The GridX1 Monitoring Framework

by

Quinn Matthews
University of Victoria, Dept. of Physics
3800 Finnerty Road
Victoria, BC

Physics Co-op Work Term Report

in partial fulfillment
of the requirements of the Physics Co-op Program

Winter 2004

Quinn Matthews

Department of Physics and Astronomy
University of Victoria

December 21, 2004

Abstract

Modern particle physics experiments, such as the ATLAS experiment invloving the Large Hadron Collider (LHC) under development in CERN, are exceeding the limits of computational resources available at any single location. The solution is to form computational grids, networking computing clusters worldwide to provide an unprecedented amount of processor power and storage space required for these new scientific endeavors. Many grids are being formed all over the world. Most noteably, the Large Hadron Collider Computing Grid (LCG) is linking together computing clusters worldwide in an effort to prepare for the onslaught of data expected when the ATLAS project goes online on 2007. Canadian researchers are involved, and GridX1, an experimental grid consisting of clusters at the Uiversity of Victoria, University of Alberta, and the National Research Council, has recently joined forces with the LCG. As a result, ATLAS simulation jobs are being run on GridX1 (as well as other worldwide LCG sites) to encourage testing and development of grid technology. One of these developments has been a basic monitoring framework for GridX1, enabling grid users to access information regarding current cluster status and availability, as well as detailed job status information. This monitoring framework has been developed using existing grid software as well as other open source Unix packages, which are interfaced with Perl scripts, web development tools, and databases to provide a basic working framework for an expandable grid monitoring system. The tools used to develop the GridX1 monitoring framework will be discussed, as will the workings of the individual existing components in the framework itelf. The methods of presenting monitoring information on the GridX1 web page, www.grid.phys.uvic.ca/gridX1, will also be outlined.

Contents

List of Figures

1 Introduction

The World Wide Web was created by physicists as a means to communicate and share information world wide, and is now exploited on a massive scale by scientists and consumers alike. However, its major limitations in handling large amounts of data has become apparent due to the demands of modern scientists. Projects in high-tech fields such as High Energy Physics and Bio-Informatics are beginning to exceed the practical limits of on-site computational resources, even with modern super-computing technology at their disposal. For example, the ATLAS experiment involving the Large Hadron Collider (LHC) currently under construction and development at the European Laboratory for Particle Physics, CERN, in Geneva, Switzerland, is projected to produce an unprecedented amount of data, exceeding a petabyte (a million gigabytes) per year [1]. The only way to analyze that much data in a practical amount of time is by using a computational grid. Analagous to an electrical power grid where power generated in BC can be supplied to a consumer in California, a computational grid can provide computer power in Victoria to a scientist in Switzerland. By forming large networks of individual computing clusters, computational grids can provide a grid user with massive amounts of processor power, storage space, and memory via a high-speed network.

In answer to the demands of the ATLAS experiment the LHC Computing Grid (LCG) was developed. The LCG consists of processors, databases, and storage space, linked together by a specially designed software package and a high-speed network. External computing sites can join LCG and dedicate a set amount of resources for LCG use. Currently, with over 80 sites worldwide and 7000 CPUs, the LCG is well on its way to meeting the requirements of the ALTAS project, scheduled to go online in 2007 [1].

One of Canada's leading grid initiatives is Grid Canada, a partnership between CANARIE, the National Research Council, and C3.ca, which promotes grid computing across Canada [2]. One of Grid Canada's major projects is GridX1, an experimental computational grid across Canada. GridX1 consists of the University of Victoria, University of Alberta, National Research Council, TRIUMF Laboratory, and Simon Fraser University, all linked together via a high speed network provided by CANARIE. GridX1 has recently joined the LCG, providing considerable resources to the already worldwide grid.

GridX1 consists of 3 primary computing clusters (also referred to as 'hosts', 'sites' or 'resources'): 'mercury' at Victoria, 'thuner-gw' at Alberta, and 'mercury' at NRC. The 'WestGrid' cluster at SFU is not a member of GridX1, but is still hooked up and providing resources to LCG. The TRIUMF computing facilities act as a gateway between GridX1 and LCG. A job submitted from LCG is only submitted to TRIUMF, and then TRIUMF decides the best place for the job to be executed. The TRIUMF machine that does this is called a 'resource broker'. LCG has its own resource broker that will submit jobs to its various sites, of which TRIUMF is one. The TRIUMF resource broker queries the status of all available resources, determines where there is free CPU time, and submits the job. Once the job is finished, it retrieves the output and sends it back to LCG. A schematic for how these Canadian computing facilities are linked to LCG is shown in Figure 1 [3]. ATLAS DC2 stands for ATLAS Data Challenge 2, the current run of simulations being submitted to LCG to test its capabilities.

The UVic grid research team led by Dr. Randall Sobie is actively involved in the development and design of GridX1,

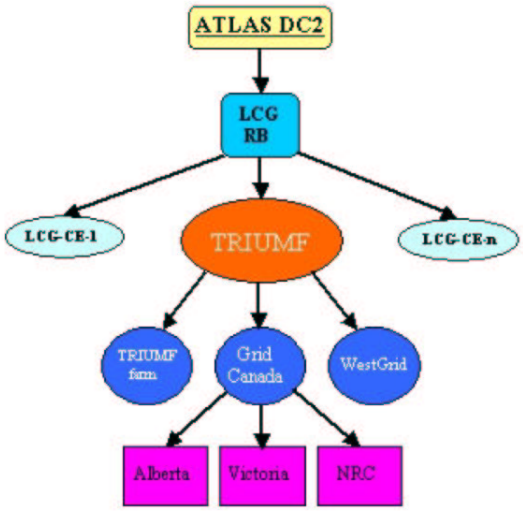

Figure 1: Canadian interface to LCG at CERN through TRIUMF resource broker [3]

which is still in its experimental stage. Now that GridX1 is actively running test jobs submitted via the LCG, the need becomes apparent for a vital aspect of any grid, a monitoring system. A basic monitoring package for a grid should be able to determine the current status of all the grid resources, show how many active or idle CPUs are at each cluster, give the current status of each job submitted to the grid and where it is running, plot historical information for a resource's status, and plot the number of jobs that have been run, completed, or failed on the grid. It should also have a database to store monitoring information, as well as a nice web interface complete with graphs and current status reports.

