University of Victoria
Faculty of Engineering
ENGR 446 Technical Report

Running ATLAS Experiment Simulations on a Grid Environment

Department of Physics
University of Victoria
Victoria, British Columbia

Lila Klektau
9908033
Computer Engineering
lklektau@engr.uvic.ca

January 4, 2004

in partial fulfillment of the requirements of the
B.Eng. Degree

3968 Oakwood Street
Victoria, British Columbia
V8N 3N1

Co-op Coordinator
Faculty of Engineering
University of Victoria
P.O. Box 1700
Victoria, B.C.
V8W 2Y2

January 4, 2004

Dear Coordinator,

Please accept the accompanying work term report entitled "Running ATLAS Experiment Simulations on a Grid Environment".

This report is the result of work completed at the University of Victoria, in the Department of Physics. As a research assistant, I was tasked with creating a resource brokering tool for submission to a grid computing environment. I was also tasked with deploying the ATLAS simulation software across a production grid. This report is the result of that work.

During the last four months, I have gained a lot of knowledge on the topic of Grid Computing technology. I have also been able to experience working in the academic sector. I am sure that the new skills I have gained will be beneficial to my future endeavors, and I have welcomed the chance to explore a new work environment.

I would like to thank Dr. Randall Sobie, Dr. Ashok Agarwal, and Daniel Vanderster for their help, insight and support throughout the last four months.

Sincerely,

Lila Klektau

Contents

List of Figures

List of Tables

Executive Summary

The ATLAS Experiment, which makes use of the Large Hadron Collider at the CERN laboratory in Switzerland, will generate one petabyte of data every year once construction is completed and experiments can be run. Processing this data demands a large amount of computer processing power, beyond the scope of many facilities. Also, software to simulate the ATLAS experiment has been developed, and running this simulation also requires a lot of processing power. One solution to address the need for processing power is Grid Computing. A production grid environment has been created by the Grid Canada group. This grid has been implemented using version 2 of the Globus Toolkit, which is an open source project. Each cluster on the grid is also running the PBS job manager and the Ganglia monitoring software.

In order to install the ATLAS simulation software on each resource in the grid environment, an installation script was written. This script makes use of Globus' tools to submit jobs to a specified remote resource, as well as Globus' Grid-FTP technology to transfer files. The installation primarily involves copying 91 rpm files, 20 input files, and some configuration scripts to the desired location, and then installing the rpm files. While the existing script does work, it could be improved upon by using MD5 checksums to verify the integrity of the transferred files. The script does not check if Atlas is already on the machine, and it may be desirable to add logic to perform this function. Currently the install takes between 2 and 3 hours to complete, depending upon network traffic and processor speeds. If a quicker install is desired, the installation should be altered to use GSI-enabled OpenSSH sessions instead of Globus' job submission tools in order to perform shell tasks, as a Globus job submission command takes some time to prepare and a GSI-enabled OpenSSH connection is almost instantaneous.

Once installed, Atlas can be run. However, ideally a user should be able to submit a job to the grid, without being aware of which resources are available and where the job will go. Unfortunately, the Globus Toolkit contains no mechanism for resource discovery and brokering, leaving it the user's responsibility to specify the resource on which a job should be executed. Three different methods were developed to add this functionality. The first acts as a wrapper for the `globus-job-submit` command. It generates a list of available resources, chooses one at random, and then submits the job. The second tool developed makes use of a MySQL database in which jobs are queued and their status is tracked. The third method is very similar to the first, but is a wrapper to the Condor-G program. Condor-G is an extended version of the Condor batch queuing system, which integrates the Globus Toolkit. Alone, Condor-G is capable of monitoring job progress, but it does not contain a resource discovery and brokering element.

Of these tools, the second is preferred. It is the only method whereby the entire status of the grid can be known, because the centralized MySQL database component is aware of every job. Utilizing this knowledge, a more intelligent resource brokering algorithm can be developed. This should be researched and a more efficient algorithm should be substituted in place of the current one, which chooses a resource at random from the list of resources available. Also, some improvements must be made to this tool. Currently, security is not handled well. In addition, the method of transferring an executable to a grid resource should be improved.

Upon running Atlas, it is apparent that the time to complete each job varies significantly with the processing power available. In the simulations performed, the time to complete each job varied from 5 to 9 hours. Memory usage is constant at around 350 MB. Developing a more intelligent resource brokering algorithm will help ensure that jobs are sent to the best available resources, making the grid environment more efficient.

Glossary

Bash Shell Bourne Again Shell. Common UNIX shell environment.
CERN European Organization for Nuclear Research.
Condor A batch queuing system.
Condor-G Globus extended version of the Condor batch queuing system.
Daemon Disk And Execution MONitor.
Delegated Proxy A grid Proxy signed by a user's already existing proxy, as opposed to being signed by a user's private key.
FTP File Transfer Protocol.
Fork A UNIX command used to create a new process.
Ganglia Load monitoring software, which publishes information online.
Globus Open source software used to create a grid environment.
GIIS Grid Index Information Service.
GRAM Globus Resource Allocation Manager.
Grid-FTP Globus enhanced version of FTP.
GRIS Grid Resource Information Service.
Grid-Proxy File An X.509 certificate signed by a user's private key, used to authenticate on a Globus grid environment.
GSI Grid Security Infrastructure.
LDAP Lightweight Directory Access Protocol.
LSF Load Sharing Facility. A batch queuing system.
MD5 Checksum A reliable way of verifying data integrity.
MDS Metacomputing Directory Service.
MyProxy Software implementing an online grid credential repository.
MySQL An open source database.
NFS Network File System.
PBS Portable Batch System. A batch queuing system.
Perl A high level programming language.
PHP Pre-processor Hypertext. Web scripting language.
Public Key Encryption A cryptographic system that uses two keys - a public key known to everyone and a private key known only to one person. Messages are encrypted using the recipient's public key, and the recipient uses their private key to decrypt the message.
RPM Redhat Package Manager.
RSL Resource Specification Language.
SSL Secure Sockets Layer.
Staged File When the file to be run on a grid environment is initially located on the submission machine and not on a grid resource, it must be "staged in" to the grid resource.
X.509 A standard defining what information should go into a public key encryption based certificate, and how that information should be formatted.

1 Introduction

The *A Toroidal LHC ApparatuS* (ATLAS) Experiment, which will utilize the Large Hadron Collider (LHC) at the CERN Laboratory in Geneva, Switzerland, is currently under construction. The objective is to investigate the fundamental nature of matter. The ATLAS experiment will be a collaboration of over 2000 physicists, representing more than 150 universities and laboratories in 34 countries. When it is complete, the ATLAS experiment will generate over one petabyte of data per year, which will then require further analysis [1].

The computational challenge presented by this experiment exceeds in scale and complexity anything that has been achieved in particle physics to date, requiring more processing power than many facilities have available. In addition, a software program that simulates the ATLAS experiment has been created. The Atlas simulation is used to create test output in order to verify that the analysis tools work correctly. It will also be used to gain an understanding of how the experimental results should look. Running this simulation to obtain data also requires a large amount of processing power. A promising solution to address the computational needs of the ATLAS project is *grid computing.*

The term "grid computing" derives its name from the electrical power grid structure, where multiple energy sources are pooled into one infrastructure, providing electricity where it is required and in the amount required. Grid computing promises to pool computer processing power and disk storage space in a similar manner, through the use of computing grids and data grids respectively. This involves linking resources together across administrative domains, and still allowing the use of a system's resources to be controlled by its owner. It also involves the use of Meta schedulers to determine which resource should execute a given job.

In order to run the Atlas software on a grid computing environment, the first step is to create a grid environment. The second step is to install the Atlas software on each grid resource where it will be run. Thirdly, a resource brokering system should be in place, which will determine where each job should be executed. The Grid Canada grouop has already completed the first step, and a production grid has been established. The Grid Canada production grid currently comprises three computing clusters, located at the University of Victoria, the University of Alberta, and the National Research Council (NRC) in Ottawa. As a result, only a brief outline of the components involved in creating a computational grid environment will be given in this report. The last two steps have only been developed in the preceding three months, and are still being improved upon. The main body of this report focuses on these last two steps, explaining what has been done, the challenges that have been faced, and alternatives and improvements which should be considered. The report concludes by successfully running a few Atlas jobs across the grid, using the system created.

2 Connecting Resources Into a Grid

2.1 Grid Software: The Globus Project

There are many software packages available that may be used to create grid computing networks. Some are proprietary, like Sun Microsystem's package, while others are open source. One of the most prominent open source projects is the Globus Toolkit, developed by the Globus Alliance [2].

The Globus Toolkit is comprised of three primary components, available as both client and server versions, and they are designed to be modular but complimentary. These components are labeled *Resource Management*, *Information Services* and *Data Management*. They all build upon the Grid Security Infrastructure (GSI) security protocol to create a common security foundation.

The GSI security protocol allows mutual authentication and single sign-on. It is an extension of public key encryption, X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol. To obtain access to Globus resources, a user must first request a certificate from a trusted certificate authority. Each grid resource must also edit their local `/etc/grid-security/grid-mapfile` file to allow the specific user access by mapping that user's identity, as specified in the certificate, to a local user. Issuing the command `grid-proxy-init` and a providing the correct pass phrase on a submission machine will sign the user's certificate with the user's private key, providing both files exist on that machine, and this results in the creation of a grid-proxy file. The grid-proxy file has a limited lifetime, and is used in all Globus transactions to authenticate the user.

The Resource Management component allows job submission to remote machines by way of the Globus Resource Allocation Manager (GRAM). GRAM is an interface between the user and the resource management tools that are in place at each resource, such as LSF, Portable Batch System(PBS) and Condor. A user with a valid grid-proxy can then issue the `globus-job-run` or `globus-job-submit` command to run a job on any resource running Globus, despite the difference in job managers. When installing the Resource Management server component, multiple local job managers can be used with Globus, providing the corresponding Globus interface is installed and enabled for each. The default job manager is `fork`, although this default can be changed using a provided script. To specify a non-default job manager, the host name specified in the `globus-job submit` or `globus-job-run` command should be proceeded with `/jobmanager-<jobmanager>`. For example, to submit to the PBS job manager on `mercury.uvic.ca`, the command would be `globus-job-submit mercury.uvic.ca/jobmanager-pbs` followed by arguments to specify the job. The Resource Management component also implements a Resource Specification Language (RSL), which allows users to define additional parameters when submitting a job to Globus. These parameters are then interpreted by the GRAM and passed off to the resource manager at the site.

The Information Services implements a Grid Resource Information Service (GRIS), which allows each resource to catalog information about itself in using the Lightweight Directory Access Protocol (LDAP). This information can also be published to a central Grid Index Information Service (GIIS) server. The GIIS can then be queried to determine which resources are a part of the grid, and also to gather specific information about each resource. The entire information system formed from GRIS components and the GIIS is termed the Globus Metacomputing Directory Service (MDS).

The Data Management component provides the Grid-FTP functionality. Grid-FTP is based upon standard FTP, with the following additional features:

- GSI security on control and data channels
- Multiple data channels for parallel transfers
- Partial file transfers

- Third-party (direct server-to-server) transfers
- Authenticated data channels
- Reusable data channels
- Command pipelining

For a more detailed overview of these components, refer to *http://www.globus.org/gt2.4/admin/guide-overview.html* on the the Globus Alliance web site.

2.2 Grid Canada Installation Requirements

Each grid resource must have version 2 of the server component for all three Globus Toolkit packages installed, and it must publish information to the Grid Canada GIIS server. It must also have the PBS job monitoring system installed and enabled through Globus, so that when a Globus job is given the argument `<hostname>/jobmanager-pbs` the job is run using the PBS batch system [3]. An additional component necessary for monitoring purposes is the Ganglia software [4]. This is a package which publishes graphs and statistics online, in real time, about the status of a resource.

For simple submissions to the grid, only version 2 of the Resource Management client package of the Globus Toolkit needs to be installed on the submission machine. However, the user must also have a certificate issued by a trusted certificate authority, and that certificate must be signed with the user's private key in order to create a grid-proxy certificate. Furthermore, the user must have authorization to access each required grid resource. In addition, the Data Management package is useful, as it contains the tools to perform Grid-FTP transfers.

3 Installing Atlas on Grid Resources

3.1 Requirements for Installation

3.1.1 Components of Installation

An Atlas installation is comprised of multiple steps. The main installation itself is performed by 91 rpm files, which need to be transferred to the desired resource. A `MYSQL` directory from a previous installation should also be copied to the resource. This directory contains the MySQL client, daemon, and the database necessary for running Atlas. Additionally, a `noisedb.tgz` file needs to be copied to the resource. There are also 4 scripts, responsible for configuration, that need to be copied from an existing installation and manually customized for each resource. These scripts are named `recon.gen.v5.with602rpmkit`, `recon.gen.v5.with602rpmkit_NG_db`, `eg7.602.job` and `eg7.602.job.db`. Finally, depending on how Atlas will be run, the input files may need to be copied to the resource. There are 20 files in total, with each file being approximately 1.5GB in size. The Atlas directory structure prior to installing the rpm files is as shown below, where `<atlas-dir>` is the fully qualified base directory of the installation:

```
<atlas-dir>/
  6.0.2-1/
    installed/
    MYSQL/
      contents of MYSQL directory from previous install
    project/
      dc1/
        lumi02/
          002000/
            data/
              input files
    rpmdb/
    eg7.602.job
    eg7.602.job.db
    noisedb.tgz
    recon.gen.v5.with602rpmkit
    recon.gen.v5.with602rpmkit_NG_db
  atlas-kit/
    6.0.2-1/
      release/
        33 rpm files
      share/
        58 rpm files
```

Once this directory structure is in place, the rpm files need to be installed using the options `--prefix <atlas-dir>/6.0.2-1/installed` and `--dbpath <atlas-dir>/6.0.2-1/rpmdb`.

After the rpm files have been installed, the `<atlas-dir>/6.0.2-1/installed/etc` directory will be created. In this directory is a script named `relocate` that needs to be run and given the `prefix` path as an argument: `./relocate <atlas-dir>/6.0.2-1/installed`. This will configure the contents of all the Atlas components installed by the rpm files so that they specify the correct paths.

3.1.2 Method of Installation

The grid is designed in such a way that users do not require direct login access on a resource in order to execute jobs there, and therefore passwords are often not created for grid accounts. However, Atlas must be installed across all resources that will be used in running Atlas jobs. It is possible to open an interactive prompt to a grid resource using GSI-enabled OpenSSH. The GSI-enabled OpenSSH client, when coupled with a GSI-enabled OpenSSH server, allows a user to authenticate to a remote machine using a grid-proxy, as opposed to a password. However, this is currently not enabled on the Grid Canada production grid, and therefore it was necessary to devise a means of installing Atlas by using the Globus tools, and not through an interactive prompt. A bash shell script was used used to automate this process.

Installing through Globus tools is easily done with the `globus-url-copy` and `globus-job-run` commands. Using `globus-url-copy`, files can be transferred from one grid resource to another, and neither of these resources needs to be a local machine. For example, the command

```
globus-url-copy gsiftp://mercury.uvic.ca/home1x/gcprod/gcprod01/file1 \
gsiftp://imp.phys.uvic.ca/homes/lmk/file1
```

will transfer `file1` from `mercury.uvic.ca` to `imp.phys.uvic.ca`, providing each of these machines has the grid-ftp server component installed and the machine where the command is executed has the grid-ftp client component installed.

Performing other installation tasks such as installing rpm files and running the relocate script can be done in one of two ways. The first method, which is the method that was employed, makes use of the `globus-job-run` command. The installation bash script creates a secondary script to run the desired commands, and then executes that script on the desired resource with the following command:

```
globus-job-run <resource_name> -s <script>
```

The `-s` option specifies that the script is to be staged, i.e. the script exists on the machine executing the `globus-job-run` command, and must first be copied over to the specified resource. This is contrasted with the `-l` option, which specified that the script or executable already exists on the specified resource.

The alternate method of performing installation tasks is to install and use GSI-enabled OpenSSH. Then a terminal session to the remote resource can be opened when commands need to be executed on the machine where the Atlas software is being installed, instead of using the Globus job submission tools. This method is a bit more complicated, but is used by employees at CERN in order to automate Atlas installations [6], although at CERN the installation is coded in Java and not scripted. The advantage is that the connection is almost instantaneous, whereas the `globus-job-run` command takes some computational time to prepare. As a result, using GSI-enabled OpenSSH would shorten the time required for installation.

Typically, the user performing the Atlas installation will only have write access to their own home directory. As a result, the chosen location for the Atlas installation is in the home directory structure. In order to standardize the installation location, the Atlas installations will all be performed by the same user, and placed into a directory named `atlas` within that user directory. This will enable Atlas on any resource to be referenced by ~`<installing-user>/atlas`

3.2 Database Component

Atlas relies upon data stored in a MySQL database in order to run simulations. That database is copied over in the installation, but the question of where to run the database daemon arises.

