# Revision history:

<table>
<thead>
<tr>
<th>RELEASE/REVISION:</th>
<th>RELEASE/REVISION DATE:</th>
<th>SUMMARY OF CHANGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
<td>May 29, 2013</td>
<td>First Draft of architectural response to canonical use case one.</td>
</tr>
<tr>
<td>0.91</td>
<td>July 21, 2013</td>
<td>This version has significant updates and follows the approval of the approach to developing the use case responses (last week of June, 2013).</td>
</tr>
<tr>
<td>.92</td>
<td>September 9, 2013</td>
<td>A series of updates to respond to questions and issues brought up on the discussion threads.</td>
</tr>
<tr>
<td>.93</td>
<td>September 18, 2013</td>
<td>Responses to Derek Simmel comments</td>
</tr>
<tr>
<td>.94</td>
<td>November, 3, 2013</td>
<td>Completed the rest of the quality attributes. Added new text in introduction to section 3.3, created text for 3.3.3-3.3.5, and 3.3.7-3.3.9.</td>
</tr>
</tbody>
</table>
Table of Contents

Contents

1 Introduction .......................................................................................................................... 3
  1.1 Structure of this Document ............................................................................................ 3
  1.2 Document Management and Configuration Control ....................................................... 3

2 Canonical Use Case 1 .......................................................................................................... 4

3 Run a Remote Job – Reviewed 2013-11-03 ........................................................................ 6
  3.1 Definitions ....................................................................................................................... 6
  3.2 Variants .......................................................................................................................... 8
    3.2.1 Variant UCCAN 1.a - Data Staging ....................................................................... 8
    3.2.2 Variant UCCAN 1.b - Access to Remote Data During Execution ......................... 8
    3.2.3 Variant UCCAN 1.c - compute resource is not known a priori ............................. 9
    3.2.4 Variant UCCAN 1.d - Use Group Credentials ...................................................... 10
    3.2.5 Variant UCCAN 1.e - Registration Notification .................................................... 11
    3.2.6 Variant UCCAN 1.f - Interact with Job Session Directory .................................... 11
    3.2.7 Variant UCCAN 1.g - At Most Once Semantics ..................................................... 11
    3.2.8 Variant UCCAN 1.h - Operations on sets of jobs .................................................. 11
    3.2.9 Variant UCCAN 1.i - Parameter Sweep .................................................................. 11

3.3 Quality of Service Attributes ......................................................................................... 12
  3.3.1 QAS-CAN1.a Any request to the execution service is acknowledged within one second. 12
  3.3.2 QAS-CAN1.b High-throughput: 1) rate jobs can be submitted, 2) the number of active jobs, 3) the total number of jobs the service can manage. ................................. 14
  3.3.3 QAS-CAN1.c The execution service can support, without error, as many queued and active jobs are permitted by its associated resource management system. .......... 15
  3.3.4 QAS-CAN1.d Once a job is complete its status can be checked for at least 24 hours. 15
3.3.5 QAS-CAN1.e Client request patterns that exceed the stated job submission rate, queued jobs, or active jobs, are handled gracefully. .......................................................... 16

3.3.6 QAS-CAN1.f The execution service can be restarted without loss of jobs at three Sigma. 16

3.3.7 QAS-CAN1.g Valid jobs complete successfully at two Sigma. ........................................... 16

3.3.8 QAS-CAN1.h Availability of the execution service is 1.8 Sigma. .......................... 17

3.3.9 QAS-CAN1.i If a compute resource associated with the execution service fails, then any job currently executing on that resource is reported as failed. ............................................. 18

3.3.10 QAS-CAN1.j The job states must be consistent and well defined across all resources. 18
1 Introduction

This document describes the realization of Canonical Use Case 1, "Run a remote job", using the XSEDE X-WAVE architectural components. See http://hdl.handle.net/2142/43877 for the use cases.

It is assumed that the reader has already read and is familiar with the XSEDE Architecture Level 3 Decomposition (L3D), in particular sections 3 (Access Layer), 4.1 (Open Standards-Based Web Services Architecture), 5(X-WAVE), and 8 (Deployment). Further, the Genesis II Omnibus Reference Manual (GORM) will be frequently referred to. The authors suggest that these two documents be open or on hand when reading this document.

1.1 Structure of this Document

This document is organized as follows. Section 2 briefly describes the remote execution use case. Section 3 describes how the X-WAVE components are used to implement the use case from section 2.