Using a combination of existing grid software, web development packages, and other open source software, a monitoring framework has been implemented that both satisfies the current basic requirements for GridX1, and is open for further improvements and additions as GridX1 evolves. The web interface to the monitoring framework can be viewed at the GridX1 home page, www.grid.phys.uvic.ca/gridX1, and by following the links provided. The reader of this document is strongly encouraged to refer to the web page and become familiar with all the monitoring components, as they will be discussed in detail at later sections of this document.

2 Description of Monitoring Tools

2.1 Grid Software

There are many existing software packages for working with grid computing, each with their own strengths and purposes. The GridX1 software achitecture is composed almost entirely from three pieces of software, the Globus Toolkit, Condor and CondorG, and PBS, each of which contribute vital components to the monitoring framework.

2.1.1 Globus Toolkit

The Globus Toolkit has become the standard for grid computing software, providing a set of binaries enabling a grid user to execute any basic grid operation, such as:

- User sign-on, authorization and authentication.
- Resource monitoring, searching and allocation.
- Job submission.
- Data movement.

The key services that Globus provides for monitoring is the ability to test a resource's availability, as well as test whether jobs can be run and data transfered to and from a grid resource.

2.1.2 Condor and CondorG

An important task for any grid is job scheduling and load balancing. Condor is a well known and widely used workload management software package, and CondorG is an extension to Condor that creates an interface with the Globus Toolkit so that Condor can be used in a grid environment. A Condor cluster has a central manager that collects the status of all the computing nodes in the cluster, and also matches resource requests for job submissions with a node that can satisfy this request. The CondorG central manager 'lcgce02.triumf.ca' is located at the TRIUMF particle accelerator in Vancouver, BC, and is the base for resource brokering on GridX1.

2.1.3 PBS

Portable Batch System (PBS) is a scheduling software that is installed locally on each GridX1 resource cluster. The PBS job server (one server node for each cluster) is responsible for creation, modification, running, and monitoring of jobs that have been submitted to the particular cluster. All jobs are submitted to a work queue. The PBS job scheduler (also on the server node) is responsible for scheduling jobs in the queue and querying the status and availability of all the execution nodes in the cluster.

2.2 Web Development Tools

In order to be practical and user accessable, a good monitoring system needs to have a web interface that is comprehensive, informative, and dynamic, yet also visually appealing and simple to understand. These goals were met by using the web development tools Plone and Zope.

2.2.1 Plone

Plone is an open source and user friendly web site content management system that is built using Zope (discussed below). To use Plone to manage a web site, you simply use any web browser and go to the site, log in with a username and password, and use basic HTML to edit the content of the visible pages. You can add new pages and create and edit simple objects or files, but to do anything more complex or dynamic, you need to enter the underlying layer of the Zope framework.

2.2.2 Zope

Zope is a widely used open source web application server and development system written in the Python programming language. Zope allows storage and access to not only web site content and data, but also to dynamic HTML templates, Python scripts, and connections to external relational databases such as MySQL. It also has its own FTP capabilities.

Plone and Zope work hand in hand to generate what the user views on the web page. Zope is controlled by the Zope Management Interface (ZMI), which is accessed through the web by logging into the Plone web interface and navigating to the ZMI link. The ZMI gives access to folders, files, user and style settings, preferences, and all other objects that are used to construct the dynamic aspects of the web page.

2.3 Unix Software

2.3.1 Perl and Perl modules

The Perl programming language provides an easy interface to the Unix command line, as well as plentiful tools for processing large amounts of text and numerical data that result from monitoring tests and database queries. Furthermore, Perl is very popular and there are a vast amount of external modules available to extend Perl's functions beyond its standard scripting capabilities. As such, almost all of the monitoring work for GridX1 is done via Perl scripts with various modules. All graphs are generated with the module GD::Graph, which uses the GD Graphics Library found at www.boutell.com/gd. Any image editing is done with the module Image::Magick, which uses the ImageMagick Library found at www.imagemagick.org. Any automated uploading of files to the webserver is done with the module Net::FTP. All database interaction is done with the module DBI. All modules are available from the Comprehensive Perl Archive Network (CPAN) website at www.cpan.org.

2.3.2 MySQL

MySQL is an open source database program that is very popular due to its speed, versatility, and ease of use. Of more importance to the GridX1 monitoring requirements, MySQL interacts very easily with Perl scripts, and is also accessible by the Zope webserver. Therefore, MySQL database data can be stored and retrieved easily from both a monitoring machine and a webserver environment.

2.3.3 Crontab

Any useful monitoring system should be almost completely automated. This can be done by using the Unix utiliy 'cron', which allows tasks to be automatically run in the background at either specified times of day, or repeating at regular intervals. A 'crontab' is a file that contains a schedule of cron entries to be run at the desired times. Furthermore, a task's output can be automatically written to a log file for reference or error checking.

3 Current Monitoring Framework

The current GridX1 monitoring framework that results in everything seen on the web page can be broken down into two main components: the webserver component, and the 'gcmon01' monitoring machine component. All of the actual monitoring work (except 'Condor-G Monitoring', discussed later) is done with Perl scripts on gcmon01. Data is stored in a MySQL database on gcmon01, and graphs generated by the Perl scripts are sent off to the webserver via FTP. The webserver component deals with retrieving the gcmon01 database data, and using Plone and Zope to effectively and dynamically display all data and graphs. These two components will be discussed separately.

3.1 The 'gcmon01' framework

The main monitoring components on gcmon01 are the crontab, the Perl scripts, the MySQL database, and the Oracle Database Client.