There are three possible implementations to consider in the case where the Globus resources are clusters. The favoured implementation is to start the MySQL daemon on the head node of the cluster and let the jobs, running on the internal nodes, contact the central database. The issue with this implementation is how to start the daemon on the head node. This would probably have to be resolved by a direct interactive connection to the head node, as when a job is submitted through Globus it is typically executed on the internal nodes. An alternative is to have the node executing the Atlas job start a daemon locally before processing the job, possibly terminating the daemon after the job has run. The only advantage to this approach is that it ensures a way to start the daemon if it is not running. There are many drawbacks. For one, jobs cannot run on nodes simultaneously, as one will be killing the database while another is still using it. Also, since the home directories in these clusters are often on NFS partitions, and the database is in the Atlas directory structure, which is itself within a home directory, there will be multiple daemons running for the same database, which could cause locking problems. The final possibility is to have a limited amount of servers provide the MySQL data component of Atlas. These servers could be separate from the grid resources, and any Atlas job would be directed to contact one of these remote servers for data.

The first two database models are currently implemented. The `recon.gen.v5.with602rpmkit` and `eg7.602.job` scripts correspond to database model where the daemon is already running on the head node of the cluster. Alternatively, the scripts named `recon.gen.v5.with602rpmkit_NG_db` and `eg7.602.job.db` will start a MySQL daemon for each Atlas process.

3.3 Challenges Encountered

In creating this installation script, many problems were encountered that needed to be addressed. The most apparent of these was that directory structures differ on different machines. In order to standardize the installation location, it was decided that the best solution would be to always install Atlas into an `atlas` directory under the same user account, as mentioned previously. In this way all Atlas installations can be referenced by ∼`<installing-user>/atlas`, no matter where the home accounts may be located. In the case of the Grid Canada Production installation, the `gcprod01` user account will be used. It should be noted that in order for this approach to work, all users must have read access to the installing user's home directory.

Another issue that needed to be resolved in an automated install was that there were some configuration files that need their contents modified in order to be resource specific. As mentioned previously, these files are: `recon.gen.v5.with602rpmkit`, `recon.gen.v5.with602rpmkit_NG_db`, `eg7.602.job` and `eg7.602.job.` This problem was solved by having the installation script create a Perl script that would be able to take as parameters a regular expression and file names, and would then evaluate the regular expression upon each filename. In this case the regular expression is a search and replace string. Typically search and replace strings are of the form `s/oldvalue/newvalue/g` where `g` specifies that all occurrences of `oldvalue` should be replaced. However, for the Atlas install, the old and new values are directories, which themselves contain the `/` character. The simplest way around this problem was to note that find and replace syntax is not dependent upon the `/` character to delimit the arguments. In fact, any character can be used as long as it is used consistently. Therefore, the find and replace string used is of the form `s@oldvalue@newvalue@g`.

An additional issue was encountered with libraries while installing Atlas. In order to run, Atlas requires the `libXm.so.3` library, and unless the source code is altered it will not source the libraries in `/usr/X11R6/lib`, which is where that library is typically located. Copying `libXm.so.3` to

`<atlas-dir>/6.0.2-1/installed/software/dist/6.0.2/InstallArea/i686-rh73-gcc295/lib`, where it will then be sourced, seems to be the easiest solution.

Unfortunately, there remains at least one task in the install that cannot be automated and must be performed by a user with root access. A link must be made in the `/usr/local` directory to the file `<atlas-dir>/6.0.2-1/installed/i386_redhat73/usr.local/gcc-alt-2.95.2`. Also, Atlas is meant to be run on a Redhat platform, and if this is not the case, the file `/etc/redhat-release` needs to be created on the resource in order to trick Atlas.

In the future, it may be desirable to add support for verifying that files were copied successfully. This could be done by comparing the MD5 checksums for the source and destination copies of each file.

3.4 Installation Performance

The final installation file is included in Appendix 7. Below is a summary of installation times on different resources.

Average times to install mercury.sao.nrc.ca:
Tarring rpm files and MYSQL directory: 1.3 minutes
Transferring tarred rpm files: 2 minutes
Transferring tarred MYSQL directory: 5 seconds
Copying input files: 5.6 minutes x 20 files $\approx$ 2 hours
Copying Configuration Files: 20 seconds
Untarring MYSQL files: 14 seconds
Untarring rpm files: 6 minutes
Installing rpm files: 28.5 minutes
Running relocate script: 7.5 minutes
Customizing Scripts: 16 seconds
Copying libraries: 16 seconds
Total $\approx$ 2.5 hours

Average times to install mercury.uvic.ca:
Tarring rpm files and MYSQL directory: 1 minute
Transferring tarred rpm files: 1 minute
Transferring tarred MYSQL directory: 3 seconds
Copying input files: 2.6 minutes x 20 files $\approx$ 1 hour
Copying Configuration Files: 19 seconds
Untarring MYSQL files: 12 seconds
Untarring rpm files: 1 minute
Installing rpm files: 24.5 minutes
Running relocate script: 22.5 minutes
Customizing Scripts: 16 seconds
Copying libraries: 4 seconds
Total $\approx$ 2 hours

4 Submitting Jobs to the Grid with a Resource Broker

4.1 Requirements of a Resource Broker

The Globus tools are capable of submitting a job to a specified remote resource, but there is no mechanism for resource discovery and brokering. Ideally, the grid is transparent to the user, with jobs being submitted to available resources without the user's knowledge of which resources are being used. A resource broker is needed for effective submission of Atlas jobs, so that when a user submits multiple jobs to the grid, they will be automatically executed on the best grid resources available.

At the University of Victoria, research is being done as to the most effective algorithm for grid job submission. In order to facilitate this research, any resource brokering tool should be designed in such a way that different algorithms can be easily substituted for each other. In addition, a resource brokering tool that has full knowledge of jobs currently running on the grid would be beneficial in order to test more intelligent algorithms. For example, it should be possible to resubmit a job if a faster resource becomes available, and this requires knowledge of which jobs are currently running and what the submission parameters of each job are.

Additional factors to be taken into consideration when designing a resource brokering tool are security, job monitoring capabilities and reliability.

4.2 Solution 1: Submission Script

4.2.1 Motivation

A very simple tool was desired for the first attempt at a solution. This tool only needed to be capable of resource discovery, brokering and submission. No logging or tracking capabilities were required for this version, as the main focus was to get something working, and then add more complicated mechanisms later.

4.2.2 Description of the Solution

The tool created is a simple Perl script named `gcsub.pl`. It is essentially a wrapper for `globus-job-submit` that adds resource discovery and brokering capabilities. For resource discovery, it makes use of the GIIS server. Resource discovery is accomplished by querying the GIIS server for the list of resources on the grid that meet the criteria specified by the user, and then filtering them to determine which resources the user is authorized to use. There is currently no way to filter the list of resources according to which are busy and which are idle. Brokering is then done by applying a chosen algorithm to the list of resources. At present, the algorithm used picks a resource from the list at random. The `globus-job-submit` command is then executed using this resource, and the Globus job-id is returned to the user as standard output. When running `globus-job-submit`, `jobmanager-pbs` is specified after the host name, and additional RSL argument is given to increase the time PBS allows the job to run: `-x '(maxtime=1439)'`. If this is not done PBS terminates the job after the default time allowed, which in some cases can be as short as one minute.

The `gcsub.pl` script accepts various arguments as specified below:

```
Usage ./gcsub.pl [-h] [-v] [-d] [requirements...] (<-s filename> |
        <-l filename>) [-a arguments] [-p parameter sweep]

where
```

```
    -h
        Show this help page.
    -v
        Verbose mode. Print some status messages to STDERR.
    -d
        Dry run. Build the job, find available resources, but do not submit
        the jobs.
    -l filename
        Execute this local executable. (eg. '/bin/uname')
    -s filename
        Stage and run this executable. (eg. '/bin/uname')
    -a arguments
        Supply these arguments (eg '-a'). Argument %1 will be filled by the
        parametric sweep.
    -p sweep
        Run a parametric sweep. 'sweep' should be of the form x-y/j, x is
        the lower bound of %1, y is the upper bound, and j is the increment.

                Example: '-p 1000-4999/100' will create 49 instances of the
                job with the argument %1 varying from 1000, 1100, 1200, ...,
                4800, 4900.

Requirements
    -O OS
        Only submit to this OS (eg. Linux).
    -C CPU MHz
        Only submit to resources with CPU MHz greater than C.
    -P platform
            Only submit to resources running this platform (eg. i686).
    -M memory MB
        Only submit to resources with memory MB greater than M.
```

The script in it's entirety is given in Appendix 7.

4.2.3 Demonstration of Solution

The first step in using `gcsub.pl` to submit jobs to the grid is to obtain a valid proxy

```
[lmk@imp lmk]$ grid-proxy-init
Your identity: /C=CA/O=Grid/OU=phys.uvic.ca/CN=Lila Klektau
Enter GRID pass phrase for this identity:
Creating proxy ............................................ Done
Your proxy is valid until: Fri Nov 28 20:28:53 2003
```

Now a job can be submitted. For this example, the `echo` command is used to create a parameter sweep submission. The command that will be used is `./gcsub.pl -v -l /bin/echo -a "I am submission %1" -p 1-20/5`. The `-v` option specifies verbose output, `-l` specifies that the file exists locally, text proceeding `-a` is the argument to the executable and the text proceeding `-p` defines the sweep. In this case, `1-20/5` specifies that numbers 1 through 20, in steps of 5 (i.e.1,6,11,16),

will be used to fill the %1 in the argument string. Below is the actual output from running this command.

```
[lmk@imp atlas]$ ./gcsub.pl -v -l /bin/echo -a "I am submission %1" -p 1-20/5
Running parameter sweep...
Jobs to be run:
        /bin/echo I am submission 1
        /bin/echo I am submission 6
        /bin/echo I am submission 11
        /bin/echo I am submission 16
Searching giis.gridcanada.ca...done.
All hosts:
        1. mercury.sao.nrc.ca (have auth) (req met)
        2. mercury.uvic.ca (have auth) (req met)
        3. thuner-gw.phys.ualberta.ca (have auth) (req met)
Available hosts:
        1. mercury.sao.nrc.ca
        2. mercury.uvic.ca
        3. thuner-gw.phys.ualberta.ca
Selected mercury.sao.nrc.ca for /bin/echo I am submission 1
globus-job-submit mercury.sao.nrc.ca/jobmanager-pbs -x '(maxtime=1439)' \
        -l /bin/echo I am submission 1
https://mercury.sao.nrc.ca:20045/4670/1070903641/
Selected mercury.uvic.ca for /bin/echo I am submission 6
globus-job-submit mercury.uvic.ca/jobmanager-pbs -x '(maxtime=1439)' \
        -l /bin/echo I am submission 6
https://mercury.uvic.ca:40001/3522/1070903643/
Selected mercury.uvic.ca for /bin/echo I am submission 11
globus-job-submit mercury.uvic.ca/jobmanager-pbs -x '(maxtime=1439)' \
        -l /bin/echo I am submission 11
https://mercury.uvic.ca:40004/3549/1070903645/
Selected thuner-gw.phys.ualberta.ca for /bin/echo I am submission 16
globus-job-submit thuner-gw.phys.ualberta.ca/jobmanager-pbs -x '(maxtime=1439)' \
        -l /bin/echo I am submission 16
https://thuner-gw.phys.ualberta.ca:51248/23059/1070903649/
```

Once the jobs have been submitted, the user can manually retrieve the output of any job by using the Globus job-id given in the output as the argument to the `globus-job-get-output` command.

4.2.4 Comparison with Requirements

Due to the lack of job tracking capabilities, this script fails to meet all the requirements desired for a resource brokering tool. However, this first solution was never intended to fulfill those requirements, instead being more of a prototype to understand how the submission aspect would work. In this respect, the script is successful. It does not compromise security, as its only interaction with the grid technology is through Globus tools and the GIIS. Since there is no job tracking, jobs cannot be monitored. The script is reliable though, due to its simplicity. There is no centralized component, so there is no single point of failure, other than perhaps the GIIS server which is an

integral part of any Globus installation. A drawback from having such a simple submission tool is that if there are no resources found which meet the criteria, it is the user's responsibility to resubmit the job at a later date.

4.3 Solution 2: Submission Script and MySQL Queue

4.3.1 Motivation

Having created a simple submission script, the next step was to elaborate and add job tracking capabilities. This solution was to be capable of logging jobs to be submitted and tracking the progress of already submitted jobs. It was also to have a monitoring interface so users see the status of their jobs.

4.3.2 Overview of the System

The system created is comprised of four components:

1. `gcsub.pl`: a Perl script to break jobs into processes and submit them to the queue. This is a modification of the `gcsub.pl` script created previously.
2. `gcmanager.pl`: a Perl script to submit the queued processes to the grid and track their status
3. a MySQL database to store queued process items and track job and process status.
4. a PHP script to display the status of processes on a web page

In this solution a distinction is made between jobs and processes. A user submits a job to `gcsub.pl`, which then creates one or more processes from it, depending on whether the job can be broken up into smaller tasks (e.g. parametric sweeps).

This system is designed so that each user can have a copy of the `gcsub.pl` script, which is what allows users to submit jobs to the grid. A centralized database is used to keep track of jobs that have been submitted. The `gcmanager.pl` script is intended to be run in only one place, and two instances can not be running concurrently. It is essentially the job manager for the grid. It is intended that `gcmanager.pl` will run as a cronjob every minute.

4.3.3 Submission Component

The `gcsub.pl` script takes the same arguments and parameters as its predecessor, described in Section 4.2.2. The difference between the two `gcsub.pl` versions is that this second version does not submit jobs to the grid. Instead it will analyze the job submitted and create one or more processes from it, depending on whether the parameter sweep option is defined. If a dry run has not been specified, it will then check to ensure the user has a valid proxy. Once this has been ascertained, an entry for the job will be created in the MySQL database. This entry will contain all of the options specified by the user, the identity of the user as obtained from their proxy and a copy of their proxy file. An entry will also be created in the database for each process that comprises the job. These process entries will contain information specific to each process, and have fields to track their status. It is this process table that is used as a queue for the grid.

It is necessary that the grid proxy be stored in the database in order for the process to be run on behalf of the user. In addition, the status of a process can only be checked by the user submitting it. Once all processes in a job have completed, the proxy is removed from the database. If the job is to be staged, it is also necessary for the staged file to be stored on the machine submitting

the job. The current solution is for the staged file to be stored in the database. When a job has completed, the file is removed from the database, as it is no longer necessary.

For more details on the database structure, refer to Section 4.3.5. The complete `gcsub.pl` script is included in Appendix 7.

4.3.4 Scheduling Component

The Perl script `gcmanager.pl` is responsible for submitting queued processes to the Grid Canada resources and tracking their status. For all interactions with the grid, `gcmanager.pl` must use the proxy of the user who queued the corresponding job. This is done by temporarily creating a proxy file in the /tmp directory using the content stored in the database, executing `chmod 600` on the file, and exporting the variable `X509_USER_PROXY` to point to the file. Once `gcmanager.pl` has finished submitting or checking the status of a process, the proxy file is removed.

For each process that has not yet been submitted to the grid, `gcmanager.pl` first determines which resource will receive the process. This is done in the same manner as in the version of `gcsub.pl` described in section 4.2.2: The GIIS is queried for resources registered with the grid, the proxy of the user who submitted the job is used to ascertain which resources are authorized for use, the resource list is filtered based on the criteria specified as input to `gcsub.pl`, and then a resource is randomly chosen from the final list. If the job is to be staged, `gcmanager.pl` must retrieve the file from the databases and store it in the `/tmp` directory for submission to the chosen grid resource. Once the process has been started, `gcmanager.pl` records the Globus job-id and the resource to which it was submitted in the database.

For each process in the queue that has been submitted but not completed, `gcmanager.pl` checks the status using the `globus-get-job-status` command and the corresponding Globus job-id. It then updates the process status in the database accordingly.

After checking the status of all processes, `gcmanager.pl` compares the list of active jobs in the database with the list of processes. When it finds a job whose processes have all completed, it marks the job as finished and removes the user's proxy information, as well as the staged file if it exists, from the job record.

The complete `gcmanager.pl` script is included in Appendix 7

4.3.5 Queue Component

The queue component of the system was implemented with MySQL, as it is available as open source and easily queried through Perl and PHP. The structure of the database used is shown in Tables 1 through 4.