1.2 Document Management and Configuration Control

This Version 0.95 of the XSEDE X-WAVE Level 3 Decomposition was first released on May 22, 2013.
2 Canonical Use Case 1

Canonical use case 1 is "Run a remote job". The description is "A user executes and manages a job (sequential or parallel) on a remote compute resource."

The use case starts with a number of assumptions, specifically:

- The client is properly authenticated.
- The client has generated a job description in the appropriate format.
- The client already knows and has the address of the execution service that is to be used to run the job. In other words there is no resource discovery required.
- The compute resource is able to execute the client’s application, e.g., there is an appropriate binary, there is sufficient memory, etc.
- The compute resource does not fail during job execution.
- The underlying resource management system is at least as reliable as the requirements for the execution service and performs at least as well as the execution service requirements. In other words, to meet execution services quality attributes requires certain quality attributes from the resource management system and the compute resource.
- All data needed by the job are already on the compute resource file system, and all results are left on the compute resource file system.

There are then a number of variants and quality attributes for the use case. These are shown in table 1.
Table 1: Run a remote job, variants, and quality attributes.

<table>
<thead>
<tr>
<th>Variant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCCAN 1.a</td>
<td>Data needs to be copied in before execution and/or out after execution.</td>
</tr>
<tr>
<td>UCCAN 1.b</td>
<td>Remote data needs to be accessed during application execution.</td>
</tr>
<tr>
<td>UCCAN 1.c</td>
<td>The specific compute resource is not known <em>a priori</em>, and must be found be the client or by a third party.</td>
</tr>
<tr>
<td>UCCAN 1.d</td>
<td>The client is not authorized, using its own credentials alone, to use a resource, but is a member of a community or group that is authorized to use the compute resource.</td>
</tr>
<tr>
<td>UCCAN 1.e</td>
<td>The client can register with the execution service for notifications of job state change.</td>
</tr>
<tr>
<td>UCCAN 1.f</td>
<td>The client can interact with the session directory on the compute resource file system before, during, and after job execution by the compute resource.</td>
</tr>
<tr>
<td>UCCAN 1.g</td>
<td>At most once semantics.</td>
</tr>
<tr>
<td>UCCAN 1.h</td>
<td>Submission, status checking, and job control operations work on sets of jobs.</td>
</tr>
<tr>
<td>UCCAN 1.i</td>
<td>The client can submit a set of jobs (e.g., a parameter sweep) in a single request, and then monitor and control them as a single job.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QAS-CAN1.a</td>
<td>Any request to the execution service is acknowledged within one second.</td>
</tr>
<tr>
<td>QAS-CAN1.b</td>
<td>High-throughput: 1) rate jobs can be submitted, 2) the number of active jobs, 3) the total number of jobs the service can manage.</td>
</tr>
<tr>
<td>QAS-CAN1.c</td>
<td>The execution service can support, without error, as many queued and active jobs are permitted by its associated resource management system.</td>
</tr>
<tr>
<td>QAS-CAN1.d</td>
<td>Once a job is complete its status can be checked for at least 24 hours.</td>
</tr>
<tr>
<td>QAS-CAN1.e</td>
<td>Client request patterns that exceed the stated job submission rate, queued jobs, or active jobs, are handled gracefully.</td>
</tr>
<tr>
<td>QAS-CAN1.f</td>
<td>The execution service can be restarted without loss of jobs at three Sigma.</td>
</tr>
<tr>
<td>QAS-CAN1.g</td>
<td>Valid jobs complete successfully at two Sigma.</td>
</tr>
<tr>
<td>QAS-CAN1.h</td>
<td>Availability of the execution service is 1.8 Sigma.</td>
</tr>
<tr>
<td>QAS-CAN1.i</td>
<td>If a compute resource associated with the execution service fails, then any job currently executing on that resource is reported as failed.</td>
</tr>
<tr>
<td>QAS-CAN1.j</td>
<td>The job states must be consistent and well defined across all resources.</td>
</tr>
</tbody>
</table>
3 Run a Remote Job – Reviewed 2013-11-03

3.1 Definitions

Some terms from the use case document require clarification, others we need to use in this document.

Client file system. The client file system is the file system on the host where the client application is running. If the user is submitting a job while logged in at a center, the client file system would be the center file system visible from a login node. If the client is executing on a PC in their office then the file systems visible from the PC are the client file systems.

Client/user. The use case uses client and user interchangeably and to mean the human user and the software client working together with the client being the software that actually interacts with other agents. We continue this usage in this L3 response document.

Local resource manager. Local resource managers include PBS, SGE, SLURM, Condor, etc.