3.1.1 Crontab

All monitoring on gcmon01 is controlled by the crontab. The crontab contains 6 shell scripts that define any necessary environment variables and run the corresponding Perl scripts, as follows:

- 1. 'go_monitor.sh' runs, in order, 'monitor2.pl', 'make_map.pl', and 'make_linegraphs_24h.pl' every 15 minutes.
- 2. 'run_make_pie.sh' runs 'make_pie.pl' every 4 minutes.
- 3. 'run_make_bar.sh' runs 'make_bar.pl' every 4 minutes.
- 4. 'run_make_line_7d.sh' runs 'make_linegraphs_7d.pl' 4 times a day, at 00:20, 06:20, 12:20, and 18:20.
- 5. 'run_make_line_30d.sh' runs 'make_linegraphs_30d.pl' twice a day, at 00:40 and 12:40.
- 6. 'run_make_logger.sh' runs 'make_logger.pl' once a day, at 05:10.

3.1.2 Perl Scripts

'monitor2.pl' The 'monitor2.pl' script is responsible for the data in the 'Grid Monitor' box shown in the bottom right corner of the web page. The three tests run by this script, Gatekeeper, Job Submission, and GridFTP, use simple Globus commands, and effectively summarize the key Globus components that must be running on a host for it to be able to accept grid jobs. Gatekeeper simply ensures that Globus is ready and waiting on the remote host, Job Submission ensures that a simple job can be run on the host, and GridFTP ensures that data can be successfully transferred to and from the host.

The script first queries the local MySQL database for all the known host names and test types, and then goes through each host and runs the three tests to determine which Globus components are running on each host. For each command executed, the script waits a maximum of one minute to receive the expected successful output. If the correct output is received within the time limit, the command returns 'Pass'. If the command times out or returns output indicating a failure, then 'Fail' is returned. Each test result obtained is then stored in the MySQL database as a boolean Up or Down (1 or 0) value (see the 'MySQL Database' section for details on how this is done).

'make_map.pl' The 'make_map.pl' script is responsible for the map of Canada on the GridX1 home page, showing the current status of each resource. The script takes an existing map of Canada and edits the file using Image::Magick. It first adds the resource names and the corresponding color-coded status. It does this by querying the MySQL database for the recent results inserted by the 'monitor2.pl' script, and determining what color to use for each resource (green for all 3 tests passed, red otherwise). It then adds a legend and datestamp, writes the image to the current directory, and uploads the new image to the web server.

'make_line_graphs(24h,7d,30d).pl' These three scripts generate the historical test result line graphs for each resource, which can be viewed by navigating from the GridX1 home page to one of the 'site info and current status pages', and clicking on 'past monitoring test results'. These scripts graphically show test result data for each resource and test in the past 24 hour, 7 day, and 30 day periods. The main reason that three scripts were used instead of one larger script was so the scripts could be automatically run at different times, as shown in the 'crontab' section above.

The scripts first query the MySQL database and obtain all host names and test names. Then, for each host and test, the result and time data is pulled from the database into arrays, which are used to generate the graph. The MySQL method DATE_SUB(NOW(),INTERVAL 'X' HOUR) is used to pull only the database data entered in the last 'X' number of hours. Once the graph is made it is written to the current directory and uploaded to the webserver, and the script moves on to generate the next graph.

'make_pie.pl' This script obtains all the data shown in the 'site info and current status' pages (viewed by clicking on the links on the Canada map or in the Grid Monitor box), and generates the pie graphs showing each resource's job load. The

data is obtained in the script by sending remote jobs to the host machines using the Globus command 'globus-job-run'. The two jobs sent are the PBS commands 'qstat' and 'pbsnodes'. The 'qstat' command returns a list of all running and queued jobs at the resource, both those that have been submitted locally and those that have arrived via the GridX1 resource broker. The 'pbsnodes' command returns status information for all of the computing nodes at the cluster, such as the node name, its current status and availability, and its number of processors.

The script first defines the paths to the PBS executables on each remote machine. Then, for each resource, it runs the two commands and directs the output into Perl filehandles for processing. If the commands were successful, then the desired data is stored into the 'hosts' table in the MySQL database, the pie chart is generated, and the file is uploaded to the webserver. If the commands fail, then values of zero are entered into the database, and an error chart is created and uploaded.

'make_bar.pl' This script generates the GridX1 Job Load bar chart in the top right corner of the GridX1 web page. The data is obtained using a local installation of Condor which points at the GridX1 resource broker 'lcgce02.triumf.ca' at TRIUMF. The command 'condor_status' returns a description of all the responding resources' statistics that are relevant to the resource broker itself. For example, the number of CPUs the resource broker 'sees' at each resource is only the number of CPUs that are able to run grid jobs, not the total number of CPUs in the cluster. The command 'condor_q' returns a detailed description of each current job submitted to the broker, such as whether it is running or queued, which resource it is running at, and how long it has been running for. These two Condor commands are used together to get all the data for the bar graph.

First, 'condor_status' is used to get the names and general statistics of the GridX1 resources. Then 'condor_q' is used to obtain all the job information, and the number of jobs running and pending at each resource is calculated. The job data and resource names are loaded into arrays and used to generate the bar graph. The total number of jobs running and pending across GridX1 is calculated and added dynamically into the legend, and the graph is written to the current directory. A datestamp is then added with Image::Magick, the file is re-written, and then uploaded to the webserver.

'make_logger.pl' This script will generate the dynamic 12-month historical bar chart logging completed ATLAS jobs, which can be viewed on the GridX1 Logger page at www.grid.phys.uvic.ca/gridX1_logger. The data is obtained by querying an Oracle database at Cern. This is done by using Oracle Client, installed locally on gcmon01, and configured to connect to the Cern database (see the 'Oracle Database Client' section for more details).

The script queries the database for all records of jobs submitted to GridX1, getting data for start time, end time, job status ('finished', 'failed', or 'running') and job native status ('Done', 'Aborted', 'Cleared', or 'Running'). Each job record is loaded as a hash into an array. It then processes each hash in the array, and monthly totals for each year are calculated for jobs that were ran, as well as jobs that are 'finished' and 'Done', 'failed' and 'Done', 'failed' and 'Aborted', and 'failed and 'Cleared'. The actual meaning of all of these terms isn't completely clear, but these combinations account for all of the non-running jobs stored in the database. Currently, only jobs that are 'finished' and 'Done' in the last 12 months are displayed on the bar chart,

and the rest of the data is printed out to a log file. The script then uses the current date to decide how to sort the data, such that the last 12 months are always shown with the most recent month's data at the right side of the graph. It also determines the maximum monthly value, and uses this to scale the y-axis to stop at the nearest hundreds value above the maximum value. Finally, it calculates the 12 month total and the graph is generated, written to the current directory, and uploaded to the webserver.

