University of Victoria
Faculty of Engineering
Department of Computer Science
Summer 2008 Work Term Report

Condor Web Services Submission of Jobs to a Computational Grid

Department of Physics and Astronomy
University of Victoria
Victoria, BC

Sean Manning
0330705
Work Term 2
Computer Science/Mathematics
seangwm@uvic.ca

3 September 2008

In partial fulfillment of the requirements of a Bachelor of Science degree

Supervisor's Approval: To Be Completed by the Co-Op Employer

I approve the release of this report to the University of Victoria for evaluation purposes only.
This report is to be considered (**pick one**): □ Confidential □ Not Confidential

Signature: _____ Position: _____Date: _____
Name (print): _____Email: _____ Fax #: _____

If the report is deemed confidential, a non-disclosure form signed by an evaluator will be faxed to the employer. This report will be destroyed following evaluation. If the report is not confidential, it will be returned to the student following evaluation.

Abstract

Grid computing is an increasingly popular solution to the growing demand of scientists for computing power. Adding a Web Services interface to an existing grid (which uses Globus, Condor, PBS, and Xen) offers several benefits. For example, it makes security easier to handle, and it is easier to use than the full command-line interface. A simple interface for submitting and monitoring was created for the HEPnet grid at UVic using the Java programming language and the Birdbath and Condor APIs. A number of obstacles were met and overcome, including poor documentation and the difficulties of adding new software to a production system. The interface is successful, but it could still be improved and expanded.

Report Specifications

- **Audience:** Someone considering working with Condor Web Services in a Globus grid environment.
- **Prerequisites:** The reader should have a basic familiarity with Unix concepts and with Java or a similar programming language.
- **Purpose:** This report documents some of my work in summer 2008. It might be of interest to other people working on the same or similar projects.

Table of Contents

Introduction

Grid computing is a popular solution to the growing demand of scientists for computing power. Rather than each project maintaining and managing enough computers to meet its needs, the researchers can join or gain access to a network called a grid. Grid computing is especially useful for tasks which can be done in parallel on several different machines, because individual machines in the grid are usually not especially powerful. A grid can be expanded easily, unlike a

single powerful computer. Some grids use groups of dedicated machines, others (like SETI@Home) are installed on machines with some other purpose and run when they are idle. It is hoped that, one day, computational capacity could be distributed as easily as water or electricity.[1]

Implementing a grid faces many technical challenges. One of the humbler ones is designing an interface which is as versatile and easy to set up as possible. A new approach using Web Services technology offered several benefits in these areas. In fall 2007 and summer 2008, a simple Web Services interface was developed which would let users submit jobs, retrieve output, and monitor the status of their jobs. This interface was completed at the end of August 2008, although it could easily be expanded further.

The Web Services interface is centered around three Java classes, CondorJobSubmitter.java, CondorJobStatus.java, and JobHelper.java. These respectively are used for submitting jobs, monitoring jobs and retrieving output, and providing support. They were written by David Gong and expanded and completed by Sean Manning.

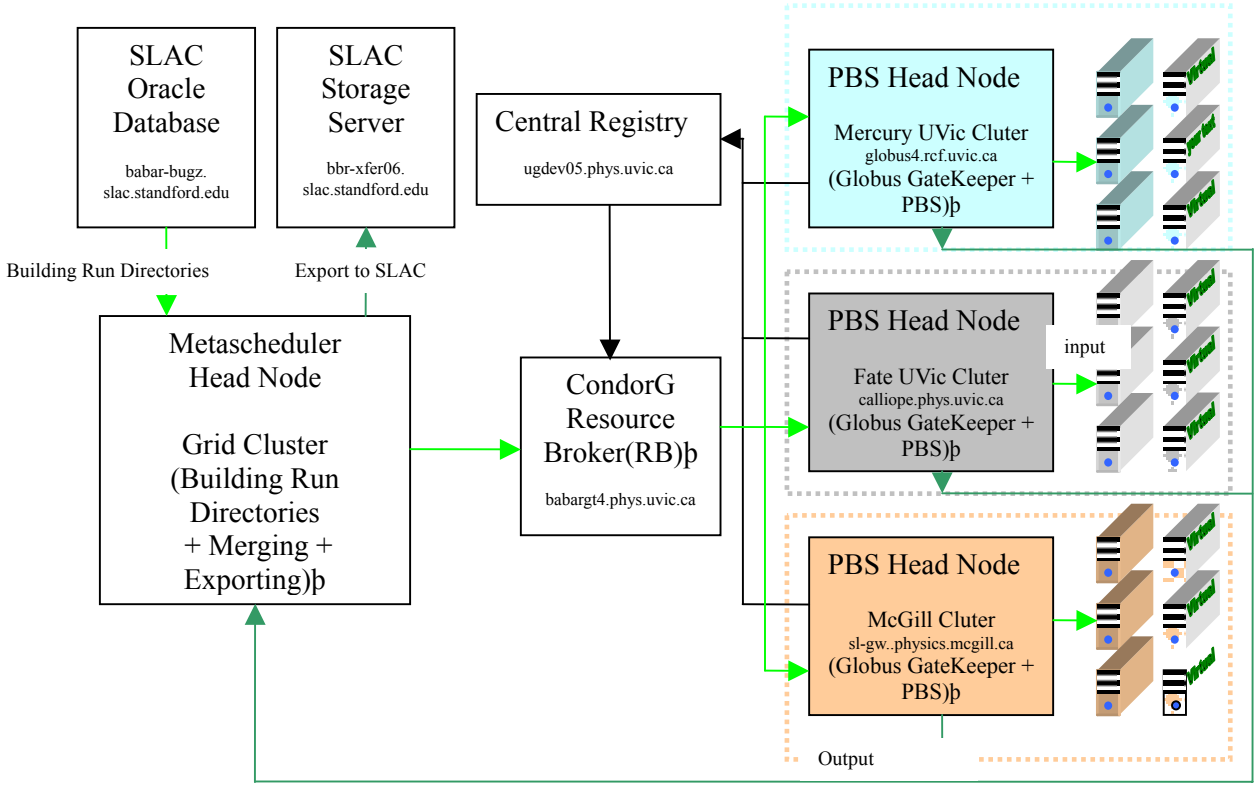
Background: The HEPnet Grid

The Physics department's HEPnet grid is primarily used for high-energy physics applications such as Monte Carlo simulations of events in particle accelerators. It is mainly used by scientists at the university of Victoria, but is also shared with other institutions such as McGill University. It is associated with the Grid X1 project, which was a pioneering Canada-wide scientific grid.[2]

---

[1] Simon Ramage. "Greater Demands on Computational Grids: Can Web Services Facilitate Global Grid Expansion?" Work Term Report, Department of Engineering, University of Victoria, 2006, p. 1. Retrieved from http://www.grid.phys.uvic.ca/documents/reports/reports.html in May 2008.

[2] University of Victoria, "About Grid X1". Retrieved 3 September 2008 from



Xen-based Computational Grid Implementation
(© David Gong 2007)[1]

A grid is organized on three levels. At the lowest level (or the right of the diagram above) are **worker nodes** which perform the actual computation. Worker nodes are organized into clusters, controlled by a **scheduler** or **head node**. The scheduler receives jobs, assigns them to a machine, monitors them, and sends back the output when they are complete. As the name suggests, the scheduler can prioritize jobs based on available resources and specific conditions set by the user who submitted the job using a piece of software called a **local resource manager**. A **metascheduler** allocates jobs to clusters, and is the gateway from the outside world to the grid. It tracks the available clusters, and assigns jobs to them based upon requirements and

[1] http://www.grid.phys.uvic.ca/index.html

priorities set by the user who submitted the job. Just like a local resource manager, it can transfer files back and forth and monitor jobs. The software package which does this is also called a **metascheduler**.

Four software packages are especially important for the HEPnet grid. Globus, Condor, PBS, and Xen provide the basic framework of the grid.

Version 4 of the Globus Toolkit has many different roles. It is an open-source project managed by the Globus Alliance. It consists of many modules designed to help users to share computing power. It provides important capabilities such as security and file staging to supplement the versions provided by Condor. The Web Services interface makes use of several libraries written for Globus by the Globus Alliance, and the Web Services concept is widely supported in Globus. It serves as the backbone to the HEPnet grid.

One feature of Globus is the **gsiftp** network protocol, also known as GridFTP. This is used to transfer files securely inside the grid, or between the grid and client machines. It uses the security features built into Globus (GSI or Grid Security Infrastructure) to communicate over a variant of the FTP protocol. When a user submits a job to the grid using the Globus client, input is transfered to the grid and output is transfered back using gsiftp.

Condor can serve as a metascheduler (Condor-G) or a local resource manager. It is free software from the University of Wisconsin-Maddison which was originally developed to apply unused CPU time to scientific projects. On a Linux system, it receives commands from and sends output to the terminal. A job is submitted to Condor as a submit description file in JDL (Job Description Language) format. Each file consists of attribute-value pairs defining things such as which executable to use and what requirements a machine must meet to run the job. It uses files called ClassAds to describe resources and requests for resources, and performs an

operation called matching to match resources to requests. The main application programming interfaces (APIs) for the Web Services interface are provided by Condor. In the HEPnet grid it is mainly used as the metascheduler.

The Portable Batch System (PBS) is designed as a scheduler or local resource manager.[3] It is used on many HEPnet clusters. Like Condor, it can be used independently or used in a grid. The Web Services interface never interacts with PBS directly, but it is sometimes used to check whether a job submitted by Web Services has been assigned a machine to run on.

Xen helps the grid support applications with specific Operating System requirements. Many scientific applications are only compatible with very specific operating system types and versions. Other applications are large enough that submitting a copy of the program with each job is inefficient. Xen lets a computer run any of several virtual machines complete with an operating system, file system, and installed software. It is normally installed on the worker nodes. The basic Web Services interface does not interact with it at present, but the final version will have to be able to submit and manage jobs which use Xen.

Background: What are Web Services?

The World Wide Web Consortium states that "a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL [Web Services Description Language]). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."[4] That is to say, a web service operates on one

3 See "An Introduction ro Portable Batch System" for an overview. Corbatto, M. "An Introduction to Portable Batch System." Retrieved 1 September 2008 from http://hpc.sissa.it/pbs/pbs.html

4 World Wide Web Consortium (W3C), "Web Services Glossary." Retrieved on 1 September 2008 from

machine but interacts with others using XML messages over the HTTP protocol (a combination known as SOAP). SOAP uses widely supported file formats, and a safe and universally supported network protocol, to maximize the situations in which it can be used. For example, it is easy for applications running on two different operating systems to communicate over SOAP. They do not have to weaken their security by allowing a new type of access to their system, as long as they have HTTP enabled. A Web Services application might be accessed online, or through an application installed on the user's machine, but its interface is described in WSDL and it communicates with other systems using SOAP.

The Traditional Interface

Without Web Services, jobs can be submitted to the HEPnet grid in two ways. Both have their advantages, but also have some deficiencies compared to a Web Services approach.

One approach bypasses Globus and uses knowledge of the implementation of the HEPnet grid. First the user must log in to the metascheduler. The next step is to get permission to use the grid by generating a **proxy certficate** using the grid-proxy-init command. This gives the user permission to submit jobs for a set period by entering their username and password once. The user submits a job to Condor by using the *condor_submit* command on the command line. The job passes from the metascheduler to the head node of a cluster to a worker nodes of the cluster, and runs there. Output is returned using Condor. The job can be monitored on the metascheduler with the *condor_q* command or on the cluster with whatever command is appropriate to the local resource manager. This approach is simple, but limited in its applicability. It requires users to log into a metascheduler (which is both a bottleneck, and too important to allow just anyone to have access to). It also prevents administrators from using

---

http://www.w3.org/TR/ws-gloss/

Globus-related tools to monitor the job.

Another approach uses the Globus interface and ignores the internal implementation of the grid. The user submits a job to Globus by logging in to a machine with the Globus client installed, creating a proxy certficiate with *grid-proxy-init*, and using the *globusrun-ws* command on the command line. The job passes from the client machine to the metascheduler, from the metascheduler to the head node of a cluster, from the head node to a worker node, and runs there. Output is returned to the client via the metascheduler. The job can be monitored using the globus-job-status command from the client machine, or by querying the metascheduler or local resource manager on their respective macines. Because the Globus client can be installed on any machine, jobs can be submitted from outside the grid. This is a better approach, but it has one disadvantage. Users need access to their original machine if they wish to monitor the job.

The Proposed Web Services Interface

Web Services makes it even easier to submit jobs to the grid. In this approach, the user executes a Java program on his or her machine, creates a proxy certificate, and submits the job using the Java program. The job passes from the client machine to the metascheduler over a gsiftp connection, is assigned and passed to a cluster by the metascheduler, is assigned to a worker node by the head node of a cluster, and runs on the worker node. Output then passes back up to the metascheduler, and from the metascheduler to the client. The process of transferring files between machines is known as **input and output staging**. The only connection between the client machine and the grid is the one created when the Java program first runs between the client and the metascheduler.

A Web Services approach offers several advantages. It simplifies the powerful but complex command-line interface. Ideally, jobs could be monitored over the Internet, so a

researcher would not even be tied to the machine from which he submitted his job. A Web Services interface is easy to expand, because every aspect uses the same format to communicate. This is a very important advantage as grids get larger and more complex, involving more different pieces of software which need to communicate with each other. Only a few ports need to be opened to create the interface, so joining the grid is unlikely to conflict with a local security policy.

Challenges

It was decided to develop the interface in Java, since there was a variety of code available in this language to help build Condor and Globus Web Services. Especially important APIs were the Condor and Birdbath packages from the University of Wisconsin-Madison. The cog-jglobus library developed by the Globus Alliance was also useful.[5] About twenty different libraries were needed in all, from a variety of different sources.

The interface was written in Java 1.5. I inherited a version written by David Gong, which had been running and failing on a Windows machine. I carefully documented the code, improved the design, and got the interface to work on several Linux machines. I also added functionality such as the ability to submit any submit description file (not just one hard-coded one) and the ability to automatically retrieve output.

Implementing the Web Services interface was not an easy task. The available APIs are designed for Condor working alone, not as the metascheduler for a Globus grid. Errors often occur deep inside imported libraries which are not documented well enough to understand the errors. One exception produced a stack trace sixty levels deep! A trial-and-error approach to problems was often necessary to solve them, where a solution could be found without

[5] This library is documented in a wiki by the Globus Alliance, "CoG JGlobus". Retrieved 5 September 2008 from http://dev.globus.org/wiki/CoG_jglobus

understanding why it worked. This is somewhat different from the more mathematical approach usually used for stand-alone programs, where problems are solved by precisely locating the problem and then understanding its cause.

The first step was to document the code, which was almost un-documented. This was helpful for me to understand it, and for future programmers who might work with the code. I created a systematic set of comments including a Javadoc description for every class and method.[6] Some changes had been made to the Birdbath and Condor libraries, and I documented these changes so that it would be clear what was new. At the same time I looked for places where the program could be improved according to good software engineering principles. Some methods were redundant, code was duplicated in other places, and other methods could be made more useful and generic.

The next step was getting the code to run on a Linux machine. David had not been successful with this, because he normally programmed on a Windows XP machine. Difficulties included importing the correct libraries, finding and changing all references in the code to a Windows filesystem or David's user account, and identifying which classes needed to work and which could be ignored. There were about a dozen half-written classes which had never been able to compile, but which distracted attention from the classes which needed to compile.

One early problem occurred when I submitted jobs. They consistently failed with the message HoldReason = "Failed to get expiration time of proxy." It turned out that files were being staged to the metascheduler with the wrong ownership and permissions. The copies of files on the metascheduler initially belonged to the user who was running Condor (“root”) not the

6 JavaDocs are a form of in-line comments which can be used to generate a plain HTML web page describing the interface to a class. This makes it easy to share and update documentation. The official Java documentation at http://java.sun.com/reference/api/ is in Javadoc format

user who submitted the job. This caused the proxy certificate to be considered invalid, because they were required to belong to the user who had created them to stop someone simply copying another user's certificate into their own account.

To solve this problem I wrote a Perl script to change the owner of these files and the folder they were in to the user who had submitted the job. Just changing the owner of the proxy certificate was not sufficient. The script had to be able to find the correct owner to give the folder to. The Perl script solved this problem by querying the Condor Quill database for current jobs, taking the owner and job number of each, and using this information to find the correct folder and change its owner. The Perl script was executed every minute by the Unix *cron* utility. Later, we discovered that changing the permissions on this folder was necessary to allow jobs to be submitted in close succession without failing with HoldReason = "Globus error: Invalid file permissions on executable". The Perl script was edited to make these additional changes.

CondorJobSubmitter.java was not parsing JDLs correctly. It required that the JDL be written in a slightly different format, and it refused to interpret values like "TRUE" and "false" as booleans. The parsing turned out to be done by a class for which neither the source code nor good documentation was available. Since it was not possible to solve the problem by studying or changing the code which was doing the parsing, I changed how I called the parser and added some code to treat "true" and "false" in any mix of uppercase and lowercase letters as of boolean type.

There were also some challgenges to do with working with a complicated, working system. For example, when jobs kept failing with HoldReason = "Failed to create proxy delegation" it turned out that my permission to create proxy certificates had expired. The problem was with the wider system, not with the Java code. In another situation, I noticed that

new output and error files were not being staged back from the metascheduler to the client. It turned out that the job was appearing as Completed before these files had finished staging back from the cluster to the metascheduler. When I retrieved the output immediately, these files were not being copied because they had not finished arriving on the metascheduler. I solved this problem by waiting a minute before retrieving the files. The problem was a discrepancy between how I thought the system worked, and how it actually did. Because there were so many places where something could go wrong, tracking down the source of a problem required systematic testing. Understanding an aspect of the grid usually required asking several people or consulting several references, because knowledge was not concentrated in one place. It would be useful to improve the HEPnet TWiki as either a central source of information, or a reference to what can be found where.

The New Java Web Services Interface

By the end of August, the interface was complete. It allows users to submit jobs, retrieve output, and monitor ongoing jobs by executing one of two different Java classes on their machines. The program could be run either through an integrated development environment such as Eclipse, or as a simple executable Jar file.[7] The code written for this project consists of three classes, CondorJobSubmitter, CondorJobStatus, and CondorJobHelper.[8] They are described below.[9]

The condorwsgui.CondorJobSubmitter.java handles job submission, although it can also retrieve output from completed jobs using CondorJobStatus. It is a fairly complex class, but

7 A jar file is a variant of a standard Unix Tar or Windows Zip file

8 David wrote outlines of a dozen other classes, but they were left incomplete and were not incorporated into the current version of the code.

9 For a more detailed references, see the source code or my file WebServicesDocumentation/README.txt. Documentation in Javadoc format, and the soruce code, are also available.

crucial methods include *submitFromFile ()*, which takes the full path to a submit description file and uses that file to submit a job to the grid, *wantNewProxy ()*, which creates a window to learn whether or not the user wants to create a new proxy certificate or use an existing one, and *proxyInitGui ()*, which creates a window allowing the user to enter his password and create the new proxy certificate. It has an inner class, JobWatcher, which handles waiting for the job to complete and retrieving output when it is done. It prints test output to the terminal, but uses arguments or pop-up windows to receive input and give the user information. Its *main ()* method simply takes the location of a submit description file as an argument, submits a single job based upon that file, and tries to retrieve its output when it completes.

The birdbath.CondorJobStatus.java class allows users to monitor jobs and retrieve their output. It is similar in complexity to CondorJobSubmitter. The *printJobStatus ()* method is used to print out information on all jobs belonging to the current user, and the *retrieveJob ()* method retrieves output from a particular job. The AutoUpdate inner class handles getting current information about the jobs in the queue and (optionally) retrieving output when they complete. CondorJobStatus prints all output to the terminal, and takes no input. Its *main ()* method starts to get current information about the jobs in the queue (and to retrieve the output if they are completed) every few seconds, prints status information with *printJobStatus ()*, