Work Term Report

Marco Yuen <marcoy@csc.uvic.ca>

Table of Contents

1. Overview

I will give a brief overview of how the monitoring system works. The whole monitoring infrastructure is comprised of numerous scripts and a relational database running locally on `gcmon01`. One of the scripts, `monitor2.pl`, is responsible for testing the availability of different services on various clusters and store the results back into the local database. Other scripts will query the local database and construct human-readable outputs. The outputs can be seen on the GridX1's website. However, not all of the scripts query the local database at all. Some scripts query a database at CERN, some query a database at TRIUMF and some don't even query any database. Most of the scripts including `monitor2.pl` are run periodically by a crontab. I will go over each script in turn, and followed by multiple howtos on maintaining the website and the monitoring scripts.

2. Monitoring Scripts

Almost the entire collection of the monitoring scripts are resided in `/home/gridmon` on `gcmon01`. The database that has the results of the tests is running on `gcmon01` as well. In some of the subdirectories in `/home/gridmon` there is a folder called `sandbox`; it is where I do all the testing before rolling out the changes by generating a patch and applying the patch to the production scripts. I strongly advise that changes are made to the scripts in `sandbox` first then to the production scripts.

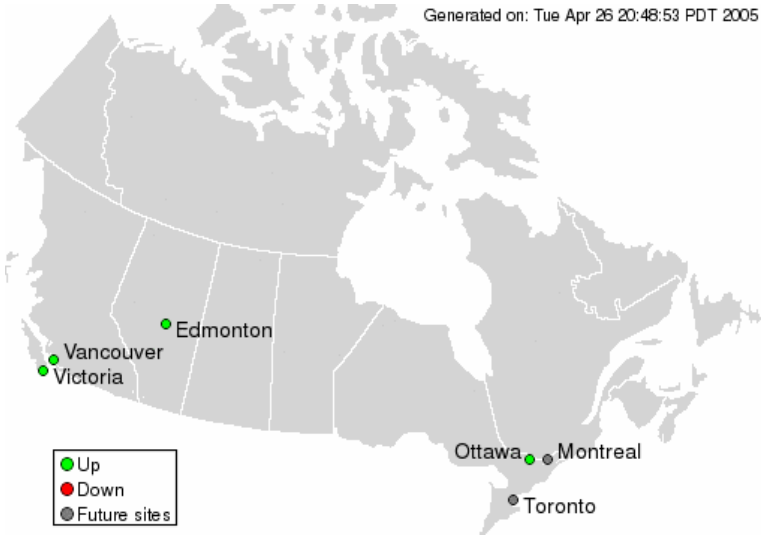
2.1. Scripts in `/home/gridmon/programs`

There are two subdirectories inside `/home/gridmon/programs`, *makemap* and *monitor*. I will explain what each of those scripts does in the following subsections.

2.1.1. Makemap

`make_map.pl` queries the local database and creates a map with color-coded dots on it. The map can be seen on the main page of GridX1 [http://www.gridx1.ca]. Due to the fact that some sites are located in close proximity of each other, their labels are drawn by manually specifying the coordinates. Sites such as Victoria, Alberta, and Ottawa have multiple clusters but the map only shows one of the clusters for each site. This problem will probably be solved in the future.

Figure 1. Map from GridX1 [http://www.gridx1.ca]



2.1.2. Monitor

`/home/gridmon/programs/monitor` houses one of the most important scripts in the monitoring infrastructure, `monitor2.pl`. Its main task is to connect to the clusters and determine whether the services they are supposed to provide are, indeed, up and running. The services that `monitor2.pl` monitors are Gatekeeper, Job Submission, and GridFTP. The results are pushed back to a database named newgridinfo. More details on newgridinfo in the next section. A few scripts are relying on the results gathered by `monitor2.pl` in order to function properly. There is a `sandbox` subdirectory in the `/home/gridmon/programs/monitor`. The script inside `sandbox` does exactly the same thing as its production counterpart except that the results are stored in a database named test instead of newgridinfo. Both databases have the same schema but one, as the name implies, is for testing purposes. `monitor2.pl` runs humbly every four minutes churning out the results back to newgridinfo.