3.1.3 MySQL Database

The local MySQL database 'newgridinfo' on gcmon01 (see Figure 2) contains all the raw data that is shown on the website, and is the repository for all the data received by the Perl monitoring scripts. The 'hosts' table contains the static data for the GridX1 sites (host_id, hostname, admin, ip), as well as the dynamic site data obtained and updated by the 'make_pie.pl' script (TotalCPUs, ActiveCPUs, NumGridCPUs, TotalJobs, NumGridJobs, NumLocalJobs). The 'test_type' table contains static information about the tests run on each grid resource by the 'monitor2.pl' script. The 'tests', 'test_result_boolean', 'test_result_float', 'test_result_string', and 'result_counter' tables work together to store all the data for every monitoring test that has been run against the grid resources.

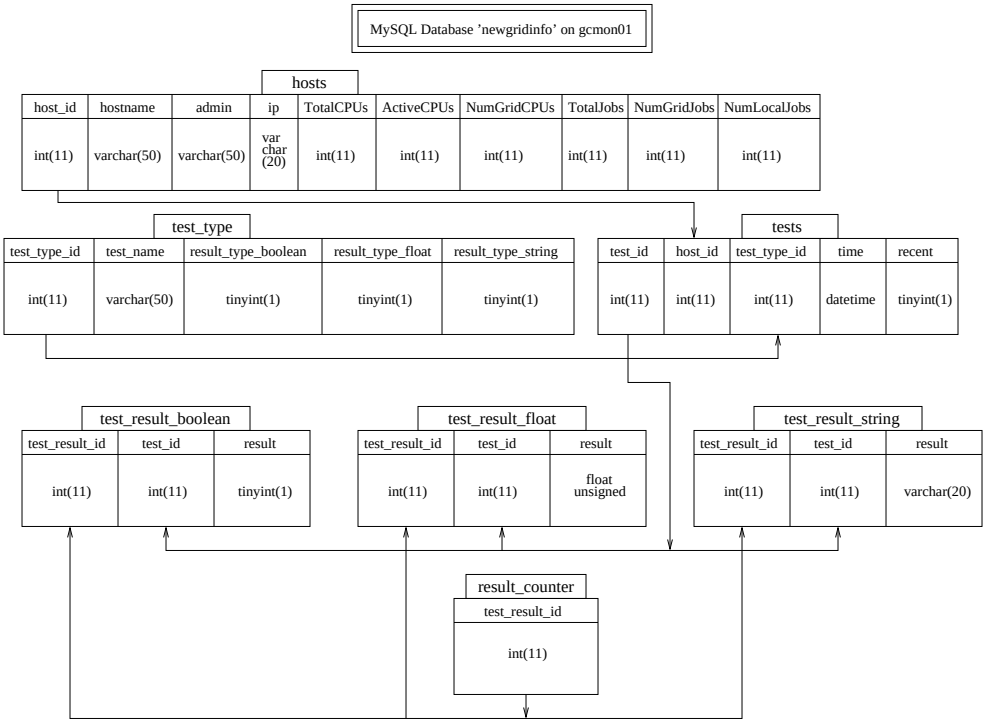

Figure 2: The MySQL database structure with column types

There is a strict algorithm that must be followed to enter new data into the database for each test done on a grid host (see the 'store_result' subroutine in the 'monitor2.pl' script to see it in action, Appendix A). First, the host name and test name must

be converted to the corresponding host_id and test_type_id, and the correct result types (boolean, float, string, or combination of these three) must be found that correspond to the current test type. Next, the 'tests.recent' column is updated such that all old entries are changed to 'recent' = '0' for this particular test and host. Then, the current host_id and test_type_id, the current time, and 'recent' = '1', are inserted into the 'tests' table causing the 'test_id' column to be auto-incremented by one. Next, for each result type, the 'result_counter' table is auto-incremented to obtain a 'test_result_id', and the 'test_result_id', 'test_id', and 'result' values are inserted into the corresponding result type tables. This algorithm allows for up to three different test results (boolean, float, string) for each 'test_id', where each result has its own unique 'test_result_id'.

3.1.4 Oracle Database Client

To obtain the data shown by the GridX1 Logger bar chart, Oracle Database Client v. 10g is currently installed on gcmon01 and configured to connect to the Cern Oracle v. 9i database 'pdb01' on servers 'sundb07.cern.ch' and 'sundb08.cern.ch'. After the installation of the Oracle Client software, there were two main steps needed for the remote connection. Firstly, the network configuration file 'tnsnames.ora' was configured to point to the remote database. Secondly, the Oracle environment variables were set, which are necessary to describe the paths to the local client installation, and to set the remote database's specifications. The remote database is then accessible for monitoring work by using the Perl module DBI.

3.2 The Webserver framework

The main monitoring components located on the webserver are accessed through the web site development tools Plone and Zope. The 'Condor-G Monitoring' framework will also be discussed.

3.2.1 Plone components

The Plone part of the monitoring framework is very straightforward. The home page displaying the Canada map, the GridX1 Logger page, the past monitoring test result pages with the line graphs, the Grid Monitor Information page, and the Grid Monitor Test Descriptions page are all built and edited within the Plone environment using basic HTML.

3.2.2 Zope components

Zope is where everything else happens (indeed, everything described above in the Plone components section can be accessed through Zope as well, but Plone is easier and quicker to use, if less flexible). The Zope part of the monitoring framework is responsible for the Grid Monitor box, the GridX1 Job Load box, and the 'site info and current status' pages with the pie charts and data tables.