4.3.6 Monitoring Component

A simple PHP script was created in order to view the status of queued processes. It queries the MySQL database component and displays information about all uncompleted processes, as well as information about processes which have completed in the last hour, in a table format. The cells are colour coded according to the status of each process. This information is currently published to *http://grid.phys.uvic.ca/gcprod/status/*

4.3.7 Demonstration of Solution

Below is a transcript from a sample operation. The verbose option has been specified in `gcsub.pl`, and `gcmanager.pl` is running with debugging turned on in order to create a more

Name	*Datatype*	*Description*
job_id	int	Unique identifier
submit_datetime	datetime	Time job was submitted to database by gcsub.pl
finish_datetime	datetime	Time all processes in job have finished
staged	boolean	True if job should be staged
user_identity	varchar	Globus Identity of user that submitted the job
alive	boolean	True if the job still has processes that are running
req_os	varchar	OS requirement for the job
req_cpu	int	CPU requirement for the job, in MHz
req_memory	int	Memory requirement for the job, in MB
req_platform	varchar	Platform requirement for the job
gcsub_command	varchar	Command used to run gcsub.pl when submitting job
filename	varchar	Name of file to be submitted
grid_proxy	blob	Proxy of user that submitted the job
staged_file	largeblob	Content of the file to be run, if job is staged

Table 1: MySQL *job* table

Name	*Datatype*	*Description*
process_id	int	Unique identifier
job_id	int	ID of job this process belongs to
resource_id	int	ID of resource this process has been submitted to
start_datetime	datetime	Time process was submitted to resource
end_datetime	datetime	Time process finished executing
command	varchar	Globus command used to submit the process
globus_id	varchar	Globus ID of process
status_id	int	ID of status pertaining to this process
submit_datetime	datetime	Time process was entered in database/queue
arguments	varchar	Arguments to be supplied when submitting the process

Table 2: MySQL *process* table

Name	*Datatype*	*Description*
ps_id	int	Unique identifier
status	varchar	Possible status of process

Table 3: MySQL *process_status* table

Name	*Datatype*	*Description*
resource_id	int	Unique identifier
hostname	varchar	Hostname of resource
institution	varchar	Institution where resource is located
contact_name	varchar	Name of contact for resource
contact_email	varchar	Email address of contact

Table 4: MySQL *resource* table

User	Filename	Arguments	Staged	Time Submitted
Lila Klektau	/usr/bin/whoami		yes	2003-12-09 12:09:37
Ashok Agarwal	/usr/bin/whoami		yes	2003-12-09 12:09:22

Status	Time Started	Time Finished	Resource	Globus Job ID
queued				
queued				

Figure 1: Output of PHP monitoring script, jobs in queued status. Table cells are wrapped to the next line for formatting purposes within this document only.

.

descriptive example.

First, jobs are submitted from two separate user accounts, using `gcsub.pl`:

```
[lmk@imp gcsub-prod]$ ./gcsub.pl -v -s /usr/bin/whoami
Processes to be run:
        /usr/bin/whoami
[lmk@imp gcsub-prod]$

[agarwal@imp gcsub-prod]$ ./gcsub.pl -v -l /usr/bin/whoami
Processes to be run:
        /usr/bin/whoami
[agarwal@imp gcsub-prod]$
```

The `gcsub.pl` script will have created one process from each of these jobs for submission to the grid. After the processes have been queued in the MySQL database, they can be seen using the online PHP script, as shown in Figure 1. The `gcmanager.pl` script is then run from any account in order to submit the processes to the chosen Grid Canada resource:

```
globus@imp gcsub-prod]$ ./gcmanager.pl
Searching giis.gridcanada.ca...done.
All hosts:
        1. mercury.uvic.ca (have auth) (req met)
        2. mercury.sao.nrc.ca (have auth) (req met)
        3. thuner-gw.phys.ualberta.ca (have auth) (req met)
Available hosts:
        1. mercury.uvic.ca
        2. mercury.sao.nrc.ca
        3. thuner-gw.phys.ualberta.ca
/homes/globus/globus-2.4.3/bin/globus-job-submit mercury.uvic.ca/jobmanager-pbs \
        -x '(maxtime=1439)' -s /tmp/processid46.file
Searching giis.gridcanada.ca...done.
All hosts:
        1. mercury.uvic.ca (have auth) (req met)
```

User	Filename	Arguments	Staged	Time Submitted	Status	Time Started
Lila Klektau	/usr/bin/whoami		yes	2003-12-09 12:09:37	pending	2003-12-09 12:31:11
Ashok Agarwal	/usr/bin/whoami		yes	2003-12-09 12:09:22	pending	2003-12-09 12:30:20

Time Finished	Resource	Globus Job ID
	mercury.uvic.ca	https://mercury.uvic.ca:40001/5535/1071001900/
	mercury.uvic.ca	https://mercury.uvic.ca:40006/3935/1071001852/

Figure 2: Output of PHP monitoring script, jobs in pending status. Table cells are wrapped to the next line for formatting purposes within this document only.

```
        2. mercury.sao.nrc.ca (have auth) (req met)
        3. thuner-gw.phys.ualberta.ca (have auth) (req met)
Available hosts:
        1. mercury.uvic.ca
        2. mercury.sao.nrc.ca
        3. thuner-gw.phys.ualberta.ca
/homes/globus/globus-2.4.3/bin/globus-job-submit mercury.uvic.ca/jobmanager-pbs \
        -x '(maxtime=1439)' -s /tmp/processid47.file
[globus@imp gcsub-prod]$
```

Note that a temp file is being submitted to Globus, and not the `/usr/bin/whoami` file. This is because the `whoami` executable was specified to exist on the submission machine, and its contents have been copied to the MySQL database. The `gcmanager.pl` script has used that content to create a temporary file for submission. This file's contents are equivalent to those of the `whoami` executable, and the file will be removed after it has been submitted.

The processes are now running on the specified resources. This can be verified by referring to the output of the PHP monitoring script, shown in Figure 2. Running gcmanager.pl again will check the status of the processes.

```
[globus@imp gcsub-prod]$ ./gcmanager.pl
/homes/globus/globus-2.4.3/bin/globus-job-status \
        https://mercury.uvic.ca:40006/3935/1071001852/
status is DONE
/homes/globus/globus-2.4.3/bin/globus-job-status \
        https://mercury.uvic.ca:40001/5535/1071001900/
status is DONE
Job 25 is Done
Job 24 is Done
[globus@imp gcsub-prod]$
```

User	Filename	Arguments	Staged	Time Submitted	Status	Time Started
Lila Klektau	/usr/bin/whoami		yes	2003-12-09 12:09:37	done	2003-12-09 12:31:11
Ashok Agarwal	/usr/bin/whoami		yes	2003-12-09 12:09:22	done	2003-12-09 12:30:20

Time Finished	Resource	Globus Job ID
2003-12-09 12:36:40	mercury.uvic.ca	https://mercury.uvic.ca:40001/5535/1071001900/
2003-12-09 12:36:39	mercury.uvic.ca	https://mercury.uvic.ca:40006/3935/1071001852/

Figure 3: Output of PHP monitoring script, jobs in done status. Table cells are wrapped to the next line for formatting purposes within this document only.

The updated status can be seen with the PHP monitoring script, shown in Figure 3. Using the Globus job-id given both on the web page and as the output of `gcmanager.pl`, the output of the jobs can be obtained.

```
[agarwal@imp gcsub]$ globus-job-get-output \
        https://mercury.uvic.ca:40006/3935/1071001852/
gcprod01
[agarwal@imp gcsub]$

{lmk@imp gcsub]$ globus-job-get-output \
        https://mercury.uvic.ca:40001/5535/1071001900/
gcprod05
{lmk@imp gcsub]$
```

From this example, it can be seen that by using the proxy for a given user, `gcmanager.pl` can submit jobs on their behalf, as both jobs returned the correct user names. Recall Globus security maps a user to a username on the remote system, which is why the username returned is not the same as the local username.

4.3.8 Challenges Encountered

There are two major challenges that needed to be addressed in creating this system. The first is concerning the user proxies, and the second is concerning staged file inputs. These challenges have been addressed as described in the preceding sections, however there are other ways of solving these problems that may be preferable.

Proxy Issues

In order to run a job on a user's behalf, it is necessary to have a copy of the user's proxy certificate. Currently, this is accomplished by copying the contents of the proxy file into a database

upon job submission, and then recreating that file into the `/tmp` directory when it is needed. There has been discussion as to whether this procedure poses a security risk.

This discussion is rooted in the fact that Globus already has a secure means of taking a copy of a proxy through a process called *delegation*. Every time a Globus job is run, a delegate proxy is created from the user's original proxy and kept on the resource where the job is being executed in order for the Globus tools to function correctly. While there are no exposed commands provided by the Globus toolkit that create a delegated proxy, there are tools available that can be used to obtain one, specifically MyProxy and GSI-enabled OpenSSH.

MyProxy is the favoured approach. It allows users to issue a `myproxy-init` command, much like the `grid-proxy-init` command, which creates a delegate proxy from the original and stores it on a MyProxy server, secured by a user provided password. This proxy can then be retrieved by the user from any machine and used to submit jobs through Globus. The advantage to this is that a user's private key file, which is used in creating the original proxy, only needs to be present on one machine, and not on every machine from which the user will be submitting jobs. This aids in security, as some machines may not be secure enough to store a private key file.

Using MyProxy, it should be possible for the user to create a delegated proxy for themselves on the MyProxy server, and then provide the `gcmanager.pl` script access to that proxy. In this way the proxy is transferred securely from one machine to the other.

As an alternative to MyProxy, it might be possible to open a GSI-enabled OpenSSH session with the machine running `gcmanager.pl`. In the process of creating this SSH session, a delegate proxy is automatically created from the user's proxy and placed on the other machine. This delegated proxy could then be copied to a special location, the permissions would need to be changed so that the user running `gcmanager.pl` owned the file, and it could be used to submit jobs on the user's behalf. However, this approach is a little less straightforward, and is not a preferred method.

The question arises as to what the inherent difference is between the delegated proxy and the original proxy. It has been thought that the user's private key is actually contained within the original proxy, but not within the delegate [5]. In this case, the method employed in this system is unacceptable, and needs to be altered before put into production use. However, other sources state that his is not true, and using the original proxy is perfectly secure [6]. To attest to this fact, some web interfaces to grids, termed *portals*, provide functionality for users to upload their original proxy file to the portal server in order for jobs to be submitted by the user from a remote machine [7].

If this is indeed the case, and using the original proxy is secure, then there is no security risk in the model adopted for the system created. A precaution that could be added would be to send the proxy over the network in an encrypted form. It should be noted that the proxy itself has a limited lifetime, so this adds to its security.

A further issue with the proxy is that there is no mechanism in place to renew it, should it expire before the job completes. A script should be made that can be run by a user in order to update their proxy in the database, or wherever it has been decided the proxy will be stored. If it is decided to keep the proxy in the database, it may be more logical to change the database structure to encompass a `user` table. One proxy could then be stored for each user, as opposed to for each job.

Staged File Issues

The other main issue that needed to be solved in this system was how to obtain a copy of a staged file for submission, as the file is initially present only on the user's machine and not on the machine running `gcmanager.pl` and submitting the jobs.

A quick solution to this problem was to copy the contents of the staged file into the database, and then recreate the file when the process was submitted. Unfortunately, this approach will only work for small files. A better solution would be to make use of the Grid-FTP commands and either transfer the file to the machine running `gcmanager.pl`, logging the location of the file in the database and then submitting it as a staged file, or to store a link to the file in the database, have the `gcmanager.pl` script perform a Grid-FTP transfer directly from the user's machine to the chosen resource, and then specify in the `globus-job-submit` command that the file exists locally.

The first approach would be the easiest. However, the second is more appealing because it only performs one transfer of the executable instead of two. The drawback to the second approach is that it would probably be necessary for every submission machine to be running the grid-ftp server tools, and not just the client tools.

4.3.9 Comparison with Requirements

With the exception of the security concerns over the way proxies are handled, this system meets the given requirements. It is coded modularly to facilitate the use of different resource brokering algorithms. The database component introduces a queue to the grid, as well as allowing jobs status to be tracked. In this manner, the system has full knowledge of jobs running over the grid. Jobs can also be monitored using the PHP script that has been published online.

The primary concern that may arise, once security issues have been properly addressed, is reliability. Having a centralized component introduces a single point of failure. However, in order to try many complicated algorithms, this centralized structure is necessary. In the future, depending on the nature of the algorithm chosen, it may be possible to adopt a more decentralized model.

4.4 Solution 3: Submission Script with Condor-G

4.4.1 Motivation

An alternative that should be considered when creating a resource brokering tool is the Condor-G package. This software solution is an extension of the Condor batch queuing system, and implements the Globus Toolkit. Through the use of condor submission files, jobs can be run on grid resources, queued if a grid resource is unavailable, tracked and monitored.

4.4.2 Description of Solution

In order to use Condor-G as a submission mechanism, the Condor software must be installed on each submission machine. Once it is in place, all of the commands associated with Condor are available, such as `condor_submit` to submit a job and `condor_q` to see the jobs currently queued and running. As far as the user is concerned, the only difference between Condor and Condor-G is that if the Condor-G extension is in place, a line in the submission file can specify that this job should be run through Globus, and an additional line will specify the resource.

A typical Condor-G submission file is as follows:

```
executable = /bin/uname
arguments = -a
transfer_executable = false
globusscheduler = mercury.uvic.ca/jobmanager-pbs
universe = globus
output = outputs/condor.uvic.out
```

```
log = outputs/condor.uvic.log
notification = never
queue
```

More information about condor submission file arguments can be obtained from the Condor Project Homepage [8].

It should be noted that in the submission file, it is necessary to specify the desired grid resource. However, in the requirements set out for a resource brokering tool, the user should not have to be aware of which resources are available. In order to meet these requirements, an additional component is needed. This was solved by modifying the original `gcsub.pl` script from section 4.2.2 to create a condor submission file and then submit that file to Condor, as opposed to submitting the job to Globus. The modified `gcsub.pl` script is given in Appendix 7.

4.4.3 Demonstration of Solution

the first step in submitting a job using this method is to run the `gcsub.pl` script:

```
[lmk@imp condorsub]$ ./gcsub.pl -v -s /bin/uname -a -a
Jobs to be run:
        /bin/uname -a
Searching giis.gridcanada.ca...done.
All hosts:
        1. mercury.uvic.ca (have auth) (req met)
        2. mercury.sao.nrc.ca (have auth) (req met)
        3. thuner-gw.phys.ualberta.ca (have auth) (req met)
Available hosts:
        1. mercury.uvic.ca
        2. mercury.sao.nrc.ca
        3. thuner-gw.phys.ualberta.ca
Selected mercury.sao.nrc.ca for /bin/uname -a
condor_submit lmk.job1.08-Dec-2003-09.28.31
```

Note that the `condor_submit` command took a filename argument. This file was created by the `gcsub.pl` script, and is a condor submission file. The contents are as follows:

```
executable = /bin/uname
arguments = -a
transfer_executable = true
globusscheduler = mercury.sao.nrc.ca/jobmanager-pbs
globusrsl = (maxtime=1439)
universe = globus
output = lmk.job1.08-Dec-2003-09.28.31.out
log = lmk.job1.08-Dec-2003-09.28.31.log
notification = never
queue
```

Now that the job has been submitted, running the command `condor_q` will display the list of jobs currently managed by the Condor system.

```
[lmk@imp condorsub]$ condor_q


-- Submitter: imp.phys.UVic.CA : <142.104.60.85:32771> : imp.phys.UVic.CA
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
  60.0   lmk            12/8  09:28   0+00:00:34 R  0   0.0  uname -a

1 jobs; 0 idle, 1 running, 0 held
```

Once the job has completed, two files will be created. One is a log file, in this case named `lmk.job1.08-Dec-2003-09.28.31.log`. The contents of this file, upon completion, are shown below.

```
000 (060.000.000) 12/08 09:28:32 Job submitted from host: <142.104.60.85:32771>
...
017 (060.000.000) 12/08 09:28:46 Job submitted to Globus
    RM-Contact: mercury.sao.nrc.ca/jobmanager-pbs
    JM-Contact: https://mercury.sao.nrc.ca:20079/5682/1070904518/
    Can-Restart-JM: 1
...
001 (060.000.000) 12/08 09:29:44 Job executing on host: mercury.sao.nrc.ca
...
005 (060.000.000) 12/08 09:31:25 Job terminated.
        (1) Normal termination (return value 0)
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
                Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
        0  -  Run Bytes Sent By Job
        0  -  Run Bytes Received By Job
        0  -  Total Bytes Sent By Job
        0  -  Total Bytes Received By Job
...
```

The other file, in this case `lmk.job1.08-Dec-2003-09.28.31.out`, contains the output of the job. The contents of the file for this example are given below.

```
Linux hg51 2.4.20 #1 SMP Mon Feb 10 15:37:54 EST 2003 i686 unknown
```

4.4.4 Comparison with Requirements

While Condor-G does add tracking and monitoring capabilities in a secure fashion, building upon the Globus Tools, this system is not capable of knowing the status of the entire grid at any given time. This is due to the fact that each submission machine is running a separate version of Condor-G. Therefore, while this solution is a step above the simple submission script developed in Section 4.2.2, it does not have the necessary awareness required for developing complex resource brokering algorithms.