2.1.3. Database: newgridinfo

The only table that is important in newgridinfo is hosts. It contains all the information related to a cluster. This table plays a major role when there is a new site that needs to be monitored. In the later section, I will demonstrate how to add a new site by using the hosts table. Other tables in newgridinfo are for storing the test results in different formats. Scripts compute the results by joining one of the result tables with the tests and hosts tables.

2.2. Scripts in `/home/gridmon/graphs`

There are quite a few scripts inside `/home/gridmon/graphs`. As you might have already guessed, the main purpose of these scripts is to create beautiful and informative graphs that represent current or past states of the clusters.

2.2.1. 24h, 7d, and 30d

These three scripts are becoming obsolete. They were used record the number of successful and failed jobs in past twenty-four hours, past seven days and past thirty days respectively. The graphs that the scripts generated would be uploaded to the GridX1's website. Even though these scripts are still being run by the crontab, they are not being displayed on the webpage any more.

2.2.2. Bar

Yet another script that is being phased out. The bar chart created by `make_bar.pl` reflects the number of the jobs being run up to four minutes ago on the clusters. The fact that `make_bar.pl` generates a static JPEG every four minutes creates an inconsistency on the webpage; therefore, it is replaced by a more dynamic script. The new script can dynamically update the graph every thirty seconds without requiring the clients to refresh the webpage. If the server can handle greater load, the update time can be shortened even more.

2.2.3. Logger

After going through all of the obsolete scripts, I can now move on to the first production script in `/home/gridmon/graphs`. `make_logger.pl` has two outputs; one shows a histogram of finished jobs for the past year, and the other shows a histogram of both finished and failed jobs for the past year. The former can be seen on the monitoring webpage [http://www.gridx1.ca/monitor/atlas_logger]. The second output requires a little manual fiddling with the URL; the graph can be seen by appending *more* to the URL [http://www.gridx1.ca/monitor/atlas_logger/more].

`make_logger.pl` gets all the data by querying a database at CERN and it uses GD::Graph to create two graphs named `gridX1_logger.gif`, and `gridX1_logger_all.gif`.

2.2.4. LoggerLast30d

The second script that is currently in production is `makeLogger30d.pl`. Its output is similar to `make_logger.pl`. However, instead of getting all the successful and failed jobs for the past year, `makeLogger30d.pl` only gets the successful jobs for the past thirty days. As with `make_logger.pl`, you can view the output of `makeLogger30d.pl` from the GridX1's webpage.

`makeLogger30d.pl` is an incarnation of `make_logger.pl`. The only two thing that are different in terms of the implementation are the SQL query used and the parsing algorithm. Other than those two differences, they are identical.

2.2.5. Pie

Finally, `make_pie.pl` is the last script in `/home/gridmon/graphs`. `make_pie.pl` is a more site specific script. Unlike `make_logger.pl` and `makeLogger30d.pl` which treat all the clusters as a whole, `make_pie.pl` treats individual site as a data source and queries each site for information. For each site, `make_pie.pl` gathers the number of running jobs, number of CPUs, and number of idle CPUs. After collecting all the required data it will create a pie chart, and it will update the corresponding columns in hosts table of newgridinfo. All the outputs can be seen on the GridX1's website under monitor.

It acquires the hostnames of each site by querying the hosts table from newgridinfo. Once the hostname is known, `make_pie.pl` will use **globus-job-run** to remotely execute two other commands **qstat**, and **pbsnodes** on the designated site.

`make_pie.pl` is known to stall when the site being queried is down. The problem persists mainly because there isn't any timeout mechanism for **globus-job-run**. Therefore, `make_pie.pl` should implement a timeout whenever it tries to execute **globus-job-run**.

2.3. Other scripts

There are a small number of scripts that are not in `/home/gridmon`. Most of them are located inside Zope. Since Zope has its own filesystem or rather its own method of storing files, there is no way to gain direct access to the files inside. Luckily, Zope provides two ways to access the objects(files) stored within. One way is to access the files via FTP, and the other way is to use the Zope management interface.

2.3.1. XmlBar

This script, `xmlBar.py`, can be found in `/var/www/html/python` on `yamon`. In a nutshell, this script works exactly the same as `make_pie.pl`, but this is more optimized, and written in Python. The output of `xmlBar.py` is XML, and the content will be parsed and displayed by another script.