Session directory. Session directory is a common term of art in remote job management systems. It is often also called the job working directory. The session directory is the established by the execution management system, one per job. When the job is started its current working directory is the session directory. The lifetime of the session directory varies, but it should remain intact until it is purged. Often the session directory is created before the job is submitted to the local resource manager and clients may manually copy data into the session directory before the job is submitted to the local resource management system. The BES specification is intentionally quite on when the session directory is purged.
Assume that

1. The Genesis II Access Layer package is installed on the computer where the job is to be submitted and monitored. (L3D 5.4.4).

2. The Genesis II or UNICORE 6 container is correctly installed on any compute resource to be used and a BES (L3D 5.1.3) has been instantiated and properly configured (e.g., GORM F.5).

3. Users have necessary permissions.

4. The user is authenticated as described in L3D 5.3.2.2 (XSEDE Portal ID Case). This means that the client session has an XSEDE MyProxy session certificate as well as delegated SAML certificates from a KerbAuthNPortType.

Steps:

1. Create a JSDL (Job Submission Description Language, LD3 5.1.2.2) job description using a text editor or Genesis II access layer client GUI (LD3 3.3.2.5, GORM E.5.1).
2. Run the job as described in (L3D 5.3.9, 5.3.10, GORM E.5.3, E.5.7, E.5.8) directly on a BES (shown above as running on an NCSA BES).

3.2 Variants

3.2.1 Variant UCCAN 1.a - Data Staging

Data needs to be copied in before execution and/or out after execution.

The JSDL specification (Job Submission Description Language, LD3 5.1.2.2) provides for the inclusion of file staging operations. Files and directory trees can be staged in prior to execution; and files and directory trees can be staged out post execution. The ability to specify files to be staged in and out is supported by the Genesis II access layer client GUI JSDL tool (LD3 3.3.2.5, GORM E.5.1). Supported protocols by the UNICORE 6 and Genesis II BES (L3D 5.1.2.5) implementations include GFFS/rns, http, https, ftp, gridftp, and scp for staging files in and rns, ftp, scp, gridftp, and mailto for staging files out.

A sequence diagram for staging GFFS files is shown in L3D 5.3.9, Running a job directly on a BES. The sequence diagrams for other protocols are similar. Basically the BES parses the JSDL to determine the set of files to stage in. In then copies the files to the location specified in the JSDL, by default into the session directory.

Security note. As per the SD&I task 75, GridFTP Proxy Certificate Delegation, certificates are transferred through the call chain on BES:CreateActivity.

3.2.2 Variant UCCAN 1.b - Access to Remote Data During Execution

We interpret this requirement as a remote job, for example being executed at TACC or a campus cluster, needs to be able to directly read and write files and directories that are not "local", i.e., cannot be NFS mounted, GPFS mounted, or Lustre mounted, from the compute nodes. Such "access" is to be via POSIX IO.

Additional assumptions:

1. BESs that support mounting the GFFS add the XML element "gii-bes: unsupported filesystem GFFS" to their resources properties (L3D 4.1.7) and to their BESResourceAttributesDocument (L3D 5.1.2.3).

2. FUSE is installed and enabled on compute nodes on which jobs will be executed, i.e., on the compute nodes in a cluster, VM's to be used by a CloudBES, or desktops running BES services. Note that not all compute node operating systems, e.g., Kracken, support FUSE.

Steps

1. The user specifies in the JSDL that they want the GFFS file system mounted using the Genesis II access layer client GUI JSDL tool (LD3 3.3.2.5, GORM E.5.1) or a text editor.

2. The client submits the job to a BES as described in (L3D 5.3.9, 5.3.10, GORM E.5.3, E.5.7, E.5.8).

---

¹ A CloudBES is a resource that implements the BES interface and executes activities on virtual machines in a cloud environment such as EC2 or Azure. At least one such implementation is available.
3. The BES generates modified submission scripts to mount the GFFS and link it into the session directory before the application is started using the Genesis II command line client (GORM E.3.3) and unmount the file system after execution. In the case of MPI jobs each compute node must independently mount the GFFS locally. The modified submission scripts or the BES must also verify that the attempt to mount GFFS succeeded, otherwise the job may fail silently (e.g. by writing to the session directory instead of a GFFS resource).

4. The application opens, closes, reads and writes files from */GFFS.* (L3D 3.4.2, 5.3.3, 5.3.4, 5.3.5) The files and directories may be anywhere in the GFFS.