4.5 Other Alternatives

4.5.1 Custom Job Manager

Another approach to creating a resource brokering system would be to create a custom job manager, running on one dedicated machine, to which all users would direct their jobs and which would in turn pass those jobs on to available grid resources. In this manner the entire status of the grid could be tracked, while proxy security concerns would be circumvented through using the standard Globus submission routes. To submit a job in this system, users would run a standard `globus-job-submit` command, specifying the resource to be the machine which runs the custom job manager.

To determine the plausibility of this approach, a test was conducted to ascertain whether it was possible to call a `globus-job-submit` command from within another `globus-job-submit` command. This involved submitting a simple script whose contents contained an instruction to submit another job. Unfortunately, the test failed. It was discovered that when a `globus-job-submit` is performed, the delegated proxy created on the remote machine is a *limited* proxy, which means that further delegated proxies cannot be created from it. Therefore, when the second `globus-job-submit` command is performed, the proxy cannot be properly delegated to the chosen resource, and authentication fails.

This is intentionally done by Globus for security reasons. In order for this approach to work, a way around this restriction would have to be found. Whether that security precaution should be overridden is something that would need to be further investigated.

4.5.2 Resource Reservation and Ticketing

An interesting approach to resource brokering was taken by the LHC Computing Grid (LCG) group. LCG has created a testbed to evaluate the third and newest version of the Globus Toolkit. This testbed is currently implementing a dummy version of a resource brokering model that makes resource reservations and creates tickets to those reservations [9].

A visual representation of the model is given in Figure 4. The user makes a request to a resource broker, which then determines which resources are available and selects one according to a chosen algorithm. The resource broker then reserves the desired resource, and sends a ticket for that reservation back to the user. Using this ticket, the user can submit the job to the reserved resource and job execution begins.

This model would meet all the requirements set out for the resource broker tool, as the centralized resource broker component would be able to track all jobs, and yet the client machine is responsible for submission, so proxies need not be copied between machines. However, this approach is also a lot more complex, and would take a lot more time to develop. It could be hoped that the LCG group would further refine it and then offer this solution to the grid community, or perhaps even for integration into the Globus Toolkit.

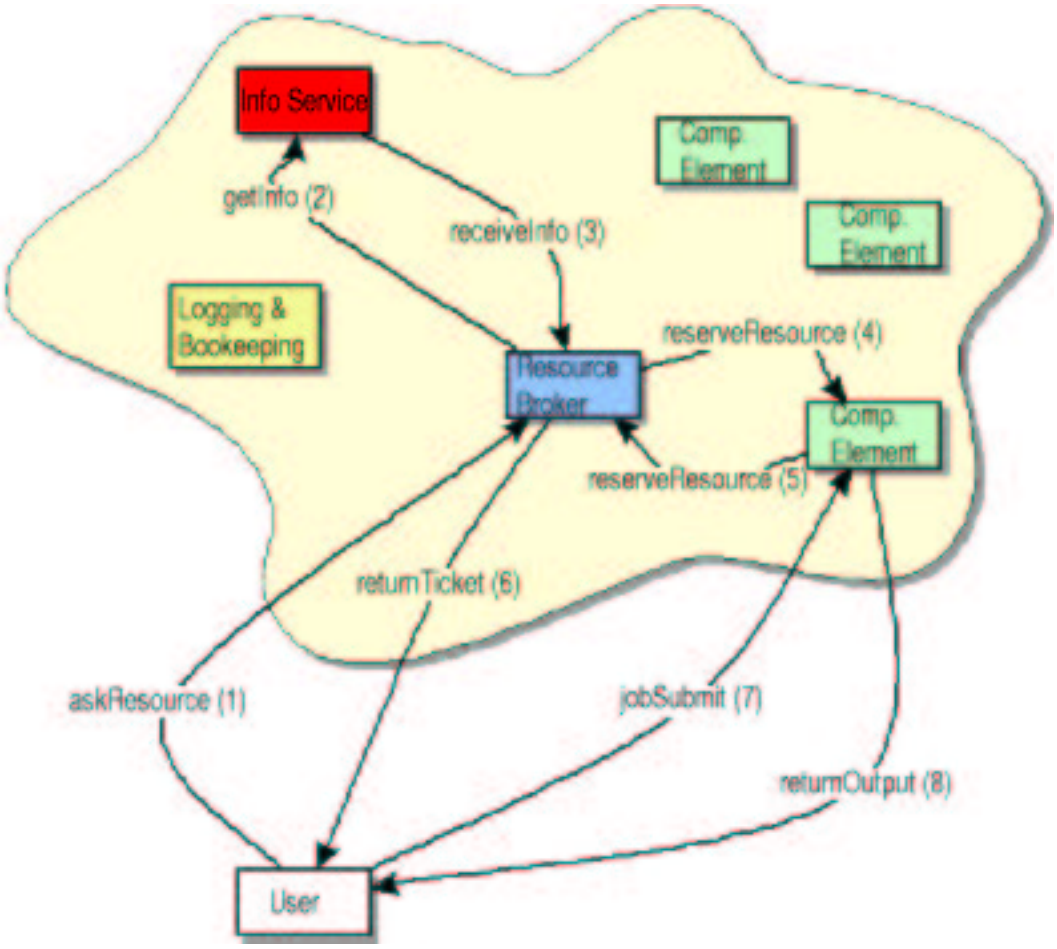

Figure 4: LCG resource brokering model [9]

5 Running Atlas Jobs Over the Grid

Now that a grid environment has been created, the Atlas software has been installed on the grid resources, and a resource brokering tool has been put in place, Atlas jobs can be run across the grid. This can be done using any of the brokering tools available, in combination with a custom script that sets up an Atlas environment in the user's home directory and runs the Atlas executable. The contents of this script are given in Appendix 7.

Once Atlas jobs have been submitted, the Ganglia software can be used to monitor the resources that each Atlas consumes. Sample outputs of Ganglia are shown in Figures 5 through 10. The configurations for each node displayed in these plots is as follows:

mercury.sao.nrc.ca, node hg53
CPU speed: 1995 MHz
Memory: 1032196 KB
Swap: 3047984 KB

mercury.sao.nrc.ca, node hg61
CPU speed: 1995 MHz
Memory: 2068428 KB
Swap: 4145128 KB

mercury.uvic.ca, all nodes
CPU speed: 2400 MHz
Memory: 1547620 KB
Swap: 1052216 KB

In the ganglia output, it can be seen that each Atlas job requires approximately 300-400 MB of memory, and takes approximately 5 hours to complete. As the processing power available decreases, simulations take longer to complete. This trend can be seen by looking at Figures 8, 9 and 10, where the processing power remains constant, but is shared between an increasing number of processes. It can also be observed by comparing Figure 5 with Figure 7, where each job uses 50% of the available CPU power, but the `mercury.uvic.ca` machine has more CPU power and thus completes the job almost an hour earlier.

In the figures shown, the time to run an Atlas simulation varies from 5 to 9 hours, depending upon the processing power available. This helps show the benefit that would come from employing an intelligent resource brokering algorithm, as jobs could then be sent to the best resource available. Almost two Atlas simulations could complete on a node that has no other processes running in the same amount of time that it would take one Atlas simulation to run on a node which already has three other processes running.

Each Atlas job requires an input file approximately 1.5 GB in size, and generates an output file of approximately 9 MB. Currently the input files exist locally with the Atlas installation, and the output files are also retained on the machine where Atlas was executed, in the submitting user's home directory. It might be beneficial in the future to keep all the input files on only a few select resources, and transfer them to the execution machine as necessary, due to the disk space they require. Also, an automated way of transferring the output to a selected repository may also be useful.

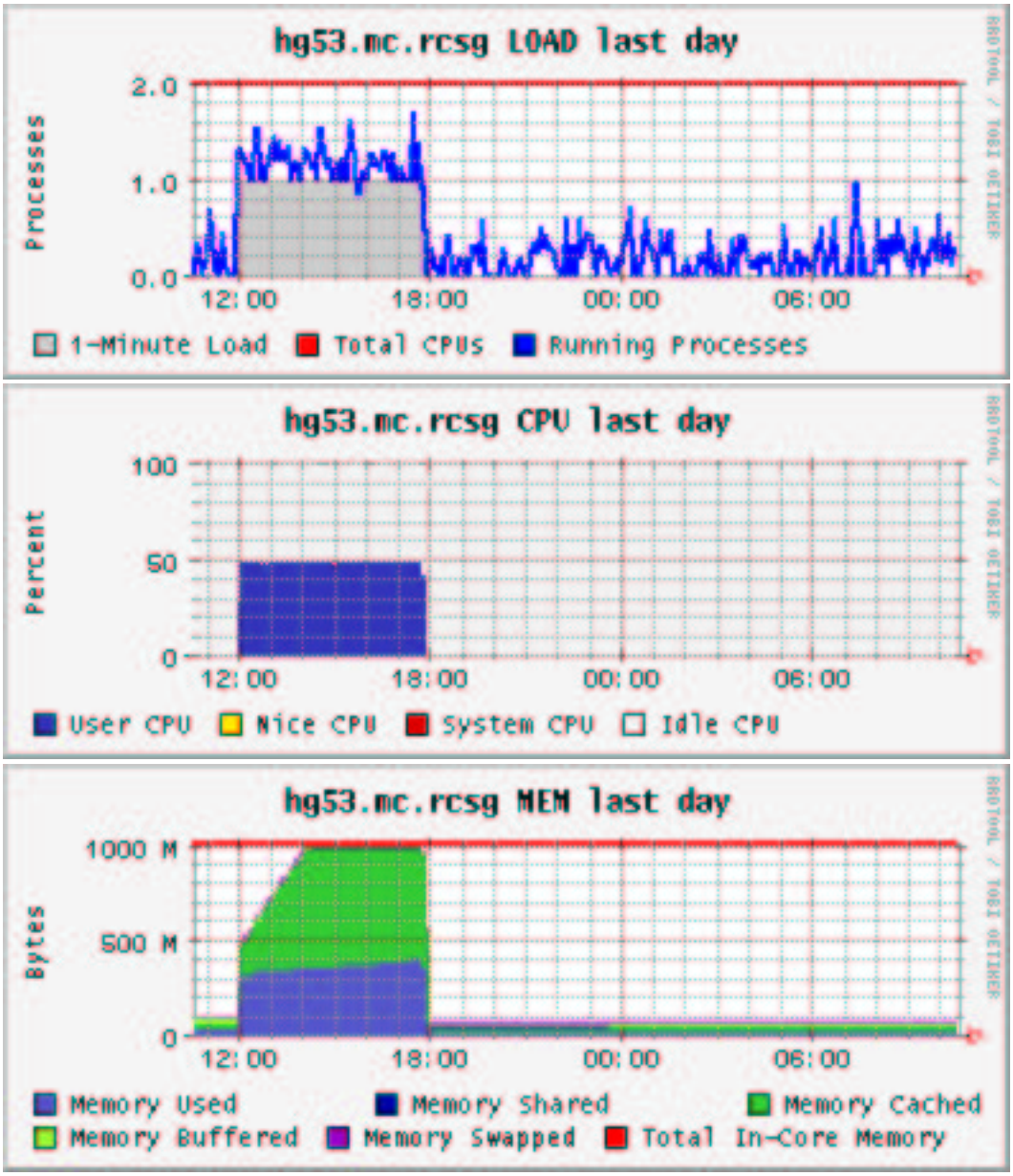

Figure 5: Ganglia output for mercury.sao.nrc.ca cluster, node hg53, 1 of 2 processors used.

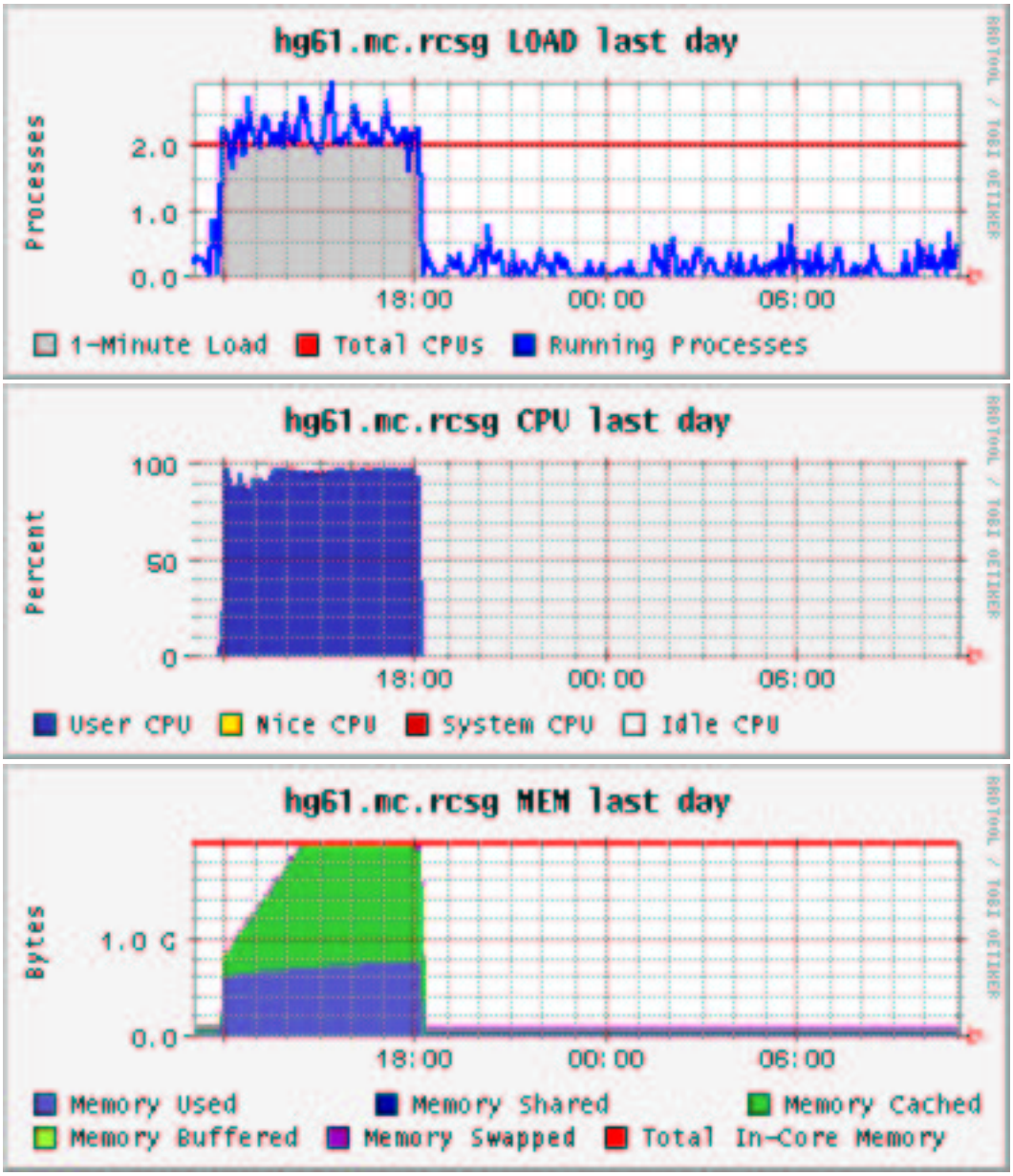

Figure 6: Ganglia output for mercury.sao.nrc.ca cluster, node hg61, 2 of 2 processors used.