Inside `xmlBar.py`, the function, `index`, is the heart of this script, because it is *the* function that does all the work. First, `index` creates two child processes. Then it parses the output of the children. Finally, it returns the data in XML format. The two child processes each runs two commands, **condor_status** and **condor_q**. `xmlBar.py` gets the hostnames for each sites by using **condor_status**. With the hostnames stored in the memory, `xmlBar.py` uses them as parameters for **condor_q**. Then, it starts parsing the output of **condor_q**, and formats the output into a valid and well-formed XML document.

The actual XML data can be viewed at http://www.gridx1.ca/~python/xmlBar.py. A web browser such as Mozilla Suite, or FireFox should be used because they have built-in XML viewing and validating capabilities.

2.3.2. Portlet: Jobs Barchart

With the data from `xmlBar.py`, there should be a way to properly display those XML data. This is

where `portlet_jobs_barchart` comes into play. Unlike other scripts `portlet_jobs_barchart` doesn't reside on any local filesystems but inside Zope. `portlet_jobs_barchart` is located at `/gridX1/portal_skins/custom`[1] inside the Zope filesystem. Its main task is to create the bar chart on the top right of the GridX1 website based on the data gathered by `xmlBar.py`.

`portlet_jobs_barchart` is mostly written in javascript. It constructs a DOM tree on the fly, and the browser will render that DOM tree and create a bar chart. Due to the fact that the DOM tree is created on the fly, it can be updated automatically without users intervention. In other words, users don't have to refresh the webpage to get the latest information; they only need to leave the webpage on the screen and the bar chart will update itself.

One drawback for `portlet_jobs_barchart`, I think, is the complexity. I manually construct a DOM tree with <table> as the root element. While this method gives me the power to do whatever I see fit, but it stifles extensibility and increases complexity. Adding a column to the table or adding a <td> element to the DOM tree means a lot of manual fiddling with the javascript code. A better way to do this is to use XSLT and transform the XML from `xmlBar.py` to SVG. One problem it poses is that not all browsers out in the market have built-in SVG support. However, in a long run, XSLT can provide a cleaner and more extensible solution.

2.3.3. Portlet: Create Site Status

Another script that resides in the Zope filesystem is `create_site_status`. The result of this script can be seen on the bottom right of the GridX1's website; it shows the current status each clusters, and it is located at `/gridX1` on the Zope filesystem.

`create_site_status` queries the database for the status of each cluster. Based on the result of the query, `create_site_status` assigns different colors for each cluster. With *Red* being at least one of the services is down for that cluster, and *Green* being all of the services for that cluster are working as expected.

2.4. Subversion

All of the scripts aforementioned are under revision by Subversion. However, not all of the scripts in `/home/gridmon` are under revision , and even the scripts that are under revision, they were only added to Subversion recently. So, the full history of the scripts is not recorded in the repository. The commands for Subversion are very similar to those in CVS with some extensions, and Subversion is more capable than CVS in terms of functionality, security, and availability. Subversion set out to be a replacement of CVS, and I think they have already exceeded their goal.

I will give a quick rundown on how to use Subversion. A Subversion command often has form **`svn <svn-command>`**. To find a list of available Subversion command, one can type **`svn help`** in the terminal. Or, to get help for a specific command, use **`svn help <svn-command>`**. Common commands such as *checkout*, *update*, *diff*, and *commit* have the same meaning as their counterpart in CVS.

3. GridX1 Website

Most of the scripts mentioned in Section 2, “Monitoring Scripts” upload their results to GridX1 [http://www.gridx1.ca]. The website users Plone as the back-end, so content can be updated or added with ease. Internally, Plone relies on Zope as the application server. Both Zope, and Plone are written entirely in Python.

3.1. Accessing Zope

[1] Information about accessing Zope can be found at Section 3.1, “Accessing Zope”.

As I mentioned before in Section 2.3, “Other scripts”, there are two main ways to gain access to the Zope filesystem. First, one can access the filesystem via the Zope's management interface. Second, one can FTP in the Zope. Actually, there is another way to do it but that requires some knowledge of ZODB and Python.