### 3.2.3 Variant UCCAN 1.c - compute resource is not known a priori

This use case has two sub variants: 1) where the client delegates the task of resource selection to some other meta-scheduling service such as a global queue or scheduler that optimizes some objective function, or 2) where the client discovers by some means, e.g., directory services or information services, the set of resources that match the job requirements. Background sections in the L3D to consider are 4.1.7 Reflection and Discovery, 5.1.2 Interfaces - Execution Management and 5.2.1 Components – Execution Management and 5.3.10 Qsub a job on a grid queue (GQ).

Note that there are no differences in the JSDL required for the two sub-varients per se, though some schedulers may look for some matching parameters. (Matching parameters are similar to Condor class ads.)

**Alternative A:**

Additional assumptions:

1. A grid queue **GQ** (L3D 5.2.1) is operational and the GFFS path to the **GQ** is known to the client. For example the Grid Queue at NCSA shown in Figure 1.
2. The client has permission to use GQ and the BES resources on which GQ schedules jobs.

Steps:

1. Create a JSDL (Job Submission Description Language, LD3 5.1.2.2) job description using a text editor or Genesis II access layer client GUI (LD3 3.3.2.5, GORM E.5.1).
2. Run and manage the job via the Grid Queue as described in (5.3.10, GORM E.5.3, E.5.4, E.5.5, E.5.7).

**Alternative B:**

Additional assumptions:

1. BES resources are arranged in a directory structure in which leaf directory entries point to the BES resources.
2. The client knows the path to the directory structure of BESs they want to use.

Steps:
1. Create a JSDL (Job Submission Description Language, LD3 5.1.2.2) job description using a text editor or Genesis II access layer client GUI (LD3 3.3.2.5, GORM E.5.1)

2. The client traverses through the directory structure of BESs, calling either `getFactoryAttributesDocument()` (L3D 5.1.2.3) or `getMultipleResourceProperties()` (L3D 4.1.7) on the BESs and then selects the best (for the client) BES to use.

**Alternative C:**

Additional assumptions:

1. An as yet unspecified information service exists that supports general queries.
2. An information service resource **info** exists at a path known to the client.

Steps:

1. Create a JSDL (Job Submission Description Language, LD3 5.1.2.2) job description using a text editor or Genesis II access layer client GUI (LD3 3.3.2.5, GORM E.5.1)

2. The client issues a query based upon the JSDL resource requirements against **info**, retrieves a list of BESs and their associated resource properties that the client wanted, and then selects the best (for the client) BES to use.

### 3.2.4 Variant UCCAN 1.d - Use Group Credentials

In X-WAVE, group membership is proven in the same manner as user identity, via a SAML assertion signed by a particular X.509 (L3D 5.1.5, 5.2.5), (GORM E.2, G.1). As described in 5.3.2.1 (Authentication – General Case), step 3.1.2, during the final phase of authentication group authentication may take place. This involves using the user credential just acquired to go to each group listed in the user’s identity directory, e.g., /users/grimshaw, and attempting to acquire a delegated group credential. For more information on group creation and management see GORM G.1 User and Group Management.

Thus, for a user to use a group credential the user must first add the group credential to their credential wallet (GORM E.2.1 Credential Wallet). This is done during login.

Steps:

1. The user is authenticated as described in L3D 5.3.2.2 (XSEDE Portal ID Case). This means that the client session has an XSEDE MyProxy session certificate as well as delegated SAML certificates from an `Kerb.AuthNPortType`. Once user identity assertions are acquired, group credentials are acquired and added to the users’ credential wallet (GORM E.2.1 Credential Wallet, 5.3.2.1 (Authentication – General Case), step 3.1.2).

2. Create a JSDL (Job Submission Description Language, LD3 5.1.2.2) job description using a text editor or Genesis II access layer client GUI (LD3 3.3.2.5, GORM E.5.1).

3. Run the job as described in (L3D 5.3.9, 5.3.10, GORM E.5.3, E.5.7, E.5.8).

4. The additional credentials in the credential wallet are automatically transferred and properly delegated through the call chain. The most interesting difference is in the BES implementation that executes the job.
5. If the BES on which the job executes has the property that it runs jobs “as the user” the BES must choose between the multiple identities (end-user, group1, group2, etc.) to use as the execution identity, e.g., it must choose an entry in the grid-map file.

3.2.5 Variant UCCAN 1.e - Registration Notification

This option is supported via a JSDL [http://www.ogf.org/documents/GFD.136.pdf](http://www.ogf.org/documents/GFD.136.pdf) extension and is described in the OGSA BES specification [http://www.ogf.org/documents/GFD.108.pdf](http://www.ogf.org/documents/GFD.108.pdf), section 7.2 page 22. Clients can subscribe to activity state change notifications by placing a notification request in the JSDL. Both the UNICORE 6 and Genesis II BES implementations support the WS-Notification option described in GFD 108. Note that the contents of the notification is being profiled.

3.2.6 Variant UCCAN 1.f - Interact with Job Session Directory

The sequence of steps is the same as for the base case, create a JSDL and run it on a BES or a grid queue. This use case makes an additional assumption.

Additional assumption:

1. The BES on which the job executes supports the RNS and ByteIO protocols (L3D 5.1.1.1, 5.1.3.1) in a manner similar to the BESActivityPortType (L3D 5.2.1.2) and the under development OGF BES Directory Protocol (BDP).

Additional Steps:

1) The client access the job session directory via the grid command line tools (L3D 3.2.3.1), GUI (L3D 3.3.2.1) or the FUSE driver. An example using the FUSE driver is given in L3D 3.4.2.

3.2.7 Variant UCCAN 1.g - At Most Once Semantics

To implement this requires BESs and grid queues to reject activities (jobs) from clients that have the same Job Name field. This in turn requires the BES or grid queue to keep track in a data base the job names used by each user over some period of time. The duration of this interval will be a configuration parameter for the BES and shall be exposed via the IdemPotent_Duration resource property of the BES.

3.2.8 Variant UCCAN 1.h - Operations on sets of jobs

Operations on sets of jobs are directly supported by BESs (L3D 5.1.2) and grid queues (L3D 5.1.8.1).

3.2.9 Variant UCCAN 1.i - Parameter Sweep

This option is directly supported in JSDL ([http://www.ogf.org/documents/GFD.149.pdf](http://www.ogf.org/documents/GFD.149.pdf)) and is directly supported in the Genesis II access layer client GUI (LD3 3.3.2.5, GORM E.5.1), grid queues (L3D 5.2.1), and BESs (L3D 5.1.2).
3.3 Quality of Service Attributes

For many of the quality attributes discussed below, 3.3, 3.4, 3.6, the implementation is critical to achieving the desired quality of service. In particular, execution services in X-WAVE (L3D 5.1.2, 5.1.8.1, 5.1.8.2) consist of Basic Execution Services (BESs) grid queues (GQs) and workflow managers. Each of these services requires that state about activities, workflows, etc. be correctly maintained (persisted) across system failure and restart. Therefore, implementations SHOULD maintain all state within a relational database management system with full ACID transactional support.

Thus, once an activity has been created via a createActivity (L3D 5.1.2) call and the call has returned to the client information about the activity is safely stored in the database. Unless the database is lost or corrupted the activity will not be lost.

Transactional relational databases are designed to avoid corruption by very carefully handling IO. That said, they usually presume certain properties of the underlying IO system (flush only returns once the data is on stable storage, etc.)

We have found that the assumptions that relational databases such as Derby make are not met by NFS mounted file systems, and that the database may become corrupted when layered on top of NFS. It is unknown whether Luster shares this problem with NFS. THEREFORE, to meet those quality attributes that depend on the database REQUIRES that the container on which execution services will run have locally attached storage for the relational database.

3.3.1 QAS-CAN1.a Any request to the execution service is acknowledged within one second.

Additional Assumptions:

1. Network round trip of at most 100 mS and a TCP bandwidth of at least 1 Mb/S.
2. Minimum Grid Interface Unit requirements met (memory, clock speed, cores, local attached disk).
3. One or more cores free on GIU, i.e., the load on the GIU is less than the number of cores.

This is a performance quality attribute that requires that the total time for the createActivity call on the selected Basic Execution Service (BES) takes less than one second.

The Web Services RPC in X-WAVE can be broken down into four steps (L3D 4.1)

- Client marshals and XML encodes the arguments to the requested operation call, creates if necessary a SSL/TCP socket to the container in which the BES is executing, delegates and signs SAML assertions as necessary, and sends the resulting SOAP message over the SSL socket.
- Container decrypts/receives the SOAP message, extracts security headers, checks if the client is authorized to perform the specified function on the specified grid resource, and if so calls the appropriate web service function.
- The container performs the requested operation updating state as necessary.
- The requested operation constructs a reply, the container encodes the reply in XML and sends the reply back down the encrypted SSL socket established in the first step.
These steps are the same for all method invocations. Besides the load on the server and the time to perform the requested operation the factors that vary between invocations that impact performance are:

1. the number of SAML credential chains to delegate, sign, and transmit,
2. the size of the arguments and return values
3. the time taken to perform the requested operation
4. the network “distance” between the client and the service.

Most of these are self explanatory. The network distance for functions with relatively small arguments and responses (under TCP buffer size) is dominated by the round-trip packet time.

In a local area environment with 100Mb/1Gb Ethernet and two SAML chains the round trip Web Services“ping” time for a simple method invocation, going up and down the call stack, delegating certificates, encoding parameters, etc. on an idle server\(^2\) is approximately 50 mS. The round trip IP ping time is under 1mS, usually around 200 microSeconds. Thus the total Web Services stack overhead is on the order of 50 mS.

In a wide area environment, e.g., Virginia to SDSC, the round trip IP ping is usually on the order of 60-80 mS. The Web Services round-trip time is usually on the order of 500-600 mS if the SSL session has not already been established.

Thus we can see that our ability to meet this quality attribute is depends on the network bandwidth and latency. There will be conditions under which it is impossible to meet the 1 second goal.

We require therefore a network round trip of at most 100 mS and a TCP bandwidth of at least 1 Mb/S.

To determine the time to complete the operation and the amount of data to be transmitted we must identify the operations and their variants.

We will assume createActivity and getActivityStatuses are the operations of interest (L3D 5.1.2.3). Both operations can take lists of parameters, JSDL documents in the createActivity case and EPRs in the getActivityStatuses case.

Neither the execution time of createActivity nor getActivityStatuses can stay under the specified 1 second execution time for an arbitrarily large number of activities to start or for which to retrieve status. Therefore, we limit our analysis to the case of creating a single activity or getting status about a single activity.

In the single activity createActivity case the major cost is creating a new grid resource, generating its’ EPR, and storing all of its state (e.g., the JSDL document, the calling security context, etc) into the relational database. Collectively these operations take approximately 200 mS.

Measuring the quality attribute:

There are already scripts in the Execution Management Services (EMS) testplan that carry out these tests. Currently they are not timed. They could be. Alternatively, from the grid shell one could execute
time run –asynchronous …. 

\(^2\)Unless noted otherwise assume an idle server, i.e., the client is the only client using the server.
This should be executed twice and the second result used so that all of the paths will have been looked up and cached.

3.3.2 QAS-CAN1.b   High-throughput: 1) rate jobs can be submitted, 2) the number of active jobs, 3) the total number of jobs the service can manage.

Three critical benchmarks have been indentified:

- Job submission rate. How many jobs per second can a grid queue accept?
- Job completion rate. Given a queue with many jobs ready to run, how fast can the queue schedule new jobs as old jobs complete?
- Queue size limits. Some queuing systems have limits on the number of jobs. The queue should be able to hold at least 12 hours of pending jobs, i.e., 120,000 jobs.

We have constructed a benchmark for each. Below we describe the testing methodology and testing infrastructure.

Each benchmark was executed on the University of Virginia campus grid. At the time of the test, the grid queue was located on a dual eight core Xeon 2.4 GHz Dell with 16GB of main memory. Each machine in the queue has a single Gb Ethernet connection to the CS department switching infrastructure. The CS department in turn is connected to the gigabit Ethernet university infrastructure via a packet filter that can operate at speeds of between 100 and 500 megabits per second.

In order to maximally stress the queue management system we constructed a JSDL job description for an “empty” job. The empty job requires no inputs, and creates no output. We use a single line .bat or batch file which echoes “Hello, World” to the console (we throw away the output).

To test the job submission rate, we created two JSDL documents. One was a JSDL document for a single empty job while the other was a batch JSDL document consisting of a selection of job counts (the queues will accept either single JSDL jobs in a document, concatenated lists of jobs in a batch document, or a JSDL parameter sweep). We then measured the submission rates for a client submitting batches of jobs using the JSDL parameter sweep extensions. In both cases the client was on a 1.6GHz Opteron running Linux.

In the short or singleton job case, we were able to achieve a job submission rate of approximately 3.80 job submissions per second. This represents the average of three measured results with the highest and lowest of five thrown out.

In the multi-job case, we collected results for batch submissions of 1000 jobs in a batch and 10,000 jobs in a batch. These took 37 seconds (27 jobs/second) and 270 seconds respectively (37 jobs/second).