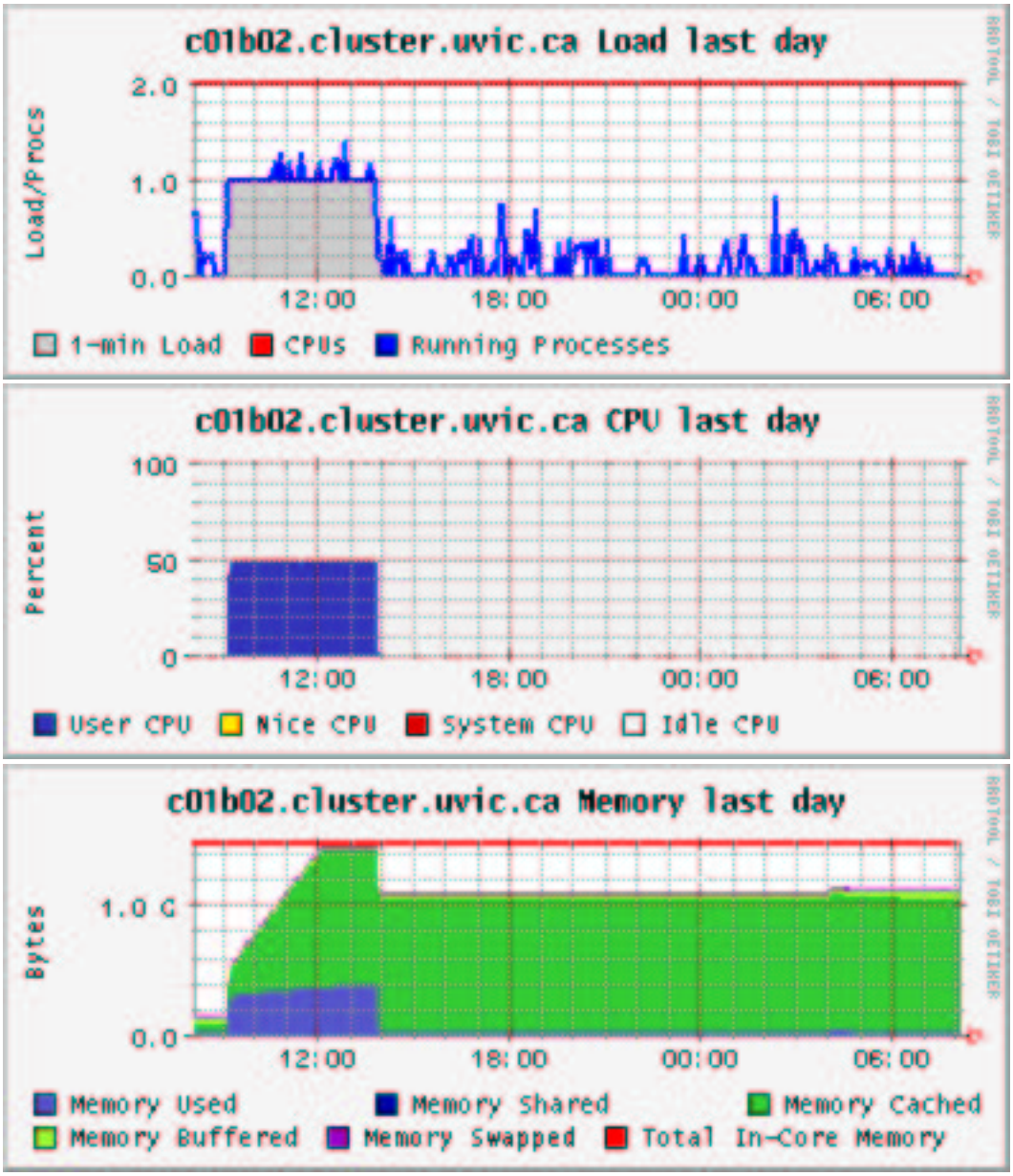

Figure 7: Ganglia output for mercury.uvic.ca cluster, node c01b02, 1 of 4 processors used.

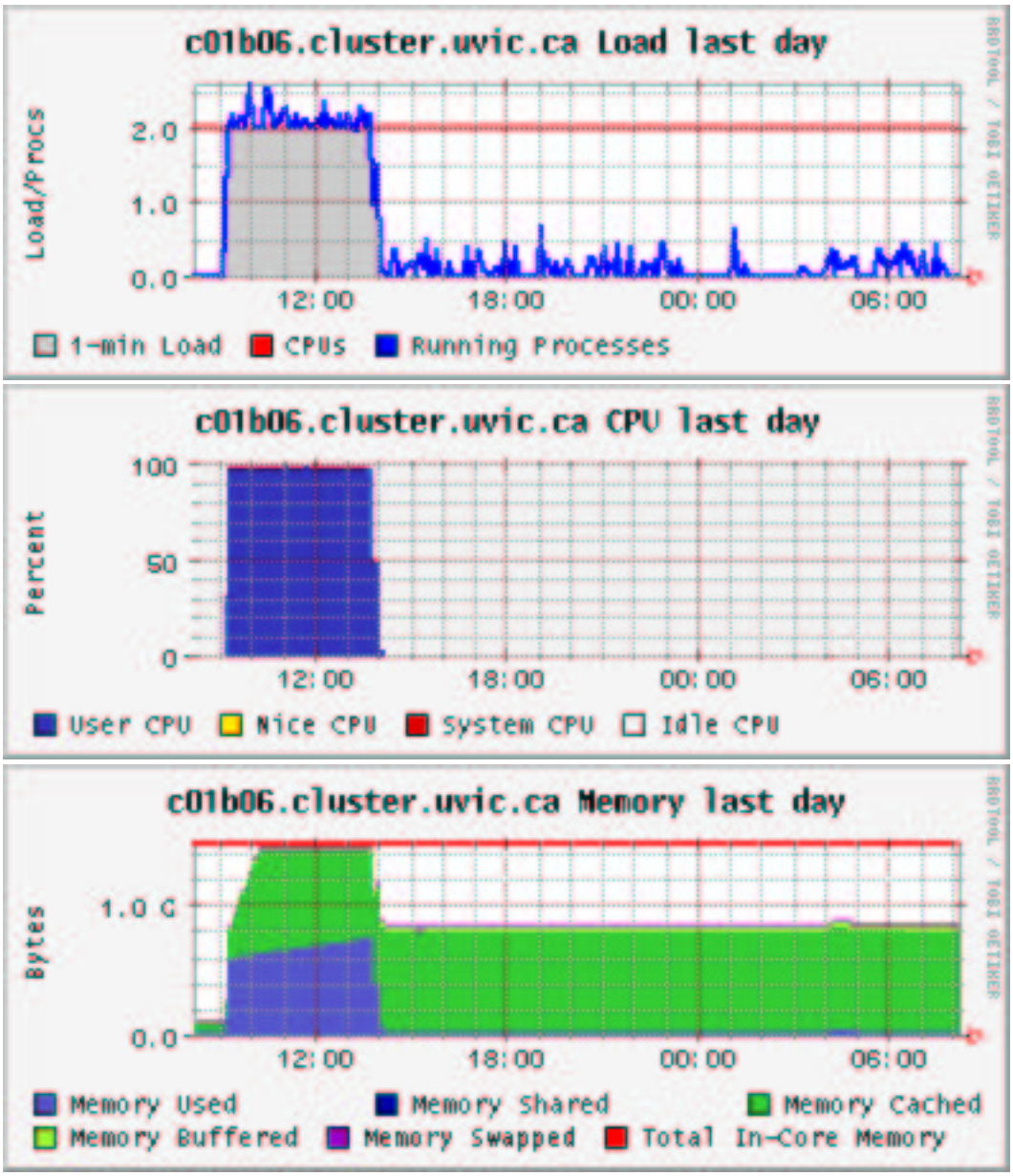

Figure 8: Ganglia output for mercury.uvic.ca cluster, node c01b06, 2 of 4 processors used.

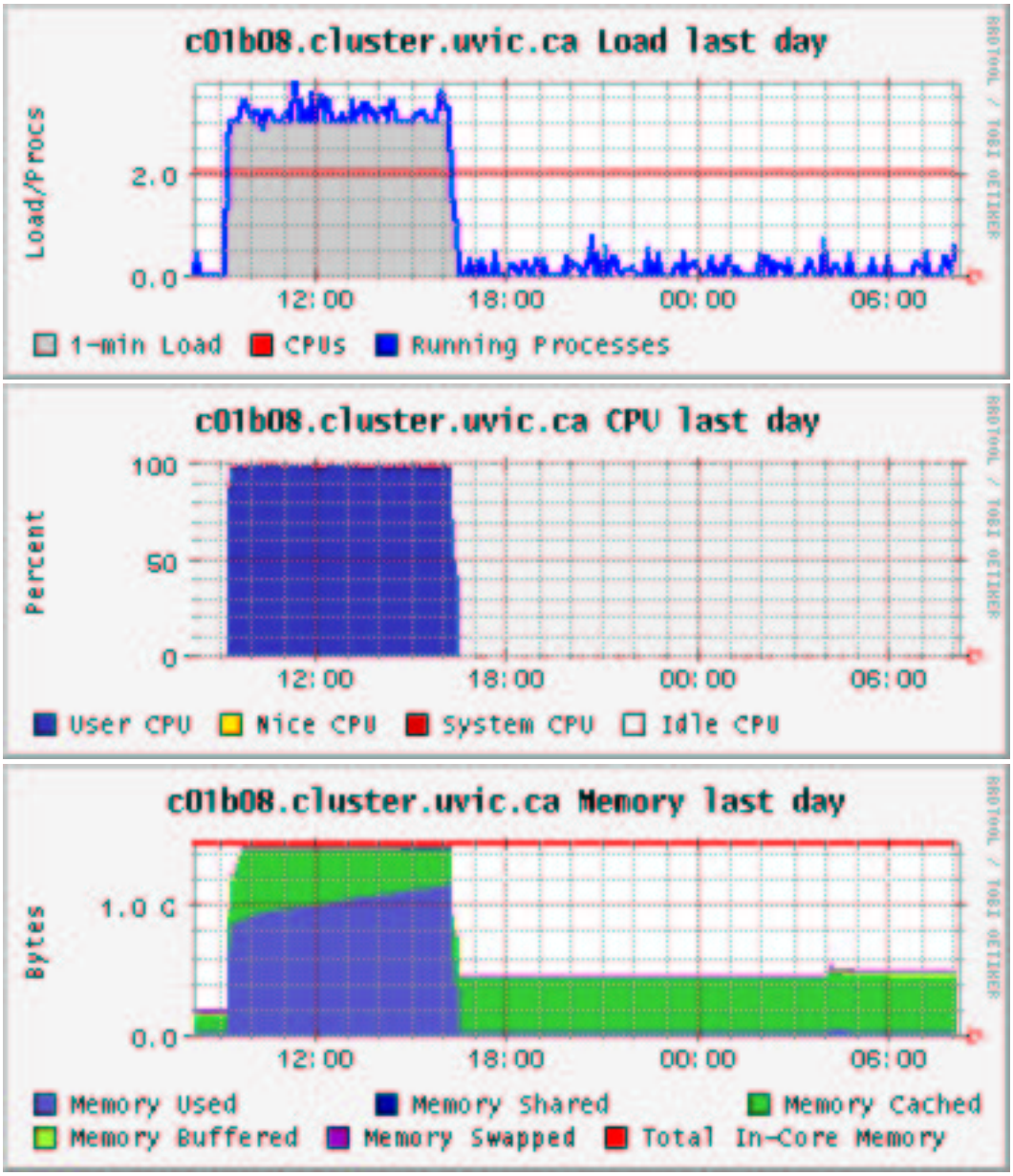

Figure 9: Ganglia output for mercury.uvic.ca cluster, node c01b08, 3 of 4 processors used.

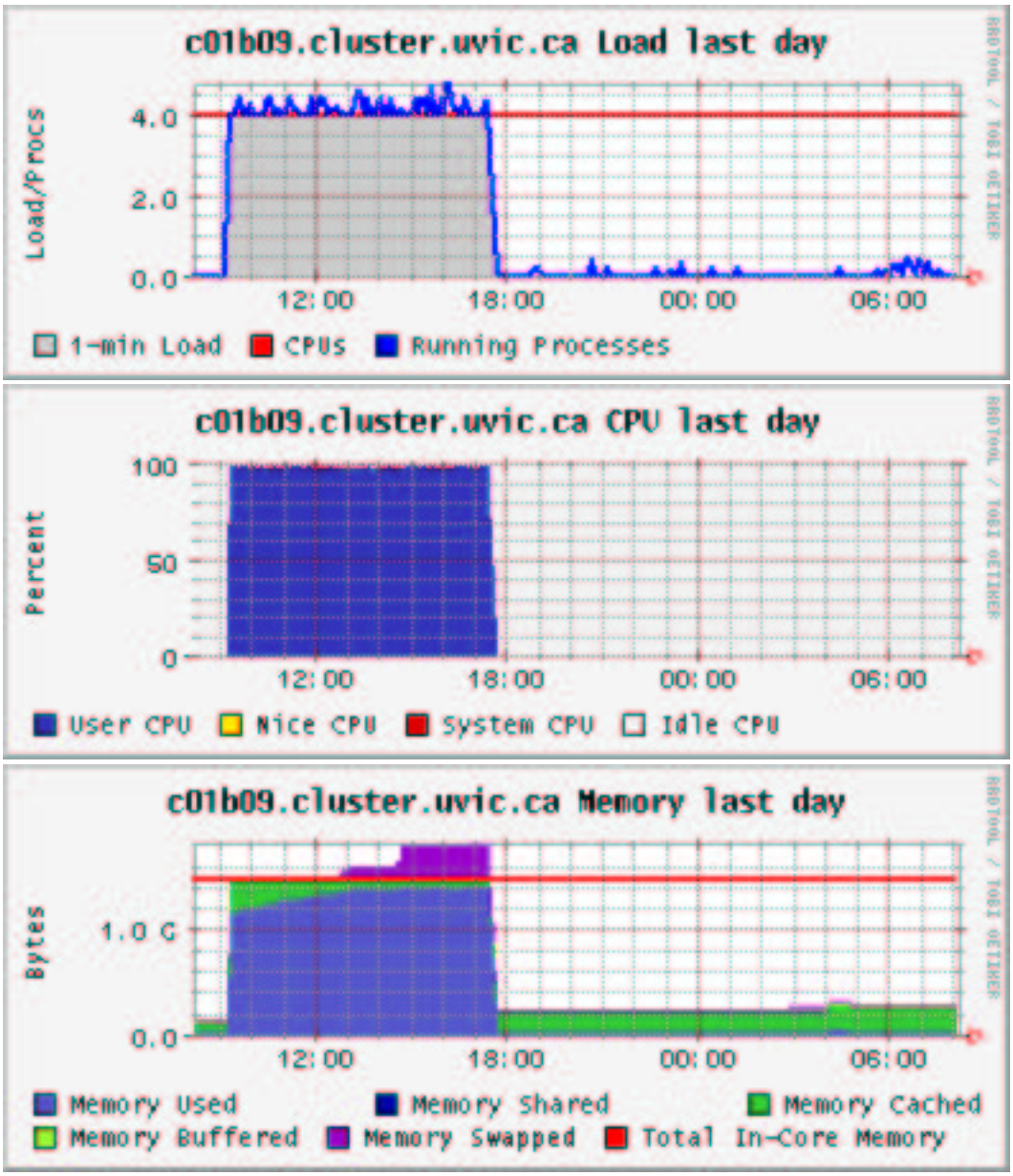

Figure 10: Ganglia output for mercury.uvic.ca cluster, node c01b09, 4 of 4 processors used.

6 Conclusion

Installing and running the Atlas software required three main tasks. The first task was to set up a grid environment. This has been successfully accomplished using the Globus Toolkit, as well as other supporting packages such as the Ganglia monitoring software and the PBS job queuing system.

The second task involved installing Atlas across the grid resources. A bash shell script was created to accomplish this, which utilizes Globus's job submission tools and Grid-FTP technology. The install requires transferring many files to the machine where it is to be installed, including 20 input files, each about 1.5 GB in size, and 91 rpm files which then need to be installed. The entire installation process requires 2-3 hours to complete, depending on network traffic and processor speeds.

The third task of creating a resource discovery and brokering mechanism was the most intensive, as there were many options to consider. Three systems were developed. The first is simply a prototype, and is only capable of simple brokering and submission. The second tool developed expands upon the first and, using a MySQL database as a queue for the grid, is capable of tracking job status. The third system is another adaptation of the first tool, which incorporate the Condor-G software for job tracking purposes. Of the three, the second tool is probably the most useful, although there are some improvements that should be made, particularly concerning security, before it is used at a production level.

After completing these steps, the Atlas software was run on the grid environment. It can be submitted using any of the resource brokering mechanisms, and the impact it has on each grid resource can be observed using the Ganglia software. It was noted that each Atlas job requires between 300-400 MB of memory, and the time to complete each simulation varied from 5 to 9 hours as available processing power decreased.

7 Recommendations

The method devised to install Atlas is currently successful. However, it could be improved upon by using MD5 checksums to verify the integrity of files transferred, especially the input files. It might also be beneficial to add logic that will determine if Atlas is already installed on the resource, so that a second redundant installation is not created. Also, if quicker installation times are desired, GSI-enabled OpenSSH should be considered as an alternative to sending commands through the Globus submission tools.