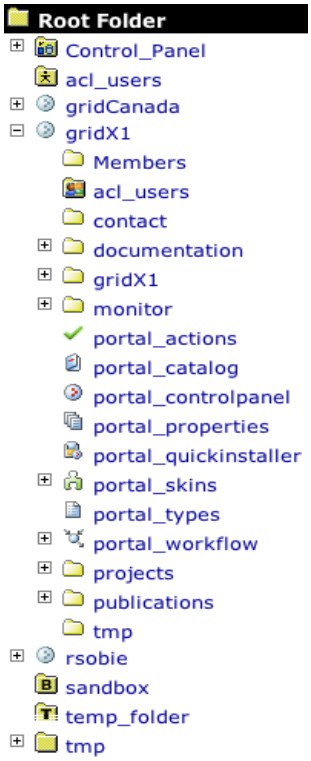
3.1.1. Accessing Zope: Management Interface

The Zope management interface is the *de facto* way to access Zope's filesystem. To access the interface:

Procedure 1. Accessing through Web browser

1. Fire up your favorite web browser.
2. Type *http://yamon.phys.uvic.ca:8080/manage* in the location bar.
3. Type in your username and password.
4. You are in!

Once inside the management interface, the left side is the navigation frame.



The navigation frame allows users to traverse the entire Zope file structure in an intuitive and graphical manner. The management interface provides a set of comprehensive operations for users to operate on the content inside. Users can *add*, *move*, *copy*, *delete*, and etc... The entire GridX1 website is under `/gridX1`. The management interface can be quite inconvenient when operating on large number of files or doing multiple operations on the same file. Nonetheless, it is an indispensable tool.

3.1.2. Accessing Zope: FTP

Another way to access the content of Zope is to use FTP. Most of the scripts in Section 2, “Monitoring Scripts”, use FTP to upload the images to the website.

Procedure 2. Accessing through FTP

1. Fire up your favorite FTP client.
2. Connect to *http://yamon.phys.uvic.ca:8021*.
3. Type in your username and password

> **Warning**
>
> Zope makes heavy use of metadata to distinguish types of files or Zope Objects in Zope parlance. FTP access does *NOT* provide the ability to upload or download metadata that belong to a Zope Object. So files uploaded to zope via FTP should only restrict to files with well-known extensions (eg. GIF, JPEG, HTML, XHTML, XML, etc...)

3.1.3. Accessing Zope: Home Brew Solutions

The third way to accessing the Zope filesystem is to create your own program and talk directly to ZODB. Zope uses ZODB exclusively to store all the data. ZODB is an object oriented database and, similar to Zope, it is written in Python. ZODB has its own API, and Python is an easy, clean, and powerful language (WAY better than using perl).

3.2. The Anatomy of the GridX1 website

The directory structure of the `/gridX1` may look intimidating at first. However, only a handful of directories are important. First, there is `/gridX1/portal_skins/custom`. That is the place where all the custom CSS, portlets, and templates go. Second, `/gridX1/portal_actions` is where the tabs and subtabs are defined. Finally, `/gridX1/monitor` contains all of the images and HTML pages related to the monitoring section.

4. HOWTOs

After introducing most of the scripts related to the monitoring infrastructure, I will move on to introducing some of the common operations required for maintaining the scripts and the website. Generally, the whole monitoring infrastructure requires little maintenance, but occasionally something unexpected may emerge.

In this collection of HOWTOs you will learn:

- How To Add A New Site
- How To Add A Pie Chart
- How To Display A Pie Chart
- How To Abbreviate Site Names
- How To Add New Tabs