Measuring job completion rate is a bit trickier because there is no way to turn on and off a queue. Instead the queue can be configured to limit the number of active jobs on a resource it is using to 0. When all resources are at 0, the queue is stopped.
To measure the queue completion rate, we used ten local Linux machines with no other external load. We started by configuring each resource in this queue with a 0-slot configuration, thereby effectively stopping the queue. We then placed 10,000 “empty” jobs in the queue. Finally, we re-configured the queue so that each of its resources had ten slots per compute node. As part of its normal routine, the grid queue logs various bits of information regarding its processing of jobs, one of which is a message about job completion. By inspecting the count of these jobs completion log events, and the time stamps associated with them, we are able to determine an average job completion rate for the queue. This technique slightly underestimates the job completion rate because it takes several seconds to reconfigure the queue resources after the jobs have been submitted, but given the overall time of the test (approximately 36 minutes), this discrepancy should largely be amortized. Using this method to measure job completion rates we were able to achieve a completion rate of approximately 5.55 jobs per second.

Finally, we verified the total job containment requirement by enqueing 120,000 jobs into a stopped grid queue and verifying that once re-started, the queue was able to make progress on jobs (i.e., that the jobs were completing).

3.3.3 QAS-CAN1.c The execution service can support, without error, as many queued and active jobs are permitted by its associated resource management system.

As described at the beginning of section 3.3 EMS services such as BESs store their state in a RDBMS. Information about jobs is stored in the relational database, the number of jobs that can be stored is limited by the storage available for the RDBMS. This is for all intents and purposes not a limit. For example, suppose that an average job requires 10K of disk storage (the bulk of that amount is typically the signed SAML assertions that make up the security context for the job.) More than 10,000 queued jobs is very unusual at XSEDE service providers, and there are typically fewer than 1,000. Assuming a 10,000 job limit at an SP, and 10K Bytes/job, that will require on order $10^8$ bytes in the database assuming that no attempt is made to eliminate duplicate information (e.g., security context information). That small amount of data will fit on a thumb drive!

To test this quality attribute simply submit a large number of jobs to the BES as above in 3.3.2.

3.3.4 QAS-CAN1.d Once a job is complete its status can be checked for at least 24 hours.

As described at the beginning of section 3.3 EMS services such as BESs store their state in a RDBMS. Keeping job information for at least 24 hours implies keeping job information in the RDBMS for 24 hours. The total number of jobs for which information must be maintained therefore is the maximum completion rate in jobs per second, times the number of seconds in day (86400). At a rate of 5 jobs/second (more than most centers sustain) this means maintaining information on the order of 400,000 jobs. Assuming adequate storage most RDBMSs can maintain 400,000 records easily. For example, suppose each job requires 10K as in 3.3.3. 10K is an over estimate as the bulk of the data is security context, which can be discarded once the job has completed. 10K X 400,000 = 4X10^9 bytes, a small database.

A more important issue is the ability to list and display 400,000 jobs. To be able to display that many job entries in the client will require the client to be configured with more than the default
amount of memory. If 400,000 jobs are to be displayed we recommend that the client program be configured with at least 4GB of memory.

3.3.5 QAS-CAN1.e Client request patterns that exceed the stated job submission rate, queued jobs, or active jobs, are handled gracefully.

We interpret “handled gracefully” as meaning that as the load increases the time to complete any given service will increase, and at some point some invocations to either create new jobs, or query the status of existing jobs, may fail. This is common practice in industry. When a web site is overloaded one often observes a browser timeout. We also believe though that while the load may cause job create and status requests to fail that no jobs already accepted into the system should be lost.

Again recall that Execution services in X-WAVE (L3D 5.1.2, 5.1.8.1, 5.1.8.2) consist of Basic Execution Services (BESs) grid queues (GQs) and workflow managers. All EMS services operate as web services in a container (L3D 4.1). All state is kept within a relational database management system with full ACID transactional support.

Web service containers can be configured with a maximum number of acceptor threads. The name differs between containers, but the concept is shared. The number of acceptor threads determines the maximum number of active client connections the container has open at any given time. When a request arrives at the container and the maximum has already been reached the container defers the new connection. There are two ways this can be done. First the connection is established and the container immediately returns a “503 Service Unavailable” or programmer defined string such as “Busy – try again later”. Alternatively, the connection may be refused.

The Web Service client SHOULD be configured to handle BUSY faults by first checking if there is a replica. If so, call the replica instead. If not enter an exponential back-off loop for some number of tries, e.g., 5 tries. If the service is still unavailable, throw a fault.

Assuming the request results in a connection, each web service request to a BES (e.g., create activity, destroy activity, get status) results in a transaction being executed against a relational database. If the load is so high that that the web service request times out, the transaction will be aborted, and the underlying database will not be updated.

3.3.6 QAS-CAN1.f The execution service can be restarted without loss of jobs at three Sigma.

(Note: in this case does three sigma refer to restarts, as in within 1000 restarts at most one job can be lost, or does it mean at most one job in 1000 will be lost?)