Concerning the resource brokering tools, the system referred to as "Solution 2" meets the requirements, and should be developed further. The most prominent issue that needs to be dealt with is the proxy security. The exact details of whether a copy of the user's proxy is secure or not should be determined. Depending upon the findings, the proxy should either be transferred using the MyProxy system, or else it should be sent in an encrypted format. A mechanism that will allow users to renew this proxy, should it expire before a job completes, should also be created. Another improvement that should be made is that staged files should be handled in an alternate manner. While the scripts submitted are currently small, the content of a staged file may not always fit inside the database. A means of transferring the file, possibly using Grid-FTP, should be researched and implemented.

In order to run Atlas jobs more efficiently, a proper resource brokering algorithm should be developed and implemented by the submission mechanism. The current method of choosing a resource at random is sufficient when testing other areas of the application, but a more intelligent decision should be made in order to maximize the efficiency of the grid environment. Also, it may be desirable to keep Atlas input and output files in specified repositories, as opposed to keeping them on the execution machines. Whether it is beneficial will depend on disk space available and network traffic speeds.

References

[1] Atlas Experiment. *http://atlasexperiment.org/.* (2003)

[2] Globus Alliance. *http://www.globus.org.* (2003)

[3] Portable Batch System. *http://www.openpbs.org/.* (2003)

[4] Ganglia: Distributed Monitoring and Execution System. *http://ganglia.sourceforge.net/.* (2003)

[5] Quesnel, Darcy. Grid Master of Grid Canada. Private Communication, October 2003.

[6] White, Dr. John. Research Associate. Private Communication, November 2003.

[7] GridBlocks. *http://gridblocks.sourceforge.net/.* (2003)

[8] Condor Project. *http://www.cs.wisc.edu/condor.* (2003)

[9] Foster, David et al. "OGSA/GT3 evaluation: Status Report". *http://lcg.web.cern.ch/LCG/PEB/GTA/GTA_OGSA/presentations/LCGseminar030924.ppt.* 3 September, 2003.

Appendix A - Atlas Installation Script

```
#!/bin/bash

VERSION=6.0.2-1

SRCHOST=mercury.uvic.ca
SRCATLASDIR=/home1x/hep/atlas
SRCZEBRADIR=${SRCATLASDIR}/${VERSION}/project/dc1/lumi02/002000/data
TEMPDIR=/home1x/gcprod/gcprod05
SRCLIB=/usr/X11R6/lib

DESTHOST=mercury.sao.nrc.ca
DESTATLASDIR=/home/gcprod05/atlas
DESTZEBRADIR=${DESTATLASDIR}/${VERSION}/project/dc1/lumi02/002000/data
DESTLIB=${DESTATLASDIR}/${VERSION}/installed/software/dist/6.0.2/InstallArea/i686-rh73-gcc295/lib

PREFIX=${DESTATLASDIR}/${VERSION}/installed
DBPATH=${DESTATLASDIR}/${VERSION}/rpmdb
echo ------------------

echo `date`: Tarring rpm and mysql files....

echo "#!/bin/bash
cd ${SRCATLASDIR}
tar cvf ${TEMPDIR}/atlas-kit.tar atlas-kit/${VERSION}

cd ${VERSION}
tar cvf ${TEMPDIR}/mysql.tar  MYSQL
" > temp-atlas-maketar.sh

echo globus-job-run ${SRCHOST} -s temp-atlas-maketar.sh
time globus-job-run ${SRCHOST} -s temp-atlas-maketar.sh
echo ------------------

echo `date`: Making directory structure....
echo globus-job-run ${DESTHOST} /bin/mkdir -p ${DESTZEBRADIR}
time globus-job-run ${DESTHOST} /bin/mkdir -p ${DESTZEBRADIR}
echo ------------------

echo `date`: Copying rpm files to ${DESTHOST}....
echo globus-url-copy -p 10 gsiftp://${SRCHOST}/${TEMPDIR}/atlas-kit.tar \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/atlas-kit.tar
time globus-url-copy -p 10 gsiftp://${SRCHOST}/${TEMPDIR}/atlas-kit.tar \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/atlas-kit.tar
echo ------------------

echo `date`: Copying mysql files to ${DESTHOST}....
echo globus-url-copy -p 10 gsiftp://${SRCHOST}/${TEMPDIR}/mysql.tar \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/mysql.tar
time globus-url-copy -p 10 gsiftp://${SRCHOST}/${TEMPDIR}/mysql.tar \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/mysql.tar
echo ------------------

echo `date`: Copying input files to ${DESTHOST}....
for ((i=141 ; ${i} <= 160 ; i++ )) ; do
     echo globus-url-copy -p 10 \
          gsiftp://${SRCHOST}/${SRCZEBRADIR}/dc1.002000.lumi02.00${i}.hlt.pythia_jet_17.zebra \
```

```
          gsiftp://${DESTHOST}/${DESTZEBRADIR}/dc1.002000.lumi02.00${i}.hlt.pythia_jet_17.zebra
     time globus-url-copy -p 10 \
          gsiftp://${SRCHOST}/${SRCZEBRADIR}/dc1.002000.lumi02.00${i}.hlt.pythia_jet_17.zebra \
          gsiftp://${DESTHOST}/${DESTZEBRADIR}/dc1.002000.lumi02.00${i}.hlt.pythia_jet_17.zebra
     echo ' '
done
echo ------------------

echo `date`: Copying configuration files to ${DESTHOST}
echo globus-url-copy -p 10 gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/eg7.602.job \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/eg7.602.job
time globus-url-copy -p 10 gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/eg7.602.job \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/eg7.602.job

echo globus-url-copy -p 10 gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/eg7.602.job.db \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/eg7.602.job.db
time globus-url-copy -p 10 gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/eg7.602.job.db \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/eg7.602.job.db

echo globus-url-copy -p 10 \
     gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/recon.gen.v5.with602rpmkit \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/recon.gen.v5.with602rpmkit
time globus-url-copy -p 10 \
     gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/recon.gen.v5.with602rpmkit \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/recon.gen.v5.with602rpmkit

echo globus-url-copy -p 10 \
     gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/recon.gen.v5.with602rpmkit_NG_db \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/recon.gen.v5.with602rpmkit_NG_db
time globus-url-copy -p 10 \
     gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/recon.gen.v5.with602rpmkit_NG_db \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/recon.gen.v5.with602rpmkit_NG_db

echo globus-url-copy -p 10 gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/noisedb.tgz \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/noisedb.tgz
time globus-url-copy -p 10 gsiftp://${SRCHOST}/${SRCATLASDIR}/${VERSION}/noisedb.tgz \
     gsiftp://${DESTHOST}/${DESTATLASDIR}/${VERSION}/noisedb.tgz
echo ------------------

echo `date`: Copy complete.  Starting installation
echo ------------------

echo `date`: Untarring mysql files....
echo globus-job-run ${DESTHOST} -l /bin/tar xvf ${DESTATLASDIR}/mysql.tar -C \
     ${DESTATLASDIR}/${VERSION}
time globus-job-run ${DESTHOST} -l /bin/tar xvf ${DESTATLASDIR}/mysql.tar -C \
     ${DESTATLASDIR}/${VERSION}
echo ------------------

echo `date`: Untarring rpm files....
echo globus-job-run ${DESTHOST} -l /bin/tar xvf ${DESTATLASDIR}/atlas-kit.tar -C \
     ${DESTATLASDIR}
time globus-job-run ${DESTHOST} -l /bin/tar xvf ${DESTATLASDIR}/atlas-kit.tar -C \
     ${DESTATLASDIR}
echo ------------------

echo `date`: Installing rpm files....
```

```
echo "#!/bin/bash
export PREFIX=${PREFIX}
export DBPATH=${DBPATH}
mkdir ${PREFIX}
mkdir ${DBPATH}
cd ${DESTATLASDIR}/atlas-kit/${VERSION}
rpm -Uvh --force --prefix ${PREFIX} --dbpath ${DBPATH} share/atlas-*.rpm
rpm -Uvh --force --prefix ${PREFIX} --dbpath ${DBPATH} release/atlas-*.rpm
" > temp-atlas-install.sh

echo globus-job-run ${DESTHOST} -s temp-atlas-install.sh
time globus-job-run ${DESTHOST} -s temp-atlas-install.sh
echo ------------------

echo `date`: Running relocate script....

echo "#!/bin/bash
export PREFIX=${PREFIX}
export DBPATH=${DBPATH}
cd ${PREFIX}/etc
./relocate ${PREFIX}
" > temp-atlas-relocate.sh

echo globus-job-run ${DESTHOST} -s temp-atlas-relocate.sh
time globus-job-run ${DESTHOST} -s temp-atlas-relocate.sh
echo ------------------

echo `date`: Customize scripts to ${DESTHOST}....
echo '#!/usr/bin/perl -w
#
# Replaces a string within multiple files
# specified on the command line

$mv = "/bin/mv";

$op = shift || die("Usage: $0 perlexpr dir [filenames]\n");

$dir = shift || die("Usage: $0 perlexpr dir [filenames]\n");

if (!@ARGV) {
  @ARGV = <STDIN>;
  chop(@ARGV);
}

chdir "$dir" or die "Cannot cd to $dir: $!\n";

foreach $file (@ARGV) {
  if (!-f $file) { print "Skipping non-regular file: $file\n"; next; }
  if (-B $file) { print "Skipping binary file: $file\n"; next; }

  $outfile = "/tmp/$file.$$";

  open(FILE, $file) || die("Could not open $file: $!\n");
  undef $/;
  $_ = <FILE>;
  close(FILE);

  if (eval $op) {
```

```
    open(OFILE, "> $outfile") || die("Could not open $outfile: $!\n");
    print OFILE;
    close(OFILE);

    system($mv, "-f", $file, "$file.bak");
    system($mv, "-f", $outfile, $file);

    print "File changed: $file\n";
  }
  else {
    print "No change to file: $file\n";
  }
}

exit(0);
' > temp-atlas-replace.pl

echo globus-job-run ${DESTHOST} -s temp-atlas-replace.pl \
     "s@${SRCATLASDIR}@${DESTATLASDIR}@g" ${DESTATLASDIR}/${VERSION} \
     recon.gen.v5.with602rpmkit recon.gen.v5.with602rpmkit_NG_db
time globus-job-run ${DESTHOST} -s temp-atlas-replace.pl \
     "s@${SRCATLASDIR}@${DESTATLASDIR}@g" ${DESTATLASDIR}/${VERSION} \
     recon.gen.v5.with602rpmkit recon.gen.v5.with602rpmkit_NG_db

echo globus-job-run ${DESTHOST} -s temp-atlas-replace.pl "s@${SRCHOST}@${DESTHOST}@g" \
     ${DESTATLASDIR}/${VERSION} eg7.602.job
time globus-job-run ${DESTHOST} -s temp-atlas-replace.pl "s@${SRCHOST}@${DESTHOST}@g" \
     ${DESTATLASDIR}/${VERSION} eg7.602.job

echo globus-job-run ${DESTHOST} -s temp-atlas-replace.pl "s@${SRCHOST}@localhost@g" \
     ${DESTATLASDIR}/${VERSION} eg7.602.job.db
time globus-job-run ${DESTHOST} -s temp-atlas-replace.pl "s@${SRCHOST}@localhost@g" \
     ${DESTATLASDIR}/${VERSION} eg7.602.job.db
echo ------------------

echo `date`: Making scripts executable....
echo "#!/bin/bash
cd ${DESTATLASDIR}/${VERSION}
chmod a+x recon.gen.v5.with602rpmkit
chmod a+x recon.gen.v5.with602rpmkit_NG_db
" > temp-atlas-chmod.sh

echo globus-job-run ${DESTHOST} -s temp-atlas-chmod.sh
time globus-job-run ${DESTHOST} -s temp-atlas-chmod.sh
echo ------------------

echo `date`: Copying necessary libraries....
echo globus-url-copy gsiftp://${SRCHOST}/${SRCLIB}/libXm.so.3 \
     gsiftp://${DESTHOST}/${DESTLIB}/libXm.so.3
time globus-url-copy gsiftp://${SRCHOST}/${SRCLIB}/libXm.so.3 \
     gsiftp://${DESTHOST}/${DESTLIB}/libXm.so.3
echo globus-url-copy gsiftp://${SRCHOST}/${SRCLIB}/libXm.so.3.0.1 \
     gsiftp://${DESTHOST}/${DESTLIB}/libXm.so.3.0.1
time globus-url-copy gsiftp://${SRCHOST}/${SRCLIB}/libXm.so.3.0.1 \
     gsiftp://${DESTHOST}/${DESTLIB}/libXm.so.3.0.1
echo ------------------

rm -rf temp-atlas*
```

```
echo ‘date‘: Installation complete.
echo ------------------

echo "Other tasks to complete:
- As root, make link on /usr/local: gcc-alt-2.95.2 -> \
     ${DESTATLASDIR}/${VERSION}/installed/i386_redhat73/usr.local/gcc-alt-2.95.2
- If linux distribution is not RedHat, create /etc/redhat-release file to trick atlas"
```

Appendix B - Solution 1: gcsub.pl Script

```
#!/usr/bin/perl -w
use strict;

use Net::LDAP;
use Getopt::Std;

my %opt;

##################
### functions ###
##################

sub usage() {
        print STDERR <<eot;
Usage: $0 [-h] [-v] [-d] [requirements...] (<-s filename> | <-l filename>) [-a arguments] \
       [-p parameter sweep]

where
    -h
        Show this help page.
    -v
        Verbose mode. Print some status messages to STDERR.
    -d
        Dry run. Build the job, find available resources, but do not submit the jobs.
    -l filename
        Execute this local executable. (eg. '/bin/uname')
    -s filename
        Stage and run this executable. (eg. '/bin/uname')
    -a arguments
        Supply these arguments (eg '-a'). Argument \%1 will be filled by the parametric
        sweep.
    -p sweep
        Run a parametric sweep. 'sweep' should be of the form x-y/j, x is the lower
        bound of \%1, y is the upper bound, and j is the increment.

                Example: '-p 1000-4999/100' will create 49 instances of the job with
                the argument \%1 varying from 1000, 1100, 1200, ..., 4800, 4900.

Requirements
    -O OS
        Only submit to this OS (eg. Linux).
    -C CPU MHz
        Only submit to resources with CPU MHz greater than C.
    -P platform
            Only submit to resources running this platform (eg. i686).
    -M memory MB
        Only submit to resources with memory MB greater than M.
eot
        exit 0;
}
sub printv($) { print STDERR $_[0] if $opt{v};}

sub error($) { die "Error: ".$_[0]; }

sub filter_auth($) {
        my $host = shift;
```

```
        return 1 if (`globusrun -a -r $host` =~ /successful/i);
        return 0;
}

sub filter_req($) {
        my $entry = shift;
        if (defined $opt{O} && $entry->exists('mds-os-name')) {
                return 0 if $entry->get_value('mds-os-name') ne $opt{O};
        }
        if (defined $opt{P} && $entry->exists('Mds-Computer-platform')) {
                return 0 if $entry->get_value('Mds-Computer-platform') ne $opt{P};
        }
        if (defined $opt{C} && $entry->exists('Mds-Cpu-speedMHz')) {
                return 0 if $entry->get_value('Mds-Cpu-speedMHz') < $opt{C};
        }
        if (defined $opt{M} && $entry->exists('Mds-Memory-Ram-sizeMB')) {
                return 0 if $entry->get_value('Mds-Memory-Ram-sizeMB') < $opt{M};
        }
        return 1;
}

sub param_sweep($$) {
        my $arg = shift;
        my $sweep = shift;
        my @jobs;
        printv("Running parameter sweep...\n");
        if ($sweep =~ /(\d+)-(\d+)\/(\d+)/) {
                my ($lb, $ub, $inc) = ($1, $2, $3);
                error("Malformed parameter sweep $sweep: LB less than UB") if $ub < $lb;
                for (my $i = $lb; $i<=$ub; $i=$i+$inc) {
                        my $thisarg = $arg;
                        $thisarg =~ s/\%1/$i/g;
                        push @jobs, $thisarg;
                }
        } else {
                error("Incorrectly formed parameter sweep $sweep:
                       Must be of form  xx-xx/xx");
        }
        return @jobs;
}

sub get_resources() {
        # Search the giis server for resources
        printv("Searching giis.gridcanada.ca...");
        my $ldap = Net::LDAP->new('giis.gridcanada.ca', port => '2135') or die "$@";
        $ldap->bind;
        my $mesg = $ldap->search (
        base   => "Mds-Vo-name=gc-production, o=Grid",
        filter => "objectClass=MdsHost",
        attrs => [      'mds-host-hn',
                                'mds-os-name',
                                'Mds-Cpu-speedMHz',
                                'Mds-Cpu-Total-count',
                                'Mds-Computer-platform',
                                'Mds-Memory-Ram-sizeMB'
                        ]
        );
        $ldap->unbind;
```

```
        printv("done.\n");
        return $mesg;
}

sub filter_resources($) {
        my $mesg = shift;
        my $n = $mesg->count;
        my @resources;
        printv("All hosts:\n");
        for (my $i = 0; $i<$n; $i++) {
                my $entry = $mesg->entry($i);
                my $hn = $entry->get_value('mds-host-hn');
                printv("\t".($i+1).". $hn");
                my $auth = (filter_auth($hn)?'have auth':'no auth');
                printv(" ($auth)");
                my $req = (filter_req($entry)?'req met':'req not met');
                printv(" ($req)\n");
                push @resources, $entry if $auth eq 'have auth' && $req eq 'req met';
        }
        printv("Available hosts:\n");
        $n = scalar @resources;
        for (my $i = 0; $i<$n; $i++) {
                printv("\t".($i+1).". ".$resources[$i]->get_value('mds-host-hn')."\n");
        }
        error("Error: no resources available which meet requirements.\n") if $n < 1;
        return @resources;
}

sub select_resource(@) {
        my @resources = @_;
        my $n = @resources;
        my $j = int(rand($n));
        my $hn = $resources[$j]->get_value('mds-host-hn');
        return $hn;
}
####################
### main program ###
####################

# Get the command line arguments
my $opt_string = 'hvds:l:C:P:M:O:a:p:';
getopts( "$opt_string", \%opt ) or usage();
usage() if ($opt{h} || (!defined $opt{s} && !defined $opt{l}));
error("-l and -s options are mutually exclusive.\n") if defined $opt{s} && defined $opt{l};
error("File ".$opt{-s}." does not exist!\n") if defined $opt{-s} && !-e $opt{-s};
error("File ".$opt{-l}." does not exist!\n") if defined $opt{-l} && !-e $opt{-l};
my $mode = 's';
$mode = 'l' if defined $opt{l};

# Prepare the job
my @jobs;
if (defined $opt{a} && $opt{a} =~ /(%\d+)/) {
        if(defined $opt{p}) {
                @jobs = param_sweep($opt{a},$opt{p});
        } else {
                error("No parameter sweep options given");
        }
} else {
```

```
        $opt{a} = '' if !defined $opt{a};
        push @jobs, $opt{a};
}

# Show jobs to be run
printv("Jobs to be run:\n");
foreach (@jobs) {
        printv("\t$opt{$mode} $_\n");
}

# Get list of available resources
my $all_resources = get_resources();
my @resources = filter_resources($all_resources);

# Pick a resource for each job and execute
foreach (@jobs) {
        printv("\t$opt{$mode} $_\n");
}

# Get list of available resources
my $all_resources = get_resources();
my @resources = filter_resources($all_resources);

# Pick a resource for each job and execute
foreach (@jobs) {
        my $hn = select_resource(@resources);
        printv("Selected $hn for $opt{$mode} $_\n");

        my $cmd = "globus-job-submit $hn/jobmanager-pbs -x '(maxtime=1439)' \
                    -$mode $opt{$mode} $_";
        print "$cmd\n";
        print `$cmd` if !defined $opt{d};
}
```

