TOG & JOSH: Grid scheduling with Grid Engine & Globus

G. Cawood, T. Seed, R. Abrol, **T. Sloan**

EPCC, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, UK

Abstract

TOG and JOSH are two fully Globus-enabled grid scheduling tools based around Grid Engine, the Globus Toolkit and a variety of data grid technologies. These tools were originally developed by the EPCC Sun Data and Compute Grids (Sun DCG) project. This paper describes these tools and their use in e-Science projects in the UK and abroad.

1. Grid Engine & Globus

Grid Engine [1] is an open source distributed resource management system that allows the efficient use of compute resources within an organisation. However, Grid Engine, alone, does not provide the means to share resources between collaborating organisations or enterprises. The Globus Toolkit [2] is essentially a Grid API for connecting distributed compute and instrument resources. Integration with Globus enables Grid Engine to provide this collaboration amongst enterprises. Sections 2 and 3 of this paper describe the TOG and JOSH tools respectively. These tools integrate Grid Engine with the Globus Toolkit to enable collaborating organisations to share resources.

2. TOG : Transfer-queue Over Globus

2.1 Overview

TOG [3] integrates Grid Engine V5.3 and Globus Toolkit V2.2.x to allow access to remote resources. This allows an enterprise to access remote compute resources at any collaborating enterprises. This is achieved by configuring queues on a local Grid Engine to use the TOG. TOG provides secure job submission and control functionality between the enterprises. TOG enables an enterprise to schedule jobs for execution on remote resources when local resources are busy. Data and executables are transferred to the remote resource with subsequent transfer of results back to the local installation. Users submit and control jobs at both local and remote resources via their existing Grid Engine user interface. Remote system administrators still have full control of their resources. TOG has been available in both binary and source downloads from the open source Grid Engine community web site [3] since July 2003. The binary download contains the TOG binaries, API documentation and a How-To document detailing installation and use of TOG. The source download contains the TOG source, systems tests that integrate with the Grid Engine test suite, design document, test plan and the How-To document.

2.2 TOG System Outline

This section briefly describes how TOG integrates Globus Toolkit V2 and Grid Engine V5.3. A fuller description of TOG's design is available in the Prototype Design document [4].

In Figure 1, A and B are distinct enterprises on separate domains. A is the local enterprise where the addition of TOG allows users at A to access resources at B through the Grid Engine at A. The resources at B are classified as remote from A. Globus Toolkit V2 provides the secure communication and data transfer between A and B. This is achieved by creating a Grid Engine queue at A that acts as a proxy for a Grid Engine queue at B. This proxy queue is configured to use TOG to run the job on the queue at B.

TOG uses the Grid Engine *'execution methods'* to intercept jobs and control operations submitted to the proxy queue at A. TOG transfers the intercepted jobs and operations and runs them on the corresponding queue at B and returns the results to A.

Execution methods allow a third party to intercept job submission and control operations on a queue, and in doing so extend the functionality of a Grid Engine queue. In Grid Engine terminology a queue that passes a job to a third-party is known as a '*transfer queue*' [5].

At enterprise A the following applications are involved:

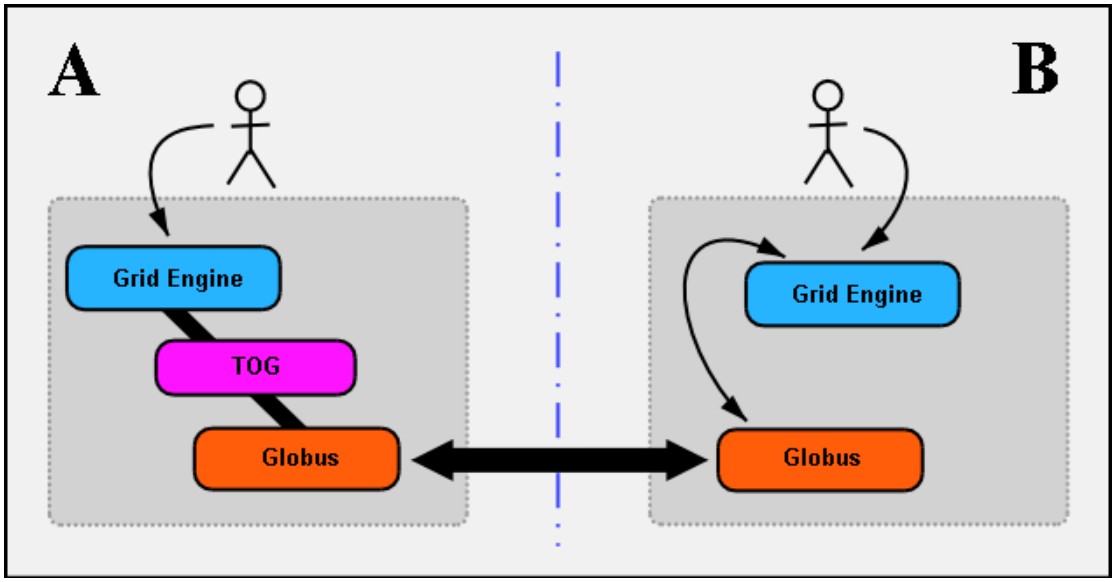

Figure 1: TOG Integration

- Grid Engine – manages the resources local to enterprise A. This is enhanced with TOG to make remote resources available.
- Globus Toolkit V2 – provides the secure communication and data link between the enterprises.
- TOG – makes the remote resource available to the local Grid Engine using Globus Toolkit V2.

At enterprise B the following is set up:

- Grid Engine – manages the resources local to enterprise B.
- Globus Toolkit V2 – provides a secure interface to those resources via the Grid Engine at B.

TOG uses the Globus Java CoG API to make use of the following Globus facilities:

- GSI – provides secure user authentication and confidential communication
- GRAM – creates and interacts with a Globus job manager on the remote resource.
- GridFTP – provides large dataset transfer and third party transfer.
- GASS – A GASS server and client provides a virtual shared file system between the local and remote resource. It is suited to small file sizes.

At enterprise B Globus uses Grid Engine as its job manager.

TOG comprises a suite of scripts that activate Java classes. The scripts are configured using Grid Engine *'complexes'* associated with the proxy queue. This configuration includes location of the remote Grid Engine and location of scratch space for transferred data on the remote site.

2.3 Data Transfer

TOG allows a user to specify the job input files to be transferred to the remote site and the output files to be retrieved on completion of the job. This is achieved by placing special comment lines within job scripts. This mechanism is familiar to Grid Engine users as it is similar to Grid Engine's own mechanism for allowing users to embed Grid Engine arguments within job scripts; the only difference being that Grid Engine arguments must be prefixed with #$, and TOG arguments must be prefixed by #%. An example of specifying an input file within a job script is:

```
#%input_file=/home/thomas/script.sh
```

2.4 Using TOG

TOG has been used successfully in a number of grid related projects most notably the OGSA-DAI Demo for Genetics (ODD-Genes) [6] where it is used to schedule micro-array analysis jobs to high performance computing resources. More recently in the INWA project [7] TOG is enabling commercial financial and telecoms data to be mined at sites in the UK and Australia. Others projects to have reported using TOG successfully include ULGrid at the

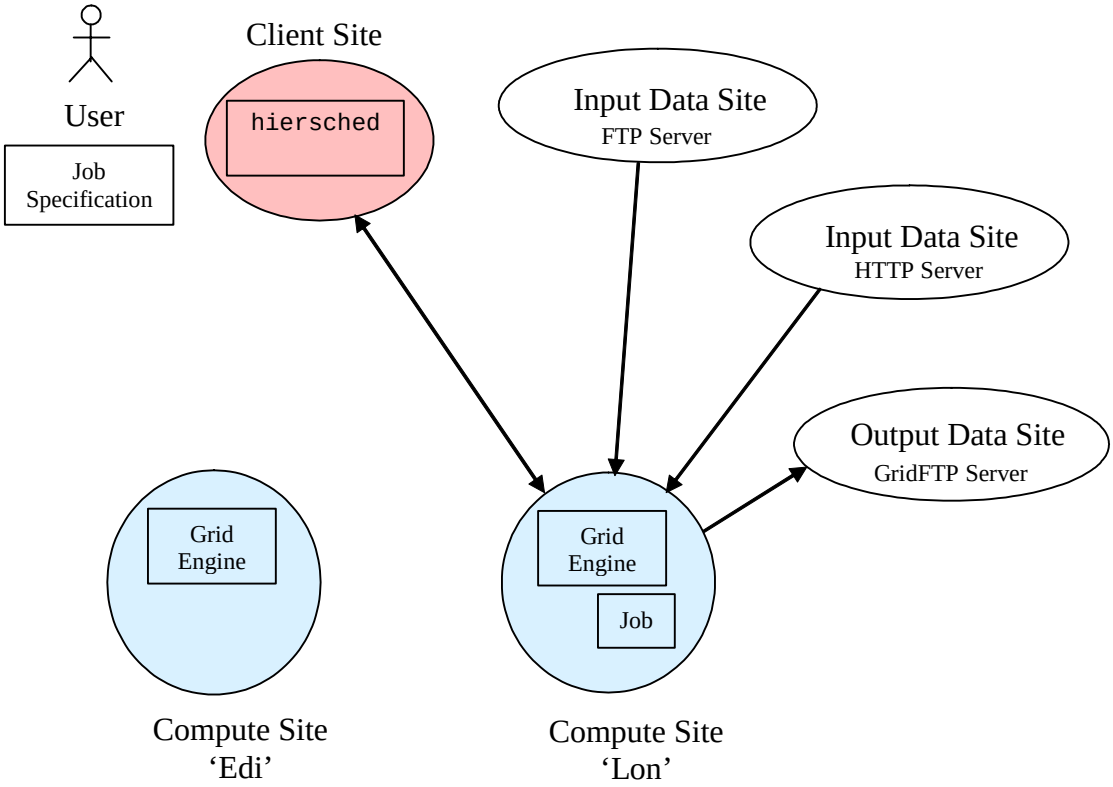

Figure 2: The hiersched tool selects Compute Site 'Lon' to run the user's job.

University of Liverpool [8] and Poseidon at MIT in the USA [9].

3. JOSH : JOb Scheduling Hierarchically

3.1 Overview

JOSH (JOb Scheduling Hierarchically) [10] improves upon the recognised limitations of TOG by scaling better in a grid environment and enabling access to remote third-party data sources via anonymous ftp, http and data grid technologies. OGSA-compliant Globus Toolkit V3.0 handles communications and data transfer between JOSH, Grid Engine and any remote data sources. JOSH matches a user's job requirements against Grid Engine instances at available compute sites. Three criteria are used to evaluate sites: capability, load and proximity to data sources. A job is then sent to the chosen compute site for local scheduling and execution. Before execution, any input files are pulled to the compute site from their data sources. Similarly, output files are pushed to their target data repositories after the job has completed. JOSH has been available in both binary and source downloads from the open source Grid Engine community web site [10] since February 2004. The binary downloads contain JOSH binaries, User and Installation Guides and the JOSH limitations document. The source downloads contain the JOSH source, build files, the Build guide, and all other user documentation. Also available is the javadoc built from source. The JOSH Functional Specification [13] and Systems Design [12] are available separately.

Figure 2 shows the sites involved in a JOSH system. The JOSH hiersched tool runs at the client site where the user submits their Job Specification. A Job Specification is a Grid Engine job script embellished with any data requirement comments similar to those used in TOG (see Section 2.3). After deciding on the best compute site, hiersched forwards the job specification to the chosen compute site for execution. One Grid Engine runs at each compute site. A grid service known as the JOSH Site Service also runs at each compute site. The hiersched tool invokes these services to interact with the underlying Grid Engines.

Any site which provides suitable access to its files can be considered a 'data site'. The compute site where a job specification executes is automatically a data site since files can be read or written directly from or to its local disc. An external site may act as a data site if it provides internet access to files via one of the following: a GridFTP server, an anonymous

FTP server, an HTTP server (for read access only).

In the example of Figure 2, hiersched decides that compute site Lon is a better prospect than Edi for the job specification in question and so submits it to Lon. Any input data sets noted in the job specification are automatically pulled to Lon just before the job runs. Similarly, output files are pushed from Lon to the chosen output data site once the job has completed.

In Figure 2 there is only one client site. However there may be multiple client sites, all operating over the same pool of compute sites. This is how organisations can mutually share their compute resources. Note also that a site may operate as both a client site and a compute site. In Figure 2 the pool of available compute sites consists of two sites {Edi, Lon}. However, there may be arbitrarily many sites in the pool. The pool is defined during the client installation procedure by providing a file called `hsconfig.xml` which maps shorthand names such as 'Edi' to long site addresses. Users need only concern themselves with shorthand site names when referring to specific compute sites.

3.2 JOSH Architecture

This section provides an overview of the architecture of the JOSH system. A fuller description is available in the JOSH User Guide [11] and System Design [12]. Figure 3 illustrates the main sites, executables, components, inputs and outputs of the JOSH system. At the client Site the user interacts with the hiersched command-line executable, invoking it once per command. This is written in Java for ease of calling grid services. The hiersched tool's CLI (Command Line Interface) performs some simple parsing on the user's commands and arguments before passing them on to the main engine for processing. In order to satisfy these commands, the main engine has to communicate with Compute Sites. This is done via the Comp Site component which hides the grid services calls being made behind the scenes.

At each Compute Site a Grid Services Container runs continuously where it offers a persistent JOSH Site Service which provides a number of query-like operations relating to the status and capabilities of that Compute Site. The implementation of these operations involves calling out from Java to other executables such as 'ping' for the data transfer time estimates and Grid Engine's 'qsub' (with appropriate arguments) for determining the site's physical ability to run a job specification.

The operations of the JOSH Site Service are normal grid service operations and so execute as the 'container owner' ie. the user that launched the Grid Services Container. Some functionality, however, such as job submission, must execute as the 'mapped user' ie. the account at the Compute Site owned by the user at the client Site. JOSH uses the Globus Toolkit V3's Managed Job Service (MJS) to run these job submission and termination scripts on behalf of the client user. This mechanism allows a script to be run as the mapped user if appropriate security credentials are available. JOSH v1.1 includes scripts for job submission and termination. Of particular note is the job Submission Script which submits three Grid Engine jobs (PRE, MAIN and POST in Figure 3) for a given job specification. The PRE and POST jobs are auto-generated and manage the transfer of input and output files, while the MAIN job is the user's original Grid Engine job script. On execution, the PRE and POST jobs call out to JOSH's 'datatran' executable, which in turn calls out to 'globus_url_copy' to perform the actual transfers.

Figure 3 also demonstrates a typical 'flow' through the system. At the client Site the user prepares a Job Specification in the form of Grid Engine job script embellished with any data requirement comments similar to those used in TOG (see Section 2.3). This is passed to the hiersched tool as part of a 'submit_ge' command. To implement this command the main engine must firstly communicate with all the candidate Compute Sites it knows about (by virtue of their presence in the Hiersched Config File) to decide which site should ultimately run the job. Each Compute Site is asked for an overall 'suitability score', and the hiersched can then straightforwardly choose the site with the best score. Determining a score involves a number of steps, the most fundamental of which is to establish a site's physical ability to run the job, as captured by the 'canRun(...)' operation on the JOSH Site Service.

A Compute Site's proximity to the job's data file sources and sinks is also a factor in calculating a compute site's suitability score. An estimate of the time required to transfer data is calculated by the dataTimeScore(...) operation. This calls out to 'ping' to gain an estimate of the round-trip times between the candidate Compute Site and the relevant Data

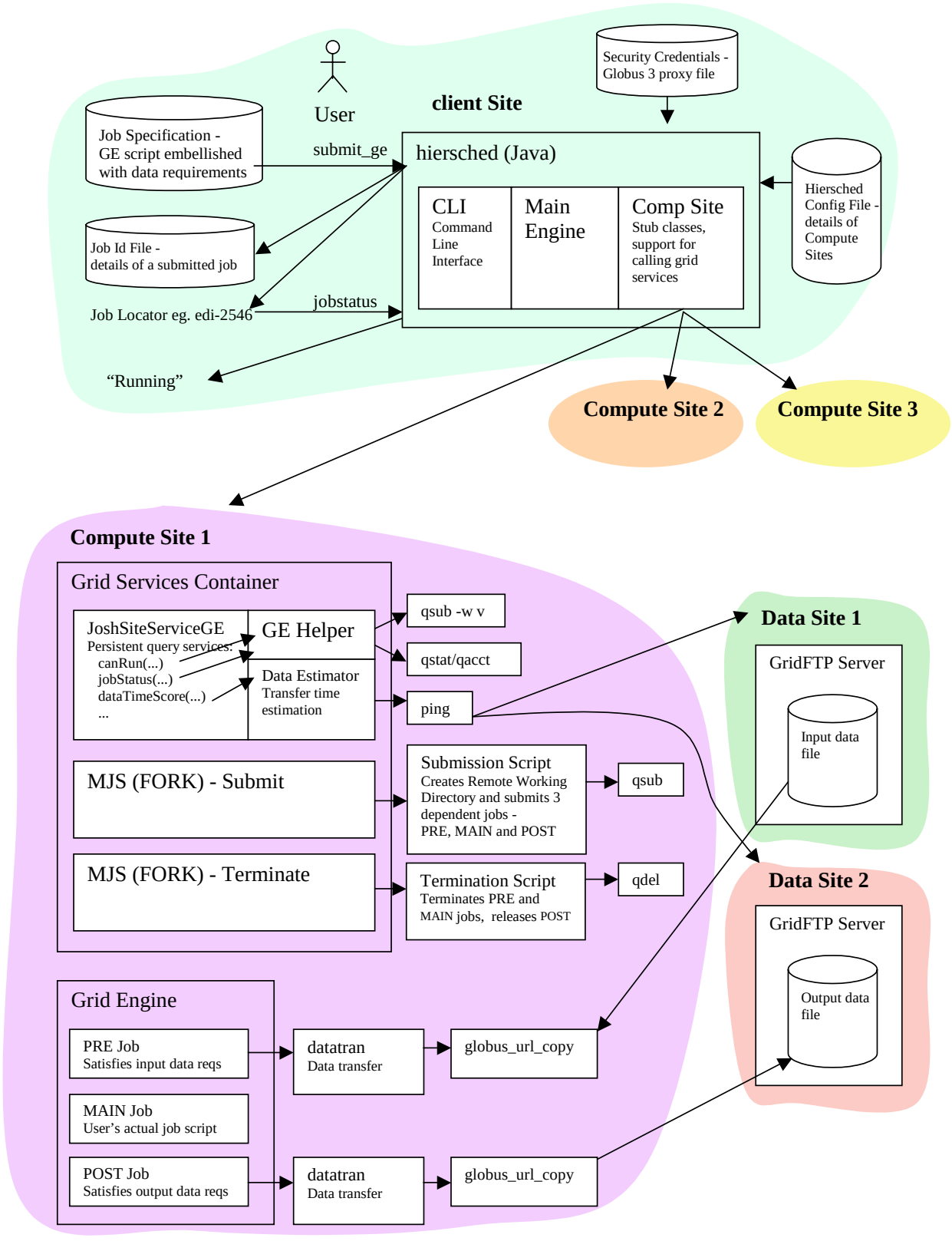

Figure 3: An informal picture of the JOSH system architecture

Sites. Times are then scaled up by the amount of data to be transferred. Having determined the best site (Compute Site 1 in this case), the hiersched tool submits the Job Specification on behalf of the user. This involves calling a Globus MJS to fork a script which submits the

PRE, MAIN and POST jobs. Once the submission of these three jobs is complete, the hiersched submit_ge command returns with a Job Locator corresponding to the MAIN job. A Job Locator is a string of the form '<Compute Site Name>-<Job Id>' eg. 'edi-2536'. This locator allows the user to uniquely identify their job in future hiersched commands.

In the example of Figure 3, the next command called is 'jobstatus'. The main engine makes a call to the jobStatus(...) operation of the JOSH Site Service, which in turn calls out to Grid Engine's qstat and qacct executables to discover the job's status - "Running" in this case.

When the MAIN job has completed, the POST job will transfer output files to the user-specified Data Site (often the client Site), allowing the user access to their results.

3.3 Architecture Justification

This section attempts to justify some of the key design decisions which have been taken in arriving at the described architecture.

Combined Grid Service and MJS Script Approach: At the Compute Site, capability querying and status reporting are implemented by the JOSH Site Service (an ordinary Java service), while job submission and termination are implemented using bash scripts sent via the MJS.

Architecturally it would have been preferable to avoid bash scripts and the rather heavyweight MJS by implementing all the functionality as operations of the JOSH Site Service, but this was not possible due to an issue with user accounts.

Job submission must be run under the remote account of the user submitting the job, (ie. the 'mapped user') in order to ensure that the correct permissions are in place when the job is executed. However, call-outs from a Java grid service operation (to Grid Engine's 'qsub' command in this case) always run under the account of whichever user launched the grid services container. This behaviour led to a combined approach involving the MJS.

Three Grid Engine Jobs per Submission: A hiersched submission is implemented using three dependent Grid Engine jobs - Pre, Main and Post - as described earlier. This is done partly to satisfy [13] and partly for efficiency at the Compute Site.

[13] states that on user-termination of a submission, an attempt should be made to transfer partial output files. If the submission were implemented as a single Grid Engine job (with Pre and Post functionality pre-pended and appended as appropriate), then its termination would of course prevent the Post functionality from ever being executed.

With three jobs, the Pre and Main jobs can be terminated and the Post job can be released from its dependency. This makes the Post functionality ready to run straight away and it can attempt the output file transfers.

Also, the Pre and Post functionality does not require any special resources, whereas the user's Main job might have very exacting resource requirements. Again, if the submission were implemented as a single Grid Engine job then those resources would be captured for the entire duration but left idle while the Pre and Post functionality (data transfer) was executed.

Having 3 jobs also allows different jobs to be directed to different queues. It may be desirable to experiment with a dedicated Data Transfer queue in which the Pre and Post jobs can execute.

Globus MJS with Fork not GRD: The Compute Site functionality that is to run as the mapped user (ie. submission, termination, suspension and resumption) is implemented using the MJS with a *Fork* Job Manager rather than a GRD (Grid Engine) Job Manager, despite GRD seeming the obvious choice.

The GRD Job Manager would certainly make sense if a hiersched submission was to be implemented as a single Grid Engine job. However, three dependent jobs are used, and access to their Grid Engine Job Ids is required in order to set up the dependencies. With the MJS and the GRD Job Manager there is no way for a client to access those Job Ids and set the dependencies.

The use of Fork gives the degree of control necessary - the forked job can call qsub for the 'real' jobs and so get hold of the Job Ids.

However, the use of Fork may be considered undesirable. There are reasons (eg. fairness) why a Compute Site administrator might want to disable the Fork job manager and force all

external jobs to come in through a central job scheduling system.

However, in the JOSH case the forked jobs are all lightweight. They call a few Grid Engine commands and then finish so they should not place undue load on a site. Also, forking the suspension script (for example) is effectively the same as the user logging on to their remote account directly and typing qsub three times. So this approach is not really giving the user more powers than they already have.

Consultation with EPCC systems administrators suggests that most system administrators are likely to install Globus Toolkit V3 'out of the box', in which case the Fork job manager will be enabled.

Future extension of the interface to Globus jobs (to include dependency setting) might remove this need for forking.

Removal of Job Dependencies? It would be possible to devise a scheme which avoided the use of Grid Engine job dependencies by having the Pre job submit the Main job, which in turn would submit the Post job.

While the removal of job dependencies might simplify the solution in some ways (and perhaps allow the use of the GRD Job Manager instead of Fork) it has a number of disadvantages. Firstly, the termination of the Main job would prevent the Post job from ever being submitted, let alone executed. This would not satisfy [13], and any workaround would probably require significant complexity at the client Site to somehow submit a 'clean-up' job to mop up the terminated job.

Also, the delayed submission of the Main and Post jobs could result in longer runtimes overall. Submitting all three jobs at the same time allows the scheduling priority of the Main and Post jobs to rise with their longer time spent in the Grid Engine pending list.

3.4 Globus Toolkit V3.0.2 Limitations

JOSH was developed and tested with Globus Toolkit V3.0.2. This revealed a number of issues with this particular release of the Globus toolkit. Some of these issues are outlined below.

Slowness: Connecting to Globus services is slow. Globus Toolkit V3.0 is immature and execution speed was clearly not a priority. Note that any sluggishness in launching a job does not affect the runtime of the job itself, which is dependent only on the resources and load at the chosen compute site.

Security: JOSH uses globus-url-copy to perform the transfer of a job's input and output files. Unfortunately globus-url-copy in Globus Toolkit V3.0 does not supply an option to allow the data channel to be encrypted. This means that the user's data could potentially be exposed during the transfer operations.

Robustness: If the MJS's proxy certificate expires at the compute site then the associated user's job can no longer be contacted. This prevents the generation of further MJSs for other user's jobs.

Since the completion of the Sun DCG project Globus Toolkit V3.2 has been released. This includes a number of bug fixes and enhancements tackling the above issues.

3.5 Using JOSH

A number of organisations have reported that the performance and robustness of the Globus Toolkit V3.0 has given them cause for concern and as result they have reluctantly decided not to consider JOSH for deployment.

JOSH take-up will be further affected by the switch to WS-RF in Globus Toolkit V4.0 (GT4). The announcement of this change in January 2004 took place weeks before the release of JOSH and the subsequent completion of the Sun DCG project. Whilst JOSH could easily be modified to work with a WS-RF compliant GT4, the Sun DCG project was unable to do so since no implementation of GT4 was available by the time of the project close. GT4 is not due for release until quarter 4, 2004. However, despite these obvious concerns, at the time of writing (June 2004), a number of organizations are still exploring using JOSH in their future grid projects. These include the previously mentioned ULGrid [8] and Poseidon [9].

Acknowledgements

The EPCC Sun Data and Compute Grids project [14] was funded as part the UK e-Science Core Programme [15]. The project partners are the National e-Science Centre [16], represented in this project by EPCC [17], and Sun Microsystems [18].

References

[1] Grid Engine, http://gridengine.sunsource.net/

[2] The Globus Alliance, http://www.globus.org/

[3] Transfer-queue Over Globus (TOG), http://gridengine.sunsource.net/project/gridengine/tog.html

[4] D4.2 Prototype Development: Design, EPCC Sun Data and Compute Grids project, available with source download from [3].

[5] C. Chaubal, "A Prototype of a Multi-Clustering Implementation using Transfer Queues", http://gridengine.sunsource.net/project/gridengine/howto/TransferQueues/transferqueues.html

[6] OGSA-DAI Demo for Genetics (ODD-Genes), http://www.epcc.ed.ac.uk/oddgenes

[7] INWA, http://www.epcc.ed.ac.uk/projects/inwa

[8] ULGrid, http://www.liv.ac.uk/e-science/ulgrid/

[9] Poseidon, http://czms.mit.edu/poseidon/new1/

[10] JOb Scheduling Hierarchically (JOSH), http://gridengine.sunsource.net/project/gridengine/josh.html

[11] JOSH 1.1 User Guide, available from http://gridengine.sunsource.net/project/gridengine/josh.html

[12] D4.2 JOSH System Design, http://www.epcc.ed.ac.uk/sungrid/PUB/D4_2-JOSHSystemsDesign.pdf

[13] D4.1 Functional Specification, EPCC Sun Data & Compute Grids project, http://www.epcc.ed.ac.uk/sungrid/PUB/D4_1-FunctionalSpecification.pdf

[14] EPCC Sun Data and Compute Grids project, http://www.epcc.ed.ac.uk/sungrid/

[15] UK e-Science Core Programe, http://www.escience-grid.org.uk/

[16] National e-Science Centre home page, http://www.nesc.ac.uk/

[17] EPCC home page, http://www.epcc.ac.uk/

[18] Sun Microsystems home page, http://www.sun.com/