Grid Monitor box There are many steps that lead to the formation of the Grid Monitor box in the bottom right corner of the web page. First, in the Zope folder '/gridX1', a Zope SQL method retrieves all the recent test result data from the database on gcmon01. Second, in the same folder, a Python script activates the SQL method and loads the test result data into an associative array. With a bit of formatting and some HTML, the script creates the nice table shown inside the box. Third, a Zope 'portlet' must be used to execute the Python script and wrap the output for viewing on the web page. This portlet file is a Zope Page Template (ZPT). ZPT's are HTML files that are pre-formatted to interface with a certain dynamic component of Zope, such as a portlet. Finally, the portlet is shown on the web page by loading it into a 'slot'. A slot is essentially a box that wraps the portlet output and displays it off to the side of a web page. The benefit of using slots is that they will appear in all pages that branch off of the main folder they are created in. Hence, both the Grid Monitor box and the GridX1 Job Load bar chart (see next section) will be visible throughout the GridX1 page, but not visible in the other www.grid.phys.uvic.ca pages.

GridX1 Job Load box The GridX1 Job Load box containing the job load bar chart is formed in much the same way as the Grid Monitor box, except that it is only showing an image file (uploaded every 4 minutes from gcmon01), rather than the output of a Python script. The image file is written to the '/gridX1' folder, and a ZPT is responsible for loading the image into a portlet. The portlet is then displayed in one of the GridX1 slots as described above.

Site Info and Current Status pages Each GridX1 resource 'site info and current status' page has its own Zope folder located at '/gridX1/monitor' which contain all the image files and Zope objects used to generate each page. All pages are generated by the same process. First, to obtain the data shown in the colour-coded table, two Zope SQL methods query the MySQL database on gcmon01. One method executes a query similar to the method used for the Grid Monitor box, but returns the latest monitoring test results for that site only. The other method obtains data from the 'hosts' table, including the dynamic data entered every 4 minutes by the 'make_pie.pl' Perl script. Next, a Python script activates both SQL methods and uses the returned data to determine the site's current 'Up' or 'Down' status, and to create a nice dynamic HTML table. In order to nicely display this dynamic table alongside the pie chart which is being uploaded every 4 minutes, a Document Template Markup Language (DTML) method is used. A DTML object allows HTML code to become dynamic by adding special tags which define the dynamic nature of the web page. In our case, the DTML tag dynamically executes the Python script and enables the output to be displayed. The DTML method uses regular HTML to format the page such that the pie chart and the data table are single elements in a larger table, with the other text placed below the table. The final step is to use a ZPT to render the DTML method such that the page is displayed nicely within the rest of the web page environment, appearing as a regular page alongside the ones created within the Plone interface.

3.2.3 Condor-G Monitoring

The Condor-G Monitoring page at www.grid.phys.uvic.ca/gridX1/condorg acts as a user interface to the 'condor_q' and 'condor_status' Condor commands described in the 'make_bar.pl' Perl script section. Each table shown is generated by a Perl script running on the webserver machine itself, which has a local Condor installation identical to the one on gcmon01. The Perl scripts are in turn called by external Python scripts also located on yamon. These Python scripts are accessed from the ZMI as Zope External Method objects within the '/gridX1' folder, which are in turn called by a DTML method which creates the dynamic format of the web page. Finally, as with the 'site info and current status' pages, a ZPT is used to render the DTML method and display the page within the Plone environment.

4 Conclusion

Through the methods discussed, a working monitoring system has been developed that will satisfy the basic needs for GridX1. A resource's current status and availability, as well as a detailed description of a resource's job load and processor availability is provided. Furthermore, important information is provided for each current job running on a resource. Historical information is provided as well, both for a resource's past status and stability, and for the total number of successful jobs ran by GridX1 in the past year.

5 Recommendations

The current GridX1 monitoring framework is open to many new developments now that a working framework has been implemented. Future additions should include network and connectivity testing between the GridX1 resources. Furthermore, there should eventually be an active interface between the monitoring system and the resource broker so that when resources are temporarily unavailable or unstable for extended periods, new jobs will not be submitted to that resource. Another useful addition would be automated noticifaction to grid administrators when the monitor detects problems with any grid component.

6 Acknowledgements

I would first and foremost like to thank Dr. Randy Sobie for providing me with the opportunity to contribute to the GridX1 team. I would also like to thank and acknowledge Dan Vanderster, Ashok Agarwal, David Bickle, and Lila Klektau for their time and efforts contributed to this project, who's previous work made this report possible.