Appendix C - Solution 2: gcsub.pl Script

```
#!/usr/bin/perl -w
use strict;

use Getopt::Std;
use DBI();

my %opt;

###############
## functions ##
###############
sub usage() {
        print STDERR <<eot;
Usage $0 [-h] [-v] [-d] [requirements...] (<-s filename> | <-l filename>) [-a arguments] \
      [-p parameter sweep]

where
    -h
        Show this help page.
    -v
        Verbose mode. Print some status messages to STDERR.
    -d
        Dry run. Build the job, find available resources, but do not submit the jobs.
    -l filename
        Execute this local executable. (eg. '/bin/uname')
    -s filename
        Stage and run this executable. (eg. '/bin/uname')
    -a arguments
        Supply these arguments (eg '-a'). Argument \%1 will be filled by the parametric
        sweep.
    -p sweep
        Run a parametric sweep. 'sweep' should be of the form x-y/j, x is the lower
        bound of \%1, y is the upper bound, and j is the increment.

                Example: '-p 1000-4999/100' will create 49 instances of the job with
                the argument \%1 varying from 1000, 1100, 1200, ..., 4800, 4900.

Requirements
    -O OS
        Only submit to this OS (eg. Linux).
    -C CPU MHz
        Only submit to resources with CPU MHz greater than C.
    -P platform
             Only submit to resources running this platform (eg. i686).
    -M memory MB
        Only submit to resources with memory MB greater than M.
eot
        exit 0;
}
sub printv($) { print STDERR $_[0] if $opt{v};}
sub error($) { die "Error: ".$_[0]; }

sub log_job_start($$) {
        my $gcsub_cmd = shift;
        $gcsub_cmd = make_safe($gcsub_cmd);
        my $mode = shift;
```

```
my $job_id;
my $staged = 0;
my $req_os;
my $req_cpu;
my $req_memory;
my $req_platform;
my $filecontents = "";

if(defined $opt{O}) { $req_os = $opt{O}; }
if(defined $opt{C}) { $req_cpu = $opt{C}; }
if(defined $opt{M}) { $req_memory = $opt{M}; }
if(defined $opt{P}) { $req_platform = $opt{P}; }
if($mode eq 's') {
        $staged = 1;
        # read in staged file
        open INFILE, "< $opt{$mode}" or die "File does not exist";
        binmode(INFILE) if(-B $opt{$mode});
        local $/ = undef;
        $filecontents = <INFILE>;
        close INFILE;
}

# get information from proxy
my $user_identity = `grid-proxy-info -identity`;
chomp $user_identity;
my $proxy_file = `grid-proxy-info -path`;
chomp $proxy_file;
# read in proxy file
open INFILE, "<  $proxy_file" or die "Couldn't read proxy file";
local $/ = undef;
my $proxy = <INFILE>;
close INFILE;

# insert all job info into database
my $dbh = DBI->connect("DBI:mysql:database=job_history;host=grid.phys.uvic.ca",
                       "user", "password",{'RaiseError' => 1});

my $statement = "INSERT INTO job (filename,submit_datetime,staged,user_identity,
   req_os,req_cpu,req_memory,req_platform,gcsub_cmd,grid_proxy,staged_file)
   VALUES (".$dbh->quote($opt{$mode}).",now(),".$dbh->quote($staged). ",
   ".$dbh->quote($user_identity).",".$dbh->quote($req_os).",
   ".$dbh->quote($req_cpu).",".$dbh->quote($req_memory).",
   ".$dbh->quote($req_platform).",".$dbh->quote($gcsub_cmd).",
   ".$dbh->quote($proxy).",".$dbh->quote($filecontents).")";
eval { $dbh->do($statement) };
if($@) {
# if not successful, give error
        error "Insert job into database failed: $@\nThis job will not be run.\n";
} else {
# if successful, get job_id of record
        my $select = "SELECT last_insert_id() AS job_id";
        my $sth = $dbh->prepare($select);
        $sth->execute();
        my $ref = $sth->fetchrow_hashref();
        $job_id = $ref->{'job_id'};
        $sth->finish();
}
$dbh->disconnect();
```

```
        return $job_id;
}

sub log_process_start($$) {
        my $arguments = shift;
        $arguments = make_safe($arguments);
        my $job_id = shift;
        my $process_id = -1;

        my $dbh = DBI->connect("DBI:mysql:database=job_history;host=grid.phys.uvic.ca",
                               "user", "password",{'RaiseError' => 1});

        # insert process record
        my $statement = "INSERT INTO process (job_id,submit_datetime,arguments) VALUES
           (".$dbh->quote($job_id).",now(),".$dbh->quote($arguments).")";
        eval { $dbh->do($statement) };
        if ($@) {
        # if not successful, give error
                print "Insert into process into database failed: $@\nProcess will not \
                       be run\n";
        }
        $dbh->disconnect();
}
sub make_safe($) {
        my $text = shift;
        # make sure all backslashes and single quotes are backslashed once and only once
        $text =~ s/\\\\/\\/g;
        while ($text =~ /\\'/) {
                $text =~ s/\\'/'/g;
        }
        $text =~ s/\\/\\\\/g;
        $text =~ s/\'/\\'/g;
        return $text;
}

##############
## program ##
##############
# Get the command line arguments
my $gcsub_cmd = $0." ".join(" ",@ARGV);
my $opt_string = 'hvds:l:C:P:M:O:a:p:';
getopts( "$opt_string", \%opt ) or usage();
usage() if ($opt{h} || (!defined $opt{s} && !defined $opt{l}));
error("-l and -s options are mutually exclusive.\n") if defined $opt{s} && defined $opt{l};
error("File ".$opt{-s}." does not exist!\n") if defined $opt{-s} && !-e $opt{-s};
error("File ".$opt{-l}." does not exist!\n") if defined $opt{-l} && !-e $opt{-l};
my $mode = 's';
$mode = 'l' if defined $opt{l};

# Prepare the job
my @jobs;
if (defined $opt{a} && $opt{a} =~ /(%\d+)/) {
        printv("Running parameter sweep...\n");
        if (defined $opt{p}) {
                if ($opt{p} =~ /(\d+)-(\d+)\/(\d+)/) {
                        my ($lb, $ub, $inc) = ($1, $2, $3);
                        error("Malformed parameter sweep $opt{p}: LB less than UB") if $ub < $lb;
```

```
                        for (my $i = $lb; $i<=$ub; $i=$i+$inc) {
                                my $args = $opt{a};
                                $args =~ s/\%1/$i/g;
                                push @jobs, "$args";
                        }
                }
        }
} else {
        $opt{a} = '' if !defined $opt{a};
        push @jobs,"$opt{a}";
}

printv("Processes to be run:\n");
foreach (@jobs) {
        printv("\t$opt{$mode} $_\n");
}

# queue up process and log job if we're not doing a dry run
if(!defined $opt{d}) {
        # check that a valid proxy exists first
        my $proxy_time = `grid-proxy-info -timeleft`;
        if(($proxy_time eq "") || ($proxy_time=='-1')) {
                print "You do not have a valid proxy\n";
        } else {
                my $job_id = log_job_start($gcsub_cmd,$mode);
                foreach (@jobs) {
                        log_process_start($_,$job_id);
                }
        }
}
```

Appendix D - Solution 2: gcmanager.pl Script

```
#!/usr/bin/perl -w
use strict;

use DBI();
use Net::LDAP;
our $algorithm = 'random';
our $debugging = '1';
our $timeout = '60';
our $globus_path = "/homes/globus/globus-2.4.3/bin";

###############
## functions ##
###############
sub check_process_state ($$$$$) {
        my $grid_proxy = shift;
        my $process_id = shift;
        my $globus_id = shift;
        my $status_id = shift;
        my $dbh = shift;

        # get proxy for this user
        my $proxy = create_proxy($process_id,$grid_proxy);
        if($proxy eq "") {
                print ("Proxy for process $process_id is expired.  Cannot run process \
                        or check status.\n");
        # only continue checking process if proxy is valid
        } else {
                $ENV{'X509_USER_PROXY'} = $proxy;
                # for new jobs, submit them to the grid
                if($status_id eq 1 || $status_id eq 0) {
                       my $hn = choose_host($process_id,$dbh);
                        if($hn ne "") {
                                my $resource_id = get_hn_id($hn,$dbh);
                                my $cmd = make_globus_command($process_id,$hn,$dbh);
                                printv ($cmd."\n");
                                my $globus_id = exec_timed_command($cmd);
                                # check that globus command didn't time out or fail
                                if($globus_id =~ /https:\/\//) {
                                        my $update = "UPDATE process SET status_id=2,
                                           globus_id=".$dbh->quote($globus_id).",
                                           resource_id=".$dbh->quote($resource_id).",
                                           start_datetime=now(),command=".$dbh->quote($cmd)."
                                           WHERE process_id=".$dbh->quote($process_id);
                                        my $sth = $dbh->prepare($update);
                                        $sth->execute();
                                        $sth->finish;
                                # if it timed out, print error
                                } else {
                                        printv ("Submit process timed out\n");
                                }
                        }
                # for old jobs, update status
                } else {
                        my $cmd = "$globus_path/globus-job-status $globus_id";
                        print "$cmd\n";
```

```
                        my $status = exec_timed_command($cmd);
                        # check that globus command didn't time out
                        if($status ne "timeout") {
                                printv ("status is $status\n");

                                # get status id
                                my $select = "SELECT ps_id FROM process_status WHERE
                                   status=".$dbh->quote($status);
                                my $sth = $dbh->prepare($select);
                                $sth->execute();
                                my $ref = $sth->fetchrow_hashref();
                                my $status_id = $ref->{'ps_id'};
                                my $update = "UPDATE process SET status_id=".$dbh->quote($status_id);
                                if(($status eq "DONE") || ($status eq "FAILED")) {
                                        $update = $update.",end_datetime=now()";
                                }
                                $update = $update." WHERE process_id=".$dbh->quote($process_id);
                                $sth = $dbh->prepare($update);
                                $sth->execute();
                                $sth->finish;
                        # if it timed out, give error
                        } else {
                                printv("Check proces status timed out\n");
                        }
                }
        }
}
sub check_job_state($$) {
        my $job_id = shift;
        my $dbh = shift;

        # check to see if all processes are done
        my $select = "SELECT COUNT(*) AS count FROM process WHERE job_id=".$dbh->quote($job_id)."
           AND status_id < 5";
        my $sth = $dbh->prepare($select);
        $sth->execute();
        my $ref = $sth->fetchrow_hashref();
        my $count = $ref->{'count'};
        # mark job as not alive if it is finished and remove proxy
        if($count==0) {
                my $update = "UPDATE job SET alive=0,finish_datetime=now(),grid_proxy='',
                   staged_file='' WHERE job_id=".$dbh->quote($job_id);
                $sth = $dbh->prepare($update);
                $sth->execute();
                printv("Job $job_id is Done\n");
        }
        $sth->finish;
}
sub choose_host($$) {
        my $process_id = shift;
        my $dbh = shift;
        my $chosen_host;

        $chosen_host = alg_random($process_id,$dbh);
        return $chosen_host;
}
sub alg_random($$) {
        my $process_id = shift;
```

```
        my $dbh = shift;

        my $mesg = find_hosts($process_id,$dbh);

        # filter resources
        my $n = $mesg->count;
        my @resources;
        printv("All hosts:\n");
        for (my $i = 0; $i<$n; $i++) {
                my $entry = $mesg->entry($i);
                my $hn = $entry->get_value('mds-host-hn');
                printv("\t".($i+1).". $hn");
                my $auth = (filter_auth($hn)?'have auth':'no auth');
                printv(" ($auth)");
                my $req = (filter_req($entry,$process_id,$dbh)?'req met':'req not met');
                printv(" ($req)\n");
                push @resources, $entry if $auth eq 'have auth' && $req eq 'req met';
        }

        printv("Available hosts:\n");
        $n = scalar @resources;
        for (my $i = 0; $i<$n; $i++) {
                printv("\t".($i+1).". ".$resources[$i]->get_value('mds-host-hn')."\n");
        }

        print("Error: no resources available which meet requirements.\n") if $n < 1;

        return "" if($n==0);
        my $j = int(rand($n));
        return $resources[$j]->get_value('mds-host-hn');
}

sub find_hosts($$) {
        my $process_id = shift;
        my $dbh = shift;

        # Search the giis server for resources
        printv("Searching giis.gridcanada.ca...");
        my $ldap = Net::LDAP->new('giis.gridcanada.ca', port => '2135') or die "$@";
        $ldap->bind;
        my $mesg = $ldap->search (
        base   => "Mds-Vo-name=gc-production, o=Grid",
        filter => "objectClass=MdsHost",
        attrs => [      'mds-host-hn',
                        'mds-os-name',
                        'Mds-Cpu-speedMHz',
                        'Mds-Cpu-Total-count',
                        'Mds-Computer-platform',
                        'Mds-Memory-Ram-sizeMB'
                ]
        );
        $ldap->unbind;
        printv("done.\n");
        return $mesg;
}
sub filter_auth($) {
        my $host = shift;
        my $cmd = "globusrun -a -r $host";
```

```
        my $result = exec_timed_command($cmd);
        return 1 if ($result =~ /successful/i);
        return 0;
}
sub filter_req($$$) {
        my $entry = shift;
        my $process_id = shift;
        my $dbh = shift;

        my $select = "SELECT job.req_os,job.req_cpu,job.req_memory,job.req_platform FROM
           process,job WHERE job.job_id = process.job_id AND
           process.process_id =".$dbh->quote($process_id);
        my $sth = $dbh->prepare($select);
        $sth->execute();
        my $ref = $sth->fetchrow_hashref();

        if (defined $ref->{'req_os'} && $entry->exists('mds-os-name')) {
                return 0 if (!($entry->get_value('mds-os-name') =~ /$ref->{'req_os'}/i));
        }
        if (defined $ref->{'req_platform'} && $entry->exists('Mds-Computer-platform')) {
                return 0 if (!($entry->get_value('Mds-Computer-platform') =~
                                /$ref->{'req_platform'}/i));
        }
        if (defined $ref->{'req_cpu'} && $entry->exists('Mds-Cpu-speedMHz')) {
                return 0 if $entry->get_value('Mds-Cpu-speedMHz') < $ref->{'req_cpu'};
        }
        if (defined $ref->{'req_memory'} && $entry->exists('Mds-Memory-Ram-sizeMB')) {
                return 0 if $entry->get_value('Mds-Memory-Ram-sizeMB') < $ref->{'req_memory'};
        }
        return 1;
}
sub get_hn_id($$) {
        my $hn = shift;
        my $dbh = shift;

        my $select = "SELECT resource_id FROM resource WHERE hostname=".$dbh->quote($hn);
        my $sth = $dbh->prepare($select);
        $sth->execute();
        my $ref = $sth->fetchrow_hashref();
        my $resource_id = $ref->{'resource_id'};
        $sth->finish();
        return $resource_id;
}
sub make_globus_command($$$) {
        my $process_id = shift;
        my $hn = shift;
        my $dbh = shift;

        my $select = "SELECT process.arguments AS arguments ,job.staged AS staged,job.filename
           AS filename, job.staged_file AS staged_file FROM process,job WHERE
           process.job_id=job.job_id AND process.process_id=".$dbh->quote($process_id);
        my $sth = $dbh->prepare($select);
        $sth->execute();
        my $ref = $sth->fetchrow_hashref();

        my $mode;
        my $filename;
        if($ref->{'staged'}) {
```

```
                $mode = 's';
                $filename = "/tmp/processid$process_id.file";
                open(OUTFILE, ">$filename");
                print OUTFILE $ref->{'staged_file'};
                close(OUTFILE);
        } else {
                $mode = 'l';
                $filename = $ref->{'filename'};
        }
        my $cmd = "$globus_path/globus-job-submit $hn/jobmanager-pbs -x '(maxtime=1439)' \
                   -$mode $filename";
        if(defined $ref->{'arguments'}) {$cmd = $cmd." ".$ref->{'arguments'}; }
        return $cmd;
}
sub create_proxy($$){
        my $process_id = shift;
        my $proxy = shift;
        my $proxy_file = "/tmp/processid$process_id.grid.proxy";

        open(OUTFILE, ">$proxy_file");
        print OUTFILE $proxy;
        close(OUTFILE);
        `chmod 600 $proxy_file`;

        # check state of proxy
        my $cmd = "grid-proxy-info -f $proxy_file -timeleft";
        my $time = `$cmd`;
        chomp($time);
        return "" if(($time eq "") || ($time=='-1'));
        return $proxy_file;
}
sub exec_timed_command($) {
        my $cmd = shift;
        my $result;
        eval {
                local $SIG{ALRM} = sub { die "alarm\n" };       # NB \n required
                alarm $timeout;
                $result = `$cmd`;
                alarm 0;
        };
        die if $@ && $@ ne "alarm\n";       # propagate errors
        if ($@) {
                # timed out
                return "timeout";
        }
        else {
                chomp($result);
                return $result;
        }
}
sub printv($) { print STDERR $_[0] if ($debugging); }

##############
## program ##
##############
#check if lock exists, create a lock if not
if(-e 'gc.lock') { die ("program is locked"); }
else { `touch gc.lock`; }
```

```
# setup environment
$ENV{'LD_LIBRARY_PATH'}="/homes/globus/globus-2.4.3/lib";
$ENV{'GLOBUS_LOCATION'}="/homes/globus/globus-2.4.3/";
$ENV{'GLOBUS_PATH'}="/homes/globus/globus-2.4.3/";

my $dbh = DBI->connect("DBI:mysql:database=job_history;host=grid.phys.uvic.ca","user",
   "password",{'RaiseError' => 1});

# check status of processes
my $select = "SELECT job.grid_proxy,process.process_id,process.globus_id,process.status_id
   FROM process LEFT JOIN job ON process.job_id=job.job_id WHERE status_id < 5";
my $sth = $dbh->prepare($select);
$sth->execute();
while (my $ref = $sth->fetchrow_hashref()) {
        check_process_state($ref->{'grid_proxy'},$ref->{'process_id'},$ref->{'globus_id'},
                            $ref->{'status_id'},$dbh);
}

# check status of jobs
$select = "SELECT job_id FROM job WHERE alive=1";
$sth = $dbh->prepare($select);
$sth->execute();
while (my $ref = $sth->fetchrow_hashref()) {
        check_job_state($ref->{'job_id'},$dbh);
}

$sth->finish();
# remove any leftover proxy files
`rm -rf /tmp/processid*`;
# remove lock
`rm -rf gc.lock`;
```