Recall execution services (L3D 5.1.2, 5.1.8.1, 5.1.8.2) maintain their state in a relational database management system with full ACID properties. Thus, once an activity has been created via a createActivity (L3D 5.1.2) call and the call has returned to the client information about the activity is safely stored in the database. Unless the database is lost or corrupted the activity will not be lost.

3.3.7 QAS-CAN1.g Valid jobs complete successfully at two Sigma.

The definition of a valid job is meant to capture the notion that there is nothing wrong with the job description, the input and output file services, and that the job will execute to completion without failure on the selected execution resource. In other words, if the execution service does its job of staging the data and queuing up the job then the job will complete normally.
Thus, for a valid job to fail implies that the execution service has failed – i.e., the parameters were correct yet it did not complete its function properly.

One challenge with measuring this quality attribute is that the set of valid JSDL strings formed by the JSDL language specification is infinite – so exhaustive testing is out of the question. Instead a representative subset must be constructed and executed against the execution management system. This is the approach that X-WAVE has taken.

SD&I has developed a set of regression tests for EMS. The tests are described in GORM I.3 Running the XSEDE Regression Test. Specifically there are both GFSS and EMS tests. The test suite includes tests for sequential jobs, parallel MPI jobs, jobs that stage files in and out, jobs that will fail, and multi-user job tests.

As a more ad-hoc “proof”, on November 3, 2013, at 9:01 CET the grid queue for the cross-campus grid hosted at Virginia showed

- 13120 Jobs Currently Queued
- 1 Jobs Currently Re-queued
- 27 Jobs Currently Starting
- 744 Jobs Currently Running
- 8 Jobs Currently in Error
- 16101 Jobs Finished but Not Reaped

So, out of 16,101 jobs 8 jobs had entered an error state. That does not mean that though that those 8 jobs were valid jobs. We have found that often the problem with failed jobs is that the target file system is down or full\(^3\). It does mean though that over the interval since the users last checked their jobs (likely two or three days as this is Sunday) jobs have completed at a rate of almost 4 sigma. This is well over the requirement of 2 sigma.

### 3.3.8 QAS-CAN1.h  
**Availability of the execution service is 1.8 Sigma.**

Availability is usually defined as probability that the system will be available at some time \(t\) over some time interval called the *performance time*. In this case the “system” is the execution service. Often one also speaks of availability per demand – in other words the probability that any given service demand will be met.

When considering availability (and reliability as well) it is important to consider the *operational context*, i.e. the assumptions about portions of the system that are out of your control. For example, whether the Internet is functioning or there is power in North America. We will assume the operational context is normal – everything else is working fine.

To achieve 1.8 sigma availability we will use the techniques described in the L3D 5.5.3.2, Mitigation Strategies, including specifically the use of containers and transactional state, the use of a watcher process that automatically restarts Web Service containers if they fail, as well as hard-

---

\(^3\) To find the cause of the failure one can examine the job history. Currently UNICORE 6 and Genesis II have different job history formats. The Genesis II Job History Tool (L3D 3.3.2.3) provides a fine grain view of job execution. The Activity Endpoint Profile being developed in OGF will provide a standard job history format. Both projects will adopt it when it is finalized.
ware mechanisms such as UPSs, and if necessary for much higher than 1.8 sigma, replicated web servers and relational databases.

To measure availability a tool such as Nagios or Inca can be used to periodically “ping” BESs and grid queues, and, optionally, send a dummy job such as `ls` through the system.

3.3.9 **QAS-CAN1.i** If a compute resource associated with the execution service fails, then any job currently executing on that resource is reported as failed.

This is an implementation requirement more so than an architectural requirement. That said, typically this is implemented by a BES by periodically polling the local queuing system (e.g., SLURM or PBS) to determine the status of jobs. Unfortunately, if the queuing system fails to respond it does not mean that the underlying compute system has failed – or indeed that the queuing system has failed. Therefore BESs typically keep trying to communicate with the queuing system until they are successful, and at that point make a determination whether the jobs they (the BESs) started have completed or failed. Often this involves looking for status files that will have been written to disk when a job completes.

3.3.10 **QAS-CAN1.j** The job states must be consistent and well defined across all resources.

The OGSA Basic Execution Services specification (OGF GFD-R 108) specifies state model with five basic states: *Pending, Running, Canceled, Failed, and Finished*. Sub-states, such as *Running: Stage-In, Stage-Out* may also be defined.

X-WAVE further adopts the file staging profile defined in the OGSA BES specification.