7 Appendices

A 'monitor2.pl' script

```
#!/usr/bin/perl -w
# monitor2.pl

use strict;
use DBI;
use Term::ANSIColor qw(:constants);
use Expect;

our $current_path = "/home/gridmon/programs/monitor";
our $globus_path = "/opt/vdt/globus/bin";
our $timeout = 60;
our $fancy = 0;

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

# runtests runs every test against each site and puts results in db

sub runtests($$$$$$$){
my $site = shift;
my $dbh = shift;
my $hn_ref = shift;
my $tn_ref = shift;
my $rtb_ref = shift;
my $rtf_ref = shift;
my $rts_ref = shift;
my $cmd;
my $result;
my $authresult;

if ($fancy==1) { print BOLD, GREEN "--$site--\n", RESET;}
else { print "--$site--\n";}

# run a gatekeeper test
$authresult = exec_timed_command("$globus_path/globusrun -a -r $site","success");
store_result($site,"Gatekeeper",$authresult,$dbh,$hn_ref,$tn_ref,$rtb_ref,
$rtf_ref,$rts_ref);

# if gatekeeper fails, automatically set GridFTP and Job Submission to fail
if ($authresult !~ /Pass/){
store_result($site,"GridFTP","Fail",$dbh,$hn_ref,$tn_ref,$rtb_ref,
$rtf_ref,$rts_ref);
store_result($site,"Job Submission","Fail",$dbh,$hn_ref,$tn_ref,$rtb_ref,
$rtf_ref,$rts_ref);
return;
}
```

```
# run a batch job test
$result = exec_timed_command("$globus_path/globusrun -o -r $site/jobmanager-fork
&(executable=/bin/echo)(arguments=jobtest)","jobtest");
store_result($site,"Job Submission",$result,$dbh,$hn_ref,$tn_ref,$rtb_ref,
$rtf_ref,$rts_ref);

# if batch job fails, automatically set GridFTP to fail, and go to next site
        if ($result !~ /Pass/){
                store_result($site,"GridFTP","Fail",$dbh,$hn_ref,$tn_ref,$rtb_ref,
$rtf_ref,$rts_ref);
                return;
        }

# run a GridFTP test
‘rm -rf /tmp/gridftp.test2‘;
‘echo gridftp.test2 > /tmp/gridftp.test2‘;
exec_timed_command("$globus_path/globus-url-copy -vb
file:///tmp/gridftp.test2 gsiftp://$site/tmp/gridftp.test2","bytes");
# try to gridFTP the other way
‘rm -rf /tmp/gridftp.test2‘;
exec_timed_command("$globus_path/globus-url-copy -vb
gsiftp://$site/tmp/gridftp.test2 file:///tmp/gridftp.test2","bytes");
$result = exec_timed_command("cat /tmp/gridftp.test2","gridftp.test2");
‘rm -rf /tmp/gridftp.test2‘;
store_result($site,"GridFTP",$result,$dbh,$hn_ref,$tn_ref,$rtb_ref,
$rtf_ref,$rts_ref);

}

sub exec_timed_command($$) {
        my $cmd = shift;
my $search = shift;
my $spawn_ok = "Fail";

if ($fancy==1){ print BOLD, "$cmd\n", RESET;}
else { print "$cmd\n";}

        my $exp = Expect->spawn($cmd) or die "Cannot spawn process $!\n";
$exp->restart_timeout_upon_receive(0);
        $exp->expect($timeout,
[
                        $search,
                        sub {
                                $spawn_ok = "Pass";
                                exp_continue;
                        }
                ]
        );
        return $spawn_ok;
}
```

```
##################
## STORE RESULT ##
##################
sub store_result($$$$$$$$$){
my $host = shift;
my $testname = shift;
my $result = shift;
my $dbh = shift;
my $hn_ref = shift;
        my $tn_ref = shift;
my $rtb_ref = shift;
my $rtf_ref = shift;
my $rts_ref = shift;

if ($fancy==1) { print BOLD, RED, "$testname: $result\n", RESET;}
else { print "$testname: $result\n";}

###patch to store results in newgridinfo on gcmon01###

my $host_id;
my $test_type_id;
my $status;
my $test_id;
my @result_array;

#convert host and test name to host_id and test_type_id
for (my $i=1; $i <= (scalar(@$hn_ref)); $i++) {
        foreach (@$hn_ref) {
                if ($host =~ /$hn_ref->[$i - 1]/) {
$host_id = $i;
                }
        }
}

for (my $i=1; $i <= (scalar(@$tn_ref)); $i++) {
        foreach (@$tn_ref) {
                if ($testname =~ /$tn_ref->[$i - 1]/) {
                        $test_type_id = $i;
                }
        }
}

#result conversion (temporary)
if ($result =~ /Pass/){
$status = 1;
}
else {
$status = 0;
}

#use @rtb, @rtf, and @rts arrays to find all result types for matched test type
if ($rtb_ref->[$test_type_id - 1] == 1){
push(@result_array, 'boolean');
```

```
}
if ($rtf_ref->[$test_type_id - 1] == 1){
        push(@result_array, 'float');
}
if ($rts_ref->[$test_type_id - 1] == 1){
        push(@result_array, 'string');
}

#Change tests.recent value from 1 to 0 for the last test_type_id and host_id

$dbh->do("UPDATE tests SET recent = 0 WHERE host_id = '".$host_id."' AND
test_type_id = '".$test_type_id."'")
  or die "Couldn't do statement: " . $dbh->errstr;