Appendix E - Solution 3: gcsub.pl Script

```
#!/usr/bin/perl -w
use strict;

use Net::LDAP;
use Getopt::Std;

my %opt;

##################
### functions ###
##################

sub usage() {
        print STDERR <<eot;
Usage: $0 [-h] [-v] [-d] [requirements...] (<-s filename> | <-l filename>) [-a arguments] \
       [-p parameter sweep]

where
    -h
        Show this help page.
    -v
        Verbose mode. Print some status messages to STDERR.
    -d
        Dry run. Build the job, find available resources, but do not submit the jobs.
    -l filename
        Execute this local executable. (eg. '/bin/uname')
    -s filename
        Stage and run this executable. (eg. '/bin/uname')
    -a arguments
        Supply these arguments (eg '-a'). Argument \%1 will be filled by the parametric
        sweep.
    -p sweep
        Run a parametric sweep. 'sweep' should be of the form x-y/j, x is the lower
        bound of \%1, y is the upper bound, and j is the increment.

                Example: '-p 1000-4999/100' will create 49 instances of the job with
                the argument \%1 varying from 1000, 1100, 1200, ..., 4800, 4900.

Requirements
    -O OS
        Only submit to this OS (eg. Linux).
    -C CPU MHz
        Only submit to resources with CPU MHz greater than C.
    -P platform
            Only submit to resources running this platform (eg. i686).
    -M memory MB
        Only submit to resources with memory MB greater than M.
eot
        exit 0;
}
sub printv($) { print STDERR $_[0] if $opt{v};}

sub error($) { die "Error: ".$_[0]; }

sub filter_auth($) {
        my $host = shift;
```

```
        return 1 if (`globusrun -a -r $host` =~ /successful/i);
        return 0;
}

sub filter_req($) {
        my $entry = shift;
        if (defined $opt{O} && $entry->exists('mds-os-name')) {
                return 0 if $entry->get_value('mds-os-name') ne $opt{O};
        }
        if (defined $opt{P} && $entry->exists('Mds-Computer-platform')) {
                return 0 if $entry->get_value('Mds-Computer-platform') ne $opt{P};
        }
        if (defined $opt{C} && $entry->exists('Mds-Cpu-speedMHz')) {
                return 0 if $entry->get_value('Mds-Cpu-speedMHz') < $opt{C};
        }
        if (defined $opt{M} && $entry->exists('Mds-Memory-Ram-sizeMB')) {
                return 0 if $entry->get_value('Mds-Memory-Ram-sizeMB') < $opt{M};
        }
        return 1;
}

sub param_sweep($$) {
        my $arg = shift;
        my $sweep = shift;
        my @jobs;
        printv("Running parameter sweep...\n");
        if ($sweep =~ /(\d+)-(\d+)\/(\d+)/) {
                my ($lb, $ub, $inc) = ($1, $2, $3);
                error("Malformed parameter sweep $sweep: LB less than UB") if $ub < $lb;
                for (my $i = $lb; $i<=$ub; $i=$i+$inc) {
                        my $thisarg = $arg;
                        $thisarg =~ s/\%1/$i/g;
                        push @jobs, $thisarg;
                }
        } else {
                error("Incorrectly formed parameter sweep $sweep:  Must be of form  xx-xx/xx");
        }
        return @jobs;
}

sub get_resources() {
        # Search the giis server for resources
        printv("Searching giis.gridcanada.ca...");
        my $ldap = Net::LDAP->new('giis.gridcanada.ca', port => '2135') or die "$@";
        $ldap->bind;
        my $mesg = $ldap->search (
        base   => "Mds-Vo-name=gc-production, o=Grid",
        filter => "objectClass=MdsHost",
        attrs => [      'mds-host-hn',
                                'mds-os-name',
                                'Mds-Cpu-speedMHz',
                                'Mds-Cpu-Total-count',
                                'Mds-Computer-platform',
                                'Mds-Memory-Ram-sizeMB'
                        ]
        );
        $ldap->unbind;
        printv("done.\n");
```

```
        return $mesg;
}

sub filter_resources($) {
        my $mesg = shift;
        my $n = $mesg->count;
        my @resources;
        printv("All hosts:\n");
        for (my $i = 0; $i<$n; $i++) {
                my $entry = $mesg->entry($i);
                my $hn = $entry->get_value('mds-host-hn');
                printv("\t".($i+1).". $hn");
                my $auth = (filter_auth($hn)?'have auth':'no auth');
                printv(" ($auth)");
                my $req = (filter_req($entry)?'req met':'req not met');
                printv(" ($req)\n");
                push @resources, $entry if $auth eq 'have auth' && $req eq 'req met';
        }
        printv("Available hosts:\n");
        $n = scalar @resources;
        for (my $i = 0; $i<$n; $i++) {
                printv("\t".($i+1).". ".$resources[$i]->get_value('mds-host-hn')."\n");
        }
        error("Error: no resources available which meet requirements.\n") if $n < 1;
        return @resources;
}

sub select_resource(@) {
        my @resources = @_;
        my $n = @resources;
        my $j = int(rand($n));
        my $hn = $resources[$j]->get_value('mds-host-hn');
        return $hn;
}

sub make_condorsub($$$$$) {
        my $hn = shift;
        my $mode = shift;
        my $job = shift;
        my $arg = shift;
        my $count = shift;
        my $username = `whoami`;
        chomp $username;
        my $time = `date +%d-%b-%Y-%H.%M.%S`;
        chomp $time;
        my $condorsub = "$username.job$count.$time";

        my $contents = "executable = $job\n";
        $contents.= "arguments = $arg\n" if ($arg ne "");
        if($mode eq 's') {
                $contents.= "transfer_executable = true\n";
        } else {
                $contents.= "transfer_executable = false\n";
        }
        $contents.= "globusscheduler = $hn/jobmanager-pbs\n";
        $contents.= "globusrsl = (maxtime=1439)\n";
        $contents.= "universe = globus\n";
        $contents.= "output = $condorsub.out\n";
```

```
        $contents.= "log = $condorsub.log\n";
        $contents.= "notification = never\n";
        $contents.= "queue";

        open(OUTFILE, ">$condorsub");
        print OUTFILE $contents;
        close(OUTFILE);

        return $condorsub;
}

####################
### main program ###
####################

# Get the command line arguments
my $opt_string = 'hvds:l:C:P:M:O:a:p:';
getopts( "$opt_string", \%opt ) or usage();
usage() if ($opt{h} || (!defined $opt{s} && !defined $opt{l}));
error("-l and -s options are mutually exclusive.\n") if defined $opt{s} && defined $opt{l};
error("File ".$opt{-s}." does not exist!\n") if defined $opt{-s} && !-e $opt{-s};
error("File ".$opt{-l}." does not exist!\n") if defined $opt{-l} && !-e $opt{-l};
my $mode = 's';
$mode = 'l' if defined $opt{l};

# Prepare the job
my @jobs;
if (defined $opt{a} && $opt{a} =~ /(%\d+)/) {
        if(defined $opt{p}) {
                @jobs = param_sweep($opt{a},$opt{p});
        } else {
                error("No parameter sweep options given");
        }
} else {
        $opt{a} = '' if !defined $opt{a};
        push @jobs, $opt{a};
}

# Show jobs to be run
printv("Jobs to be run:\n");
foreach (@jobs) {
        printv("\t$opt{$mode} $_\n");
}

# Get list of available resources
my $all_resources = get_resources();
my @resources = filter_resources($all_resources);

# Pick a resource for each job and execute
my $count = 1;
foreach (@jobs) {
        my $hn = select_resource(@resources);
        printv("Selected $hn for $opt{$mode} $_\n");
        my $condorsub = make_condorsub($hn, $mode, $opt{$mode}, $_, $count);
        printv("condor_submit $condorsub\n");
        `condor_submit $condorsub`;
        $count++;
}
```

Appendix F - Script for Running Atlas Simulation

```
#!/bin/bash
echo Job Started at: `/bin/date`
echo Job Running on: `/bin/uname -a`

VERSION=6.0.2-1
ATLASDIR=~gcprod01/atlas/${VERSION}

# GET VARIABLES FROM ARGUMENTS
partition=00$1

# SET OTHER VARIABLES
DEBUG=true
DATASET=002000
LUMI=lumi02
DESCR=hlt.pythia_jet_17
ZEBRADIR=${ATLASDIR}/project/dc1/${LUMI}/${DATASET}/data
INPUTFILE=${ZEBRADIR}/dc1.${DATASET}.${LUMI}.${partition}.${DESCR}.zebra
EXECFILE=recon.gen.v5.with602rpmkit
JOBOPTFILE=eg7.602.job
MAXEVENT=1000

export JOBNAME=dc1.${DATASET}.${LUMI}.recon.${partition}.${DESCR}
export WORKDIR=~/atlas/${VERSION}/${JOBNAME}
export OUTDIR=~/atlas/${VERSION}/project/dc1/recon/data/${DATASET}

#echo variables
if ( ${DEBUG} == "true" ) ; then
        echo "Variables:"
        echo "  VERSION=${VERSION}"
        echo "  ATLASDIR=${ATLASDIR}"
        echo "  partition=${partition}"
        echo "  ZEBRADIR=${ZEBRADIR}"
        echo "  INPUTFILE=${INPUTFILE}"
        echo "  JOBNAME=${JOBNAME}"
        echo "  WORKDIR=${WORKDIR}"
        echo "  OUTDIR=${OUTDIR}"
fi

####################
### Start Script ###
####################

# SETUP
echo Job Setup Started at: `/bin/date`
if ( ${DEBUG} == "true" ) ; then
        echo "Setup Commands:"
        echo "  mkdir -p ${WORKDIR}"
        echo "  cd ${WORKDIR}"
        echo "  cp ${ATLASDIR}/${EXECFILE} ."
        echo "  cp ${ATLASDIR}/${JOBOPTFILE} ."
fi

mkdir -p ${WORKDIR}
cd ${WORKDIR}
cp ${ATLASDIR}/${EXECFILE} .
cp ${ATLASDIR}/${JOBOPTFILE} .
```

```
# ATHENA EXECUTION
echo Job Execution Started at: `/bin/date`
if ( ${DEBUG} == "true" ) ; then
        echo "Execute Command:"
        echo "  time ./${EXECFILE} ${INPUTFILE} ${JOBNAME}.ntuple ./${JOBOPTFILE} ${MAXEVENT} \
             >& ${JOBNAME}.log"
fi

time ./${EXECFILE} ${INPUTFILE} ${JOBNAME}.ntuple ./${JOBOPTFILE} ${MAXEVENT} >& ${JOBNAME}.log

# SAVE IMPORTANT FILES
echo Job Cleanup Started at: `/bin/date`
if [ ! -e ${OUTDIR} ]
then
        mkdir -p ${OUTDIR}
fi
if ( ${DEBUG} == "true" ) ; then
        echo "Cleanup Commands:"
        echo "  cp atlas.his ${OUTDIR}/${JOBNAME}.atlas"
        echo "  cp histo.hbook ${OUTDIR}/${JOBNAME}.histo"
        echo "  cp ${JOBNAME}.ntuple ${OUTDIR}/${JOBNAME}.ntuple"
        echo "  cp ${JOBNAME}.log ${OUTDIR}/${JOBNAME}.log"
fi

cp atlas.his         ${OUTDIR}/${JOBNAME}.atlas
cp histo.hbook       ${OUTDIR}/${JOBNAME}.histo
cp ${JOBNAME}.ntuple ${OUTDIR}/${JOBNAME}.ntuple
cp ${JOBNAME}.log    ${OUTDIR}/${JOBNAME}.log

# REMOVE WORKING DIRECTORY
cd ${WORKDIR}
cd ..
rm -rf ${WORKDIR}
echo Job Finished at: `/bin/date`
```