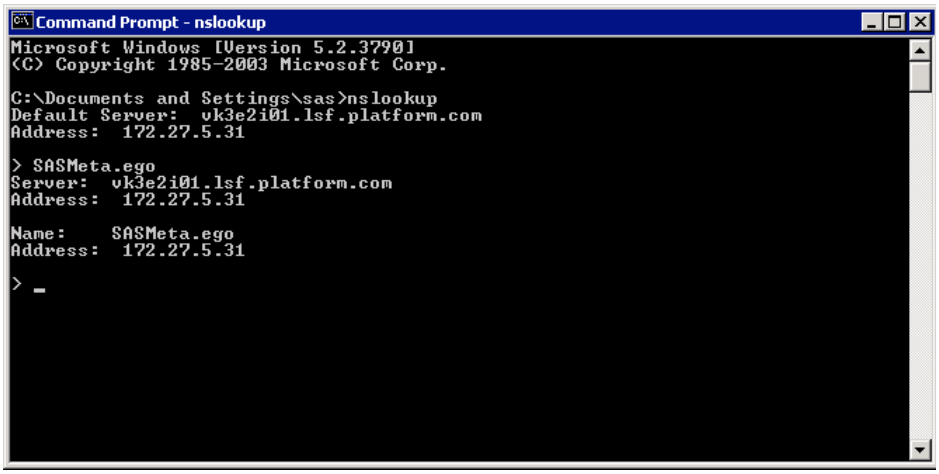
5. Once the SAS Metadata Server and EGO service director are running, you need to configure your DNS server to point to the EGO service director name server so that the names of the active EGO service instances can be resolved. Alternatively, you can configure your Corporate DNS by adding an NS record such that all ego sub-domain DNS lookup will be forwarded to the EGO service director. For example: `ego NS egonameserver`

   Refer to Figure 7 – Figure 10 for a sample configuration of EGO service director.

   Note: Refer to **Appendix – EGO Service Director** for more details on how EGO service director interacts with your Corporate DNS.

6. It is important to disable the DNS cache on the client machine, so that when failover of the EGO name server occurs, the client will query the new name server instead of using the stale copy of name entries in the DNS cache. To disable DNS cache, go to the windows Services panel, stop DNS Client service, and change the startup Type to Manual.

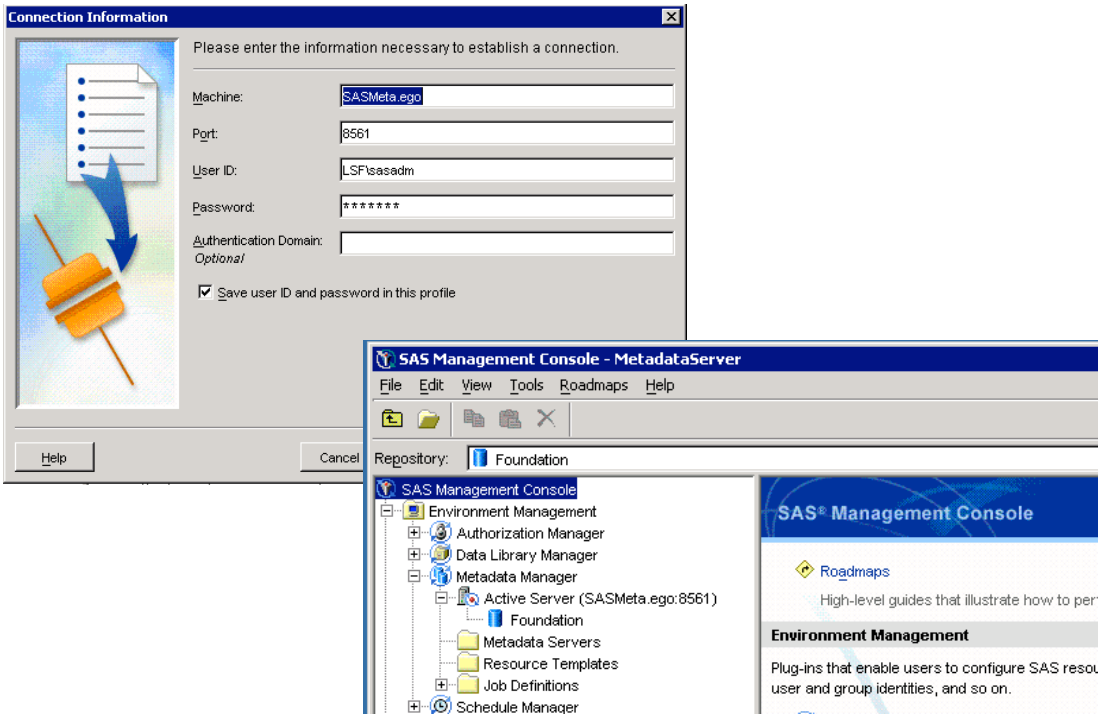
7. Test to make sure that your hostname can be resolved:



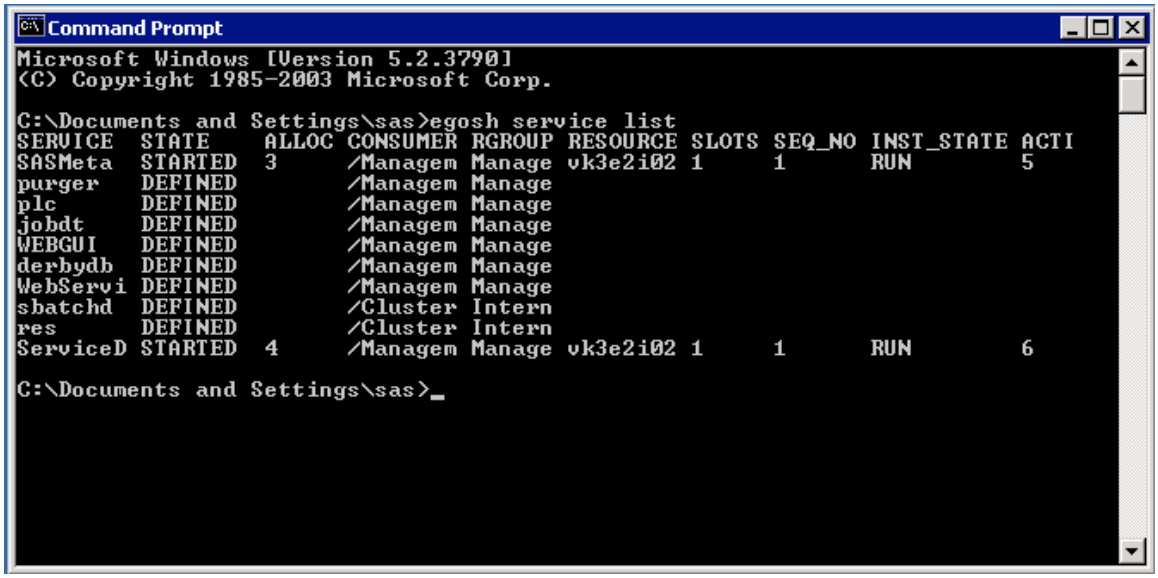
8. When configuring other SAS software, make sure to always use SAS Metadata Server service name (e.g. `SASMeta.ego`) rather than the physical host name when specifying the location of the SAS Metadata Server.

STEP 4: FAILOVER TESTS

1. At this point, you have a functioning SAS Metadata Server that leverages EGO to provide failover. In order to validate this, install SAS Management Console outside of the LSF/EGO cluster. Configure the metadata profile to connect to the Metadata Server using SASMeta.ego, as shown in the following diagrams:



2. To validate failover, shut down hostA. After a while, EGO master will failover to hostB, and the EGO service director and SAS Metadata Server will then be started on hostB:



3. Restart SAS Management Console, and it should connect to the new SAS Metadata Server correctly.

ELIMINATING THE NEED FOR A HOT-STANDBY DURING FAILOVER

One advantage of EGO is that it does not require a hot-standby during failover of EGO services. This is achieved by configuring one of the grid nodes as the failover host. When an EGO service such as the SAS Metadata Server experiences failover to a new host, we can configure EGO to automatically reconfigure LSF such that no jobs will be dispatched to this host from this point forward. Therefore, the failover host will be used exclusively for running EGO services.

To implement this, we have to replace the EGO command (refer to figure 4 & 5) with a wrapper script like the one below:

```
@ ECHO OFF

REM Close the local host
badmin hclose

REM Add the path to the EGO service command here
C:\Program Files\SAS\SAS 9.1\sas.exe" -config
"\\HostF\SAS\MetadataServer\Lev1\SASMain\MetadataServer\sasv9_MetadataServer.cfg"
```

Figure 11. Sample Wrapper Script for Stopping a Failover Host from Accepting Jobs

Before running the normal EGO service, the above wrapper script will stop LSF from dispatching any more jobs to the failover host by closing the host with the "`badmin hclose`" command.

It should be noted that jobs that had started before the failover will continue to run until they complete.

FAILOVER OF PLATFORM SUITE FOR SAS

The Platform Suite for SAS also has inherent failover capabilities built-in to many of its components specifically LSF and EGO.

FAULT-TOLERANT DESIGN IN LSF

SAS Grid topology requires a single control node to be present for orchestrating activities among the rest of the grid nodes. Because of this important function, ability to recover from a control node failure is of critical importance for achieving high availability.

One of the critical services running in the control node is the LSF Load Index Manager (LIM). LIM runs on every host with one host running in the master mode, and the rest of the hosts running in slave mode. We will use this as an example to illustrate how failure is detected and recovered.

FAILURE DETECTION AND MASTER 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.

Figures 12 and 13 illustrate the interaction between the master and slaves, and the transition diagram of a slave self-promoting to master.