- How To Setup A New Page For A Condor Queue

4.1. HOWTO: Add A New Site

In Figure 1, “Map from GridX1”, two sites are labeled as future sites - Montreal and Toronto. There may be a need to include them as part of the monitoring infrastructure. Adding a new site to the infrastructure is not as trivial as it sounds, as it requires multiple prerequisites in place. The first thing to do is to make sure their required services are up and running. The monitoring scripts will need to run some commands on the remote site; therefore, a running GateKeeper, Broker, and GridFTP server is essential in order for the scripts to work properly. Then, make sure Dan's subject is on the remote site's gridmap file; that will give the scripts privileges to run jobs on the remote site, since all of the jobs are running with Dan as the owner. With the services up and running and Dan's subject in place, the remote host's information needs to be inserted into the hosts table of newgridinfo. A close look at the schema for hosts; there are ten fields in the table, and that is all the information required.

```
mysql> describe hosts;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| host_id      | int(11)     |      | PRI | 0       |       |
| hostname     | varchar(50) | YES  |     | NULL    |       |
| admin        | varchar(50) | YES  |     | NULL    |       |
| ip           | varchar(20) | YES  |     | NULL    |       |
| TotalCPUs    | int(11)     |      |     | 0       |       |
| ActiveCPUs   | int(11)     |      |     | 0       |       |
| NumGridCPUs  | int(11)     |      |     | 0       |       |
| TotalJobs    | int(11)     |      |     | 0       |       |
| NumGridJobs  | int(11)     |      |     | 0       |       |
| NumLocalJobs | int(11)     |      |     | 0       |       |
+--------------+-------------+------+-----+---------+-------+
10 rows in set (0.02 sec)
```

Even though there are ten fields, a couple of the fields are especially important, and their values *must* be correct. First, host_id *must* be unique and of the value MAX(host_id) + 1. Second, the hostname *must* be a valid hostname, because some of the scripts mentioned in Section 2, “Monitoring Scripts” query specifically for this field. If for some reason, this field is incorrect, the scripts will not gather any statistics for the designated site. Finally, the ip field of hosts is the corresponding ip address of the hostname field. The ip address can simply be acquired by typing **`host <hostname>`** in the terminal. The information for the remaining fields can be found by inquiring the administrator of the remote site, or by reading the ClassAd. Now that everything is in place, a quick test with `monitor2.pl` should indicate whether the site is successfully added or not.

Note

There is usually a wrapper script associated with each monitoring script. Remember to run the wrapper script instead, because it sets environment variables that are needed for the monitoring script. The wrapper scripts are written in BASH, and they usually just execute the actual monitoring script after setting the needed environment variables.

If it is successful, the output of `monitor2.pl` should look something like the following with the hostname replaced with the newly added hostname.

```
--venus.sao.nrc.ca--
/opt/vdt/globus/bin/globusrun -a -r venus.sao.nrc.ca
```

```
GRAM Authentication test successful
Gatekeeper: Pass
/opt/vdt/globus/bin/globusrun -o -r venus.sao.nrc.ca/jobmanager-fork \
  "&(executable=/bin/echo)(arguments=jobtest)" jobtest
Job Submission: Pass
/opt/vdt/globus/bin/globus-url-copy -vb \
  file:///tmp/gridftp.test2 \
  gsiftp://venus.sao.nrc.ca/tmp/gridftp.test2
14 bytes         0.01 KB/sec avg         0.01 KB/sec inst
/opt/vdt/globus/bin/globus-url-copy -vb \
  gsiftp://venus.sao.nrc.ca/tmp/gridftp.test2 \
  file:///tmp/gridftp.test2

cat /tmp/gridftp.test2
gridftp.test2
GridFTP: Pass
```

4.2. HOWTO: Add A Pie Chart

In Section 2.2.5, “Pie”, `make_pie.pl` generates a site specific pie chart. However, in order for it to generate a pie chart with the current states of the new site, it has to know the path, on the *remote* site, of the two executables **qstat** and **pbsnodes**. To find out the path requires access to the remote site, usually Askhok or Dan will have access to the new site. The script needs to know the new information, so use your favorite editor and open up `make_pie.pl`. Look for the follow line.

```
my @names = ("mercury.uvic.ca", "thuner-gw.phys.ualberta.ca",
"mercury.sao.nrc.ca", "calliope.phys.uvic.ca", "thuner-grid.nic.ualberta.ca",
"hep.westgrid.ca", "venus.sao.nrc.ca");
```

Once it is located, add the hostname of the new site to `@names` array. After that, scroll towards the end until you reach:

```
my %qstatpath = (
  'mercury.uvic.ca'=>"/usr/local/pbs/bin",
  'thuner-gw.phys.ualberta.ca'=>"/usr/local/bin",
  'mercury.sao.nrc.ca'=>"/usr/pbs/bin",
  'calliope.phys.uvic.ca'=>"/usr/pbs/bin",
  'thuner-grid.nic.ualberta.ca'=>"/opt/local/bin",
  'hep.westgrid.ca'=>"/opt/globus/lib/perl/Globus/GRAM/JobManager",
  'venus.sao.nrc.ca'=>"/usr/pbs/bin");
my %pbsnodespath = (
  'mercury.uvic.ca'=>"/usr/local/pbs/bin",
  'thuner-gw.phys.ualberta.ca'=>"/usr/local/bin",
  'mercury.sao.nrc.ca'=>"/usr/pbs/bin",
  'calliope.phys.uvic.ca'=>"/usr/pbs/bin",
  'thuner-grid.nic.ualberta.ca'=>"/opt/local/bin",
  'hep.westgrid.ca'=>"/global/software/torque-1.2.0p2/bin",
  'venus.sao.nrc.ca'=>"/usr/pbs/bin");
my %Number_Grid_CPUs = (
  'mercury.uvic.ca'=>100,
  'thuner-gw.phys.ualberta.ca'=>32,
  'mercury.sao.nrc.ca'=>24,
  'calliope.phys.uvic.ca'=>50,
  'thuner-grid.nic.ualberta.ca'=>32,
  'hep.westgrid.ca'=>998,
  'venus.sao.nrc.ca'=>68);
```

Add the paths to of **qstat**, and **pbsnodes** to `%qstatpath`, and `%pbsnodepath` respectively. Finally, add the number of the CPUs to `%Number_Grid_CPUs`.

It's always a good idea to make sure `make_pie.pl` is still working. Excute the wrapper script and see if the output looks like the following:

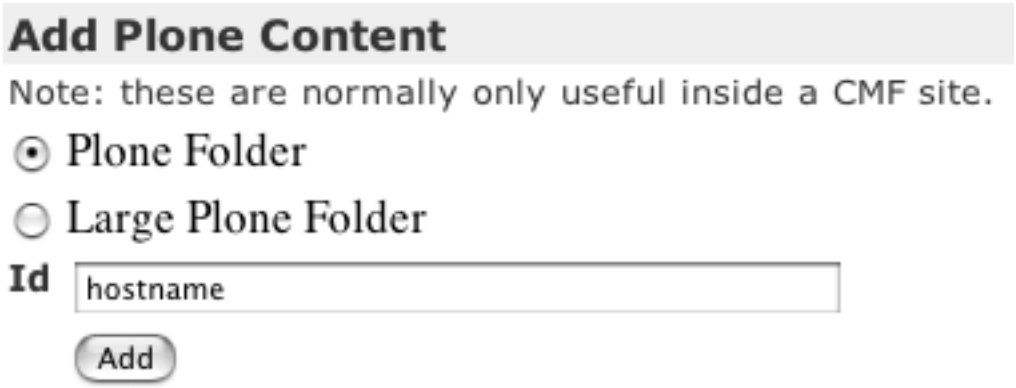
```
Checking qstat on mercury.uvic.ca...
Checking pbsnodes on mercury.uvic.ca...
success, 9 Jobs, 8 Local, 1 Grid, 110 Active CPUs, 112 Total CPUs
```

Again, the hostname should be the hostname of the new site instead of `mercury.uvic.ca`.

4.3. HOWTO: Display A Pie Chart

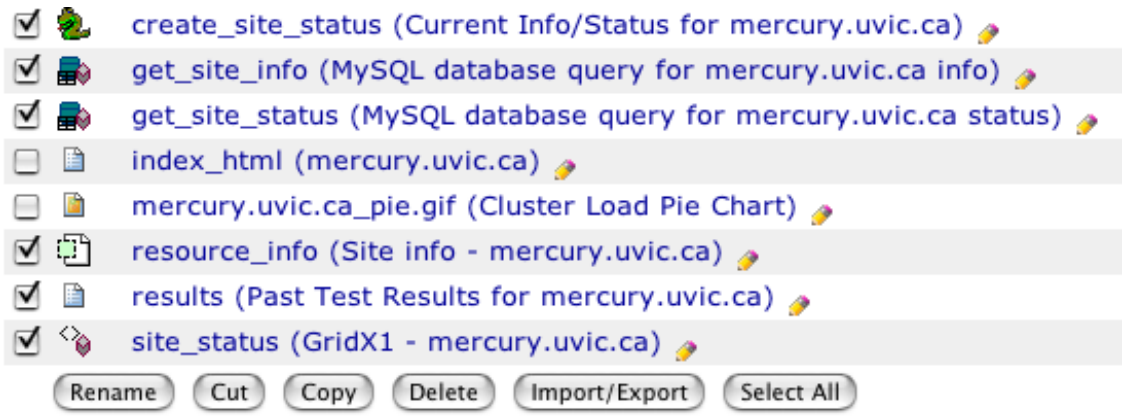
A new pie chart is generated by `make_pie.pl`, but it is nowhere to be found! So, we need to add something the GridX1's website such that there is a link to the new pie chart. First, connect to the Zope management interface[2], then browse to `/gridX1/monitor`. Inside `/gridX1/monitor`, there are multiple directories with names same as the hostnames. As you may expect, we need to add a new directory with a name same as the new site's hostname. To add a new directory, choose "Plone Content" from the drop list on the right.

A new page will come and this time, type in the hostname in the textbox



A newly created directory should be present in `/gridX1/monitor`. Now, we need to copy some files from one of the existing host directories to the newly created one. Pick an existing directory, and check the following six files.

[2]Steps to connect to Zope management interface can be found in Section 3.1.1, “Accessing Zope: Management Interface”



Then, click "Copy" on the bottom. After that, go back to `/gridX1/monitor` (the parent), and click on the new directory, then click paste located at the bottom. The required files are now copied over to the new directory, however, the hostname inside the files are hard-coded. So we need to change the hostname for each of the file. Please refer to the Zope manual on how to edit files inside the management interface.

4.4. HOWTO: Abbreviate Site Names

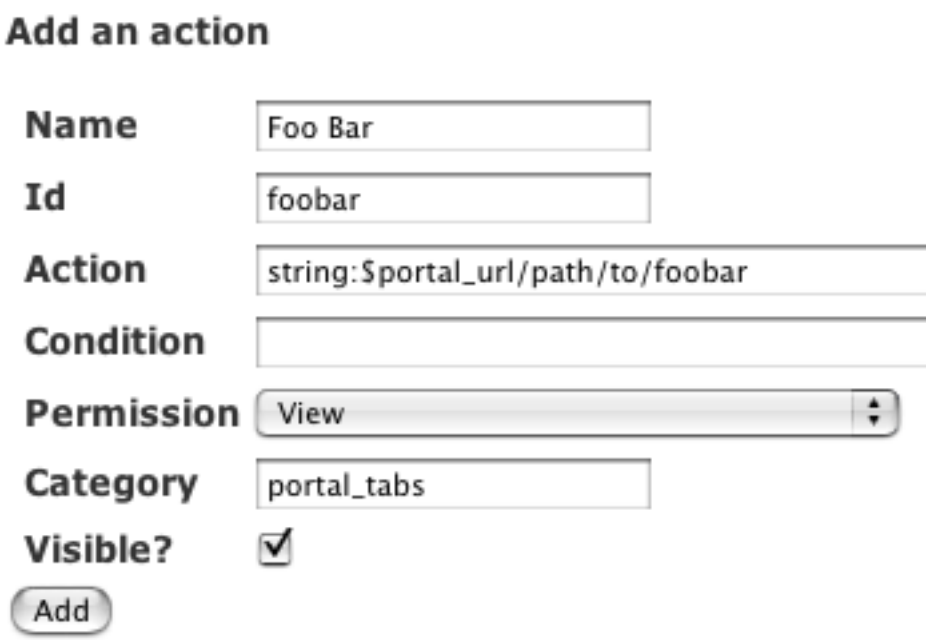
The names show up on the portlets are being shortened. If a new site shows, those portlets do not know how to shorten them. For `create_site_status`, if there is a new site, `create_site_status` will simply display the hostname of that new site, and for `portlet_jobs_barchart`, it will display *undefined*. Fortunately, this is a simple problem to fix. Since both of these scripts are inside the Zope filesystem, the first thing to do to fix the problem is to go the Zope management interface. The fix for `create_site_status` is to add the hostname to the `sitesMap` dictionary.

```
sitesMap = {}
sitesMap['mercury.sao.nrc.ca'] = 'NRC-Mercury'
sitesMap['mercury.uvic.ca'] = 'Victoria-Mercury'
sitesMap['thuner-gw.phys.ualberta.ca'] = 'Alberta-Thor'
sitesMap['thuner-grid.nic.ualberta.ca'] = 'Alberta-Thor-grid'
sitesMap['calliope.phys.uvic.ca'] = 'Victoria-Muse'
sitesMap['hep.westgrid.ca'] = 'Vancouver-Westgrid'
sitesMap['venus.sao.nrc.ca'] = 'NRC-Venus'
```

The same solution applies to `portlet_jobs_barchart` as well. Simply add the new hostname to the `sitesMap` dictionary, but this time it's in JavaScript.

4.5. HOWTO: Add New Tabs

If a new section is needed to onto the webpage, creating a new tab for that section is the best way to go. To add new tabs to website, first go to the Zope management interface. Once inside the management interface, go to `/gridX1/portal_actions`. Scroll all the way to the end of the page and there are a couple textboxes, simply fill in the required info like the following.



Click “Save” and that should create a new tab on the webpage. However, if you want to create a subtab, you need to fill in the same information except for Category. Instead of putting *portal_tabs* in Category, you need to put *portaltab-<parentid>*, and it will create a subtab underneath *parent*.

4.6. HOWTO: Setup A New Page For A Condor Queue

In order to create a page that displays jobs currently queued in a condor queue, you need to go the `/opt/Zope-2.7/Extensions` on `yamon`. Inside `/opt/Zope-2.7/Extensions` there is a file called `condorg_q.pl`. Simply open it with you favorite editor and find this line.

```
open (CONDORQ, "/opt/condor-6.6.6/bin/condor_q -name lcgce02.triumf.ca
-pool lcgce02.triumf.ca -l|");
```

Change the arguments for `-name` and `-pool` to the new hostname. The most important step is finished, the only thing left to do is to duplicate some files under `/gridX1/monitor`, and add a new subtab. I will leave that as an exercise.

5. Suggestion

My first suggestion is to *STOP* using perl. There are other alternatives out there the can do exactly same thing, if not better, but without the arcane syntax and semantics. This section mostly consists of quotes from people who share the same feelings with me towards Perl.

> Frankly, I'd rather not try to compete with Perl in the areas where Perl is best -- it's a battle that's impossible to win, and I don't think it is a good idea to strive for the number of obscure options and shortcuts that Perl has acquired through the years.
>
> — Guido van Rossum

> [Perl] combines all the worst aspects of C and Lisp: a billion different sublanguages in one monolithic executable. It combines the power of C with the readability of PostScript.
>
> —Jamie Zawinski

It's not that perl programmers are idiots, it's that the language rewards idiotic behavior in a way that no other language or tool has ever done.

—Erik Naggum

The ultimate laziness is not using Perl. That saves you so much work you wouldn't believe it if you had never tried it.

—Erik Naggum

...but I guess there are some things that are so gross you just have to forget, or it'll destroy something within you. perl is the first such thing I have known.

—Erik Naggum

Like I said, it’s a pop culture. A commercial hit record for teenagers doesn’t have to have any particular musical merits. I think a lot of the success of various programming languages is expeditious gap-filling. Perl is another example of filling a tiny, short-term need, and then being a real problem in the longer term.

—Alan Kay

Perls OO is like MacDonalds food. Fast, cheap, and great if you aren't used to anything else. Kids love it. But utterly disgusting for lovers of gourmet food.

—Abigail (Slashdot Comment)

Perl: The only language that looks the same before and after RSA encryption.

—Slashdot Comment

If you want to shoot yourself in the foot, Perl will give you ten bullets and a laser scope, then stand by and cheer you on.

—Teodor Zlatanov

Perl is like vise grips. You can do anything with it but it is the wrong tool for every job.

—Bruce Eckel

[Perl] is the sanctuary of dunces. The godsend for brainless coders. The means and banner of sysadmins. The lingua franca of trial-and-error hackers. The song and dance of stultified engineers.

—Xah Lee

Perl is worse than Python because people wanted it worse.

—Larry Wall

Index