#STEP 1: Insert host_id, test_type_id, time, and 'recent'=1 into tests table

$dbh->do("INSERT INTO tests (host_id, test_type_id, time, recent) VALUES
('".$host_id."','".$test_type_id."',now(), 1)")
or die "Couldn't do statement: " . $dbh->errstr;

#STEP 2: Get $test_id from auto_incremented column
my $sth3 = $dbh->prepare("SELECT MAX(test_id) FROM tests")
or die "Couldn't prepare statement: " . $dbh->errstr;
$sth3->execute()
or die "Couldn't execute statement: " . $sth3->errstr;
$test_id = $sth3->fetchrow_array; #one column, so scalar context OK
$sth3->finish;

#STEP 3: for each result type in @result_array increment result counter and
#insert result into correct table

foreach my $type (@result_array) {
$dbh->do("INSERT INTO result_counter (test_result_id) VALUES (NULL)")
or die "Couldn't do statement: " . $dbh->errstr;

$dbh->do("INSERT INTO test_result_".$type." (test_result_id, test_id, result)
VALUES (LAST_INSERT_ID(),'".$test_id."','".$status."')")
or die "Couldn't do statement: " . $dbh->errstr;
}
}


##################
## MAIN PROGRAM ##
##################

#check if lock exists, create a lock if not
my $date = `date`;
if(-e "$current_path/monitor2.lock") { die ("program is locked at $date"); }
else { `touch $current_path/monitor2.lock`; }

# define proxy file
my $proxy = "$current_path/monitor.proxy";
```

```
# check time left
my $cmd = "grid-proxy-info -f $proxy -timeleft";

my $time = ‘$cmd‘;
chomp($time);
if(($time eq "") || ($time==’-1’)){
die "Proxy file expired\n";
}
# use proxy
$ENV{’X509_USER_PROXY’} = $proxy;

# connect to database
my $dbh = DBI->connect(’DBI:mysql:newgridinfo’,’gridmon’,’gr1dm0n’)
or die "Couldn’t connect to database: " . DBI->errstr;


#query database to get host_id, test_type_id, and result_type information

##hosts##
my $sth1 = $dbh->prepare("SELECT host_id, hostname FROM hosts")
        or die "Couldn’t prepare statement: " . $dbh->errstr;
$sth1->execute()
        or die "Couldn’t execute statement: " . $dbh->errstr;

my (@host_array) =();
my $host_hash = {};
while ($host_hash = $sth1->fetchrow_hashref()) {
        push(@host_array, $host_hash);
}
$sth1->finish;

#store hostnames (in order of id#) into array @hn
my @hn = ();
my $hn_ref = \@hn;
for (my $i=1; $i <= ($#host_array + 1); $i++) {
        foreach my $hashref (@host_array) {
                if ($hashref->{host_id} == $i) {
                        $hn[$i - 1] = $hashref->{hostname};
                }
        }
}

##tests##
my $sth2 = $dbh->prepare("SELECT test_type_id, test_name, result_type_boolean,
result_type_float, result_type_string FROM test_type")
        or die "Couldn’t prepare statement: " . $dbh->errstr;
$sth2->execute()
        or die "Couldn’t execute statement: " . $dbh->errstr;

my (@test_array) =();
my $test_hash = {};
while ($test_hash = $sth2->fetchrow_hashref()) {
        push(@test_array, $test_hash);
```

```
}
$sth2->finish;

#store test_name (in order of id#) into array @tn
my @tn = ();
my $tn_ref = \@tn;
for (my $i=1; $i <= ($#test_array + 1); $i++) {
        foreach my $hashref (@test_array) {
                if ($hashref->{test_type_id} == $i) {
                        $tn[$i - 1] = $hashref->{test_name};
                }
        }
}

#store result types (in order of id#) into array @rtb, @rtf, and @rts (boolean,
#float, and string)
my @rtb = ();
my $rtb_ref = \@rtb;
for (my $i=1; $i <= ($#test_array + 1); $i++) {
        foreach my $hashref (@test_array) {
                if ($hashref->{test_type_id} == $i) {
                        $rtb[$i - 1] = $hashref->{result_type_boolean};
                }
        }
}
my @rtf = ();
my $rtf_ref = \@rtf;
for (my $i=1; $i <= ($#test_array + 1); $i++) {
        foreach my $hashref (@test_array) {
                if ($hashref->{test_type_id} == $i) {
                        $rtf[$i - 1] = $hashref->{result_type_float};
                }
        }
}
my @rts = ();
my $rts_ref = \@rts;
for (my $i=1; $i <= ($#test_array + 1); $i++) {
        foreach my $hashref (@test_array) {
                if ($hashref->{test_type_id} == $i) {
                        $rts[$i - 1] = $hashref->{result_type_string};
                }
        }
}

# go through sites in @hn and run tests, but not on GIIS server "grid2.canarie.ca"
foreach my $site(@hn){
if ($site =~ /canarie/){
next;
}
runtests($site,$dbh,$hn_ref,$tn_ref,$rtb_ref,$rtf_ref,$rts_ref);
}

# disconnect
```

```
$dbh->disconnect();
# remove lock
‘rm -rf $current_path/monitor2.lock‘;
```