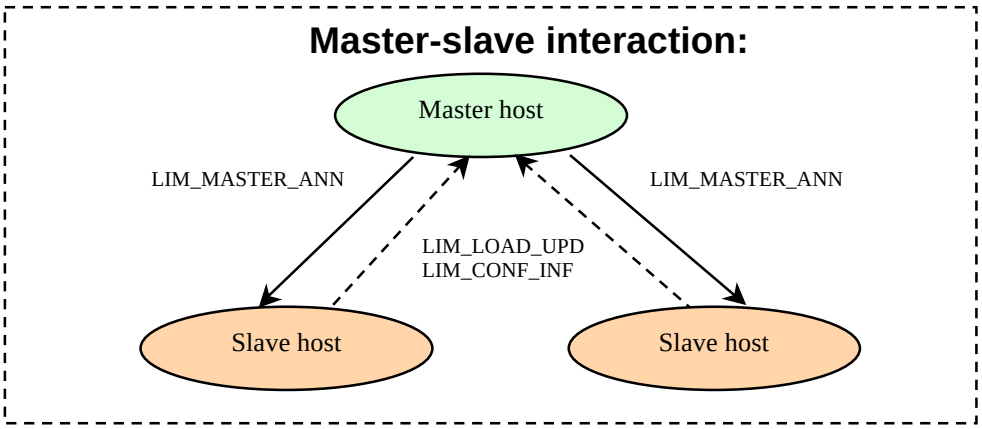


Figure 12. The Interaction Between Master and Slave Daemons

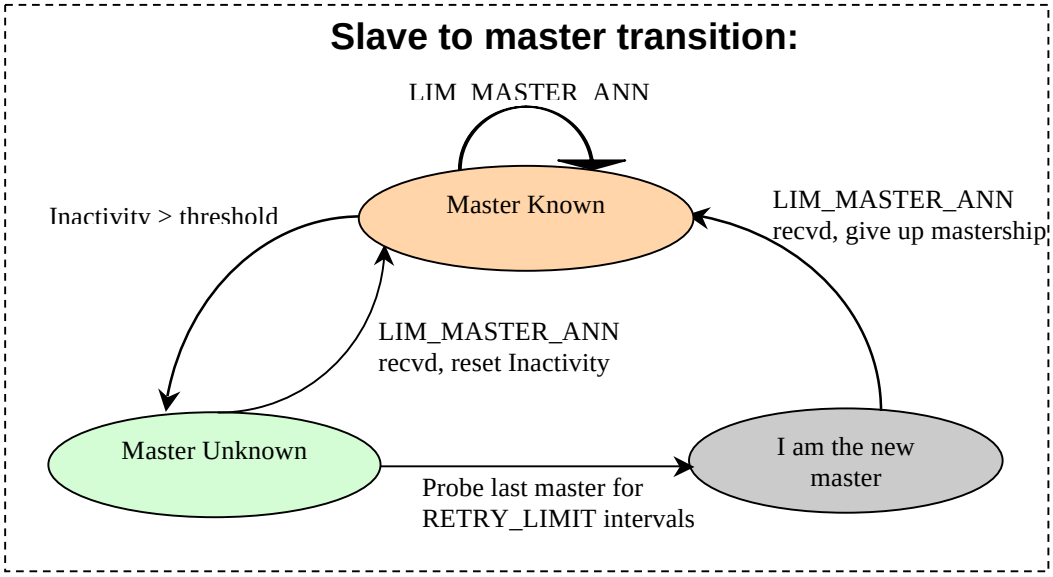


Figure 13. The Slave to Master Transition

FAILOVER OF EGO

A frequently asked question is what will happen when EGO itself fails or the machine on which it is running fails?

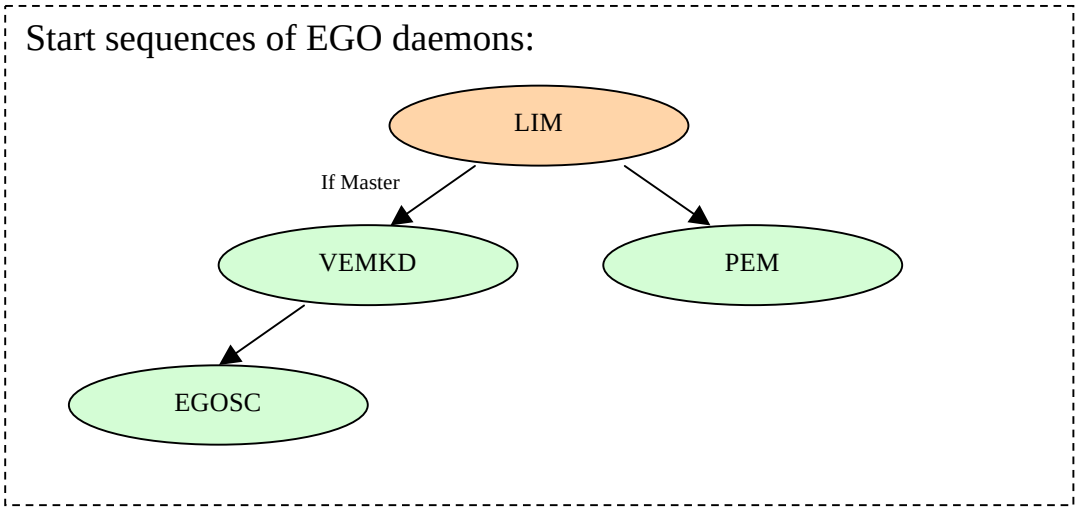


Figure 14. Start Sequence of EGO Daemons

LIM is always the first daemon to start on a host. If it is a master LIM, it will start up EGO kernel (VEMKD), which in turn starts up the EGO Service Controller (EGOSC).

During its running, master LIM keeps monitoring VEMKD. If VEMKD exits, master LIM will restart it to support host-level fail over. In case VEMKD core dumps or exits abnormally, master LIM will gradually extend VEMKD restart interval.

During its running, VEMKD keeps monitoring EGOSC. If EGOSC exits, VEMKD will automatically restart EGOSC.

If master host goes down, one of slave LIMs on master candidate list will take over mastership. Once this LIM becomes master, it will start up VEMKD and then EGOSC on its host.

All EGO service instances information (e.g., instance location, instance status, and etc.) is persisted for recovery purposes. All service instances running on hosts that are not reachable will be restarted during failover.

HA FOR RUNNING SAS JOBS WITH SAS CHECKPOINTING INTEGRATION

If a grid node crashes while running a SAS job, LSF has the built-in capability to detect and restart all jobs[3] that were running on the crashed node on other suitable nodes. One drawback for this arrangement is that a job must be rerun again from the beginning. For long running jobs, this may imply a significant lost of productivity.

To address this problem, we are currently testing functionality to integrate the SAS checkpoint/restart feature with LSF such that jobs can be automatically restarted from the last checkpoint image rather than starting from the very beginning. We expect to release this functionality in an early 9.2 maintenance release.

SAS 9.2 CHECKPOINT AND RESTART MODE

Checkpointing consists of storing a snapshot of the current application state, and use it for restarting in case of failure. When checkpoint mode is enabled, SAS records information about DATA and PROC steps in a checkpoint library. When a batch program terminates prematurely, we can resubmit the program in restart mode to complete execution.

In restart mode, global statements and macros are re-executed and SAS reads the data in the checkpoint library to determine which steps completed. Program execution resumes with the step that was executing when the failure occurred.

JOB RERUN AND REQUEUE IN LSF

Automatic job rerun occurs when the execution host becomes unavailable while a job is running. When a job is rerun or restarted, it is returned to the queue from which it was dispatched with the same options as the original job. The priority of the job is set sufficiently high to ensure that the job gets dispatched before other jobs in the queue. The job uses the same job ID number. It is executed when a suitable host is available.

A similar feature to job rerun is job requeue. Automatic job rerun occurs when a job finishes and has a specific exit code. This exit code can be used by a SAS program to indicate whether it should be rerun from scratch or from the last checkpoint.

By combining job rerun and requeue, we can automatically restart SAS jobs from last checkpoint on another suitable host when the original execution host failed. We can also automatically restart a failed job from the last checkpoint automatically on a suitable host should the exit code indicates that the job should be rerun.

CONCLUSION

We discussed challenges in providing high availability in a SAS Grid environment, and described in details some of the built-in failover mechanism to overcome these challenges. In addition, we introduced the EGO technology and provided step-by-step configuration examples to enable failover support for SAS Metadata Server. To recap, this approach has the following unique advantages:

- Works under all SAS supported Windows, Unix and Linux platforms
- No hot-standby required – can leverage one of the grid node for failover, keeping all hardware fully utilized at all time
- Included in SAS 9.2 version of Platform Suite for SAS

Finally, we discussed the possibility of integrating SAS checkpoint support with LSF as our future research focus.

[3] Jobs must be set as rerunnable jobs, otherwise LSF will report a job failed status. Currently the only way to set a SAS job as rerunnable is using the Advanced Properties tab in the Scheduling Manager plug-in.

APPENDIX - EGO SERVICE DIRECTOR

The EGO service director is an EGO system service that functions as a locating mechanism for other system services. The service director contains a stand-alone Domain Name Server (DNS), which is the authoritative name server for the EGO DNS sub-domain and responds to DNS queries for system services.

The service director runs on the EGO master host and relies on the service controller to provide location information and state change notifications of service instances. When a service instance enters the RUN state, the service director adds its location information into the service director DNS server. When a service instance transfers from the RUN state into other states, the service director deletes the location information from its DNS server.

A client who is looking to find a host with an EGO service started on it must directly query/point to the service director DNS server as nameserver. The nameserver entries (e.g. /etc/resolv.conf in linux) gets updated each time to point to the service director DNS server and to resolve the name and IP address. To avoid making operating system configuration changes each time there is a query, one can configure service director to provide name-to-address mapping to the corporation DNS server (corpdns). Doing this provides location independence for the hosts such that a service can start on anywhere.

When the location of the service director DNS server is changed, the service director updates the corresponding resource record in the DNS database of the corporation DNS server. When the locations of EGO system service instances are changed, the service director updates the corresponding resource records in the service director DNS server.

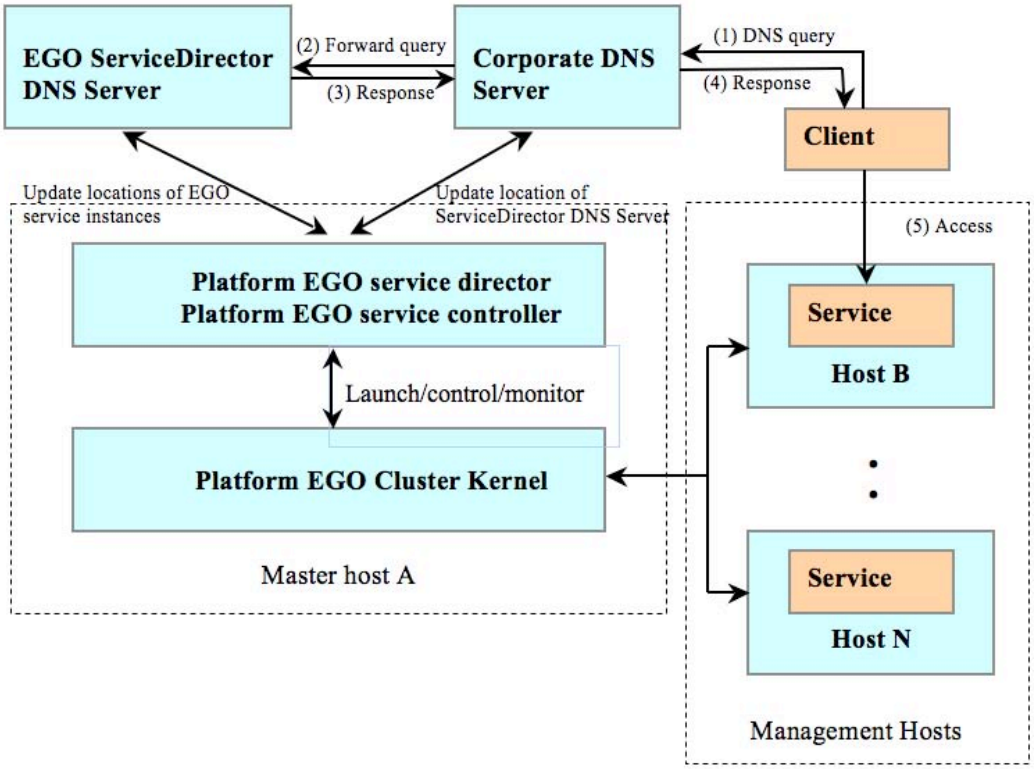


Figure 15. The Interaction Between Client Application, Corporate and EGO DNS Servers

ACKNOWLEDGMENTS

The author of this paper would like to acknowledge the contributions and help he has received from the following people:

Cheryl Doninger of SAS Institute Inc.

Gary Ciampa of SAS Institute Inc.

Qingda Wang of Platform Computing Inc.

Meng Ding of Platform Computing Inc.

CONTACT INFORMATION

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

Name:	Daniel Wong
Enterprise:	Platform Computing Inc.
Address:	3760 14th Avenue, Suite 600
City, State ZIP:	Markham, ON L3R 3T7
Work Phone:	(905) 948-4531
Fax:	(905) 948-9975
E-mail:	dwong@platform.com
Web:	www.platform.com