B ’make_pie.pl’ script

```
#!/usr/bin/perl -w
#make_pie.pl
#Generates pie graphs for each GridX1 resource, which are displayed on
#GridX1 web site.
#
#Also updates ’hosts’ table in the ’newgridinfo’ MySQL database, and
#relevant results are shown in a table next to pie chart.

use GD::Graph::pie;
use Net::FTP;
use DBI;
use Image::Magick;

our $current_path = "/home/gridmon/graphs/pie"; #run directory
our $zope_path = "/gridCanada/gridX1/monitor"; #ftp destination

sub make_chart($$$$$)
{
my $name = shift;
my $Active_CPUs = shift;
my $Grid_Jobs_Running = shift;
my $Local_Jobs_Running = shift;
my $status = shift;
my $Available_CPUs = $Active_CPUs - $Grid_Jobs_Running - $Local_Jobs_Running;
my $label;

if ($status =~ /success/){
$label = "Active CPUs: $Active_CPUs";
}
else{
$label = ’error, no data’;
}

my @data = (
        [ ("$Available_CPUs", "$Local_Jobs_Running", "$Grid_Jobs_Running"  ) ],
        [ ($Available_CPUs, $Local_Jobs_Running, $Grid_Jobs_Running) ]
);

my $my_graph = new GD::Graph::pie( 170, 200 );

$my_graph->set(
label => $label,
         start_angle => 90,

accentclr => ’marine’,
labelclr => ’black’,
dclrs => [qw(marine lgray lgreen)],
```

```
'3d' => 0,

t_margin => 5,
b_margin => 55,
l_margin => 25,
r_margin => 25,

transparent => 0,
suppress_angle => 0,
);
$my_graph->plot(\@data);
save_chart($my_graph, "$current_path/$name"."_pie");
write_legend($name);
}

sub write_legend($) #uses Image:Magick to create legend and datestamp
{
my $name = shift;

my $image = Image::Magick->new;
$image->Read("$current_path/$name"."_pie.gif");
#green
        $image->Draw(primitive=>'Rectangle',stroke=>'black',fill=>'lime',
strokewidth=>0.5,points=>"30,147 45,157");
        $image->Annotate(fill=>'black',geometry=>('+'.(45+5).'+'.(157)),
font=>'Generic.ttf',pointsize=>12,text=>"Grid Jobs");
#gray
$image->Draw(primitive=>'Rectangle',stroke=>'black',fill=>'darkgrey',
strokewidth=>0.5,points=>"30,162 45,172");
$image->Annotate(fill=>'black',geometry=>('+'.(45+5).'+'.(172)),font=>'Generic.ttf',
pointsize=>12,text=>"Local Jobs");
#blue
$image->Draw(primitive=>'Rectangle',stroke=>'black',fill=>'cornflowerblue',
strokewidth=>0.5,points=>"30,177 45,187");
$image->Annotate(fill=>'black',geometry=>('+'.(45+5).'+'.(187)),font=>'Generic.ttf',
pointsize=>12,text=>"Idle CPUs");

my $date = `date`;
if ($date =~ /(.*) PST [0-9]{4}/){
$date = $1;
}
$image->Annotate(fill=>'black',geometry=>('+'.(0).'+'.(200)),style=>'italic',
font=>'Generic.ttf',pointsize=>10,text=>"Generated: $date");

$image->Write("$current_path/$name"."_pie.gif");
}

sub save_chart
{
        my $chart = shift or die "Need a chart!";
        my $name = shift or die "Need a name!";
        local(*OUT);
```

```
        my $ext = $chart->export_format;

        open(OUT, ">$name.$ext") or
                die "Cannot open $name.$ext for write: $!";
        binmode OUT;
        print OUT $chart->gd->$ext();
        close OUT;
}

sub ftp_chart($) {
        # ftp image to Zope on yamon
my $name = shift;
my $date = ‘date‘;
        my $ftp = Net::FTP->new("yamon.phys.uvic.ca", Port => 8021, Passive => 1)
                or die "Cannot connect to yamon.phys.uvic.ca: $@";
        $ftp->login("qmatthew",’qmatthew!314’)
                or die "Cannot login at $date: ", $ftp->message;
        $ftp->binary;
        $ftp->put("$current_path/$name"."_pie.gif","$zope_path/"."$name"."/
$name"."_pie.gif")
                or die "get failed ", $ftp->message;
        $ftp->quit;
}

sub update_db($$$$$$$$) {
#updates the hosts table in mysql database ’newgridinfo’
my $site = shift;
my $TotalCPUs = shift;
my $ActiveCPUs = shift;
my $NumGridCPUs = shift;
my $TotalJobs = shift;
my $NumGridJobs = shift;
my $NumLocalJobs = shift;
my $dbh = shift;

$dbh->do("UPDATE hosts SET TotalCPUs = ".$TotalCPUs.", ActiveCPUs = ".$ActiveCPUs.",
NumGridCPUs = ".$NumGridCPUs.", TotalJobs = ".$TotalJobs.", NumGridJobs =
".$NumGridJobs.", NumLocalJobs = ".$NumLocalJobs." WHERE hostname = ’".$site."’")
         or die "Couldn’t do statement: " . $dbh->errstr;
}

##################
###MAIN PROGRAM###
##################

#check if lock exists, create a lock if not
if(-e "$current_path/make_pie.lock") { die ("program is locked"); }
else { ‘touch $current_path/make_pie.lock‘; }

# define proxy file
my $proxy = "/home/gridmon/programs/monitor/monitor.proxy";
# check time left
my $cmd = "grid-proxy-info -f $proxy -timeleft";
```

```
my $time = ‘$cmd‘;
chomp($time);
if(($time eq "") || ($time==’-1’)){
        die "Proxy file expired\n";
}
# use proxy
$ENV{’X509_USER_PROXY’} = $proxy;

#set paths to local executables
my $globus_path = "/opt/vdt/globus/bin";
my $condor_path = "/opt/vdt/condor/bin";

my @names = ("mercury.uvic.ca", "thuner-gw.phys.ualberta.ca",
"mercury.sao.nrc.ca");

#fix paths to remote executables, qstat and pbsnodes
my %path = (
        ’mercury.uvic.ca’=>"/usr/local/pbs/bin",
        ’thuner-gw.phys.ualberta.ca’=>"/usr/local/bin",
        ’mercury.sao.nrc.ca’=>"/usr/pbs/bin"
);

#fix number of CPUs at each resource that are able to run Grid jobs
my %Number_Grid_CPUs = (
’mercury.uvic.ca’=>100,
’thuner-gw.phys.ualberta.ca’=>32,
’mercury.sao.nrc.ca’=>24
);

#connect to database
my $dbh = DBI->connect(’DBI:mysql:newgridinfo’,’gridmon’,’gr1dm0n’)
        or die "Couldn’t connect to database: " . DBI->errstr;

#print newline and date for debugging info
my $date = ‘date‘;
print "\n".$date;

foreach my $name (@names){
#update database, make and ftp charts, and then reset values
#define/init values
my $Number_Grid_CPUs = $Number_Grid_CPUs{$name};
my $path = $path{$name};

my $Total_Jobs_Running = 0;
my $Grid_Jobs_Running = 0;
my $Local_Jobs_Running = 0;
my $Total_CPUs;
my $Inactive_Nodes = 0;
my $Inactive_CPUs = 0;
my $Active_CPUs;

#check running jobs
```

```
print "Checking qstat on $name...\n";
open (QSTAT, "$globus_path/globus-job-run $name/jobmanager-fork
$path/qstat | grep R|");
while (my $entry = <QSTAT>){
if ($entry =~ /gcprod/){
$Grid_Jobs_Running += 1;
$Total_Jobs_Running += 1;
}
else{
$Local_Jobs_Running += 1;
$Total_Jobs_Running += 1;
}
}
close QSTAT;

#check CPUs
print "Checking pbsnodes on $name...\n";
open (PBSNODES, "$globus_path/globus-job-run $name/jobmanager-fork
$path/pbsnodes -a|");
while (my $line = <PBSNODES>){
if (($line =~ /np = ([0-9])$/)||($line =~ /pcpus = ([0-9])$/)){
$Total_CPUs += $1;
}
if (($line =~ /state = (.*)offline(.*)/)||($line =~ /state =
(.*)down(.*)/)||($line =~ /state = (.*)unknown(.*)/)){
$Inactive_Nodes += 1;
}
}

if ($Total_CPUs =~ /([0-9]+)$/){
$Inactive_CPUs = ($Inactive_Nodes)*2;
$Active_CPUs = $Total_CPUs - $Inactive_CPUs;
update_db($name,$Total_CPUs,$Active_CPUs,$Number_Grid_CPUs,
$Total_Jobs_Running,$Grid_Jobs_Running,$Local_Jobs_Running,$dbh);
make_chart($name,$Active_CPUs,$Grid_Jobs_Running,$Local_Jobs_Running,
'success');
ftp_chart($name);
}
else {
print "error\n";
update_db($name,0,0,0,0,0,0,$dbh);
make_chart($name,0,0,0, 'error');
ftp_chart($name);
}
}

#disconnect from database
$dbh->disconnect();
# remove lock
`rm -rf $current_path/make_pie.lock`;
```

References

[1] W.Tomlin, ”LHC Computing Grid Project”, [Online Document], Available at http://lcg.web.cern.ch/LCG/.

[2] UVic Grid Research team, ”What is Grid Computing?”, [Online Document], Available at http://www.grid.phys.uvic.ca.

[3] Rodney Walker et al, ”Federating Grids: LCG Meets Canadian HEPGrid”, [Online Document], Available at http://www.grid.uvic.ca/documentation/papers/CHEP04-paper-final.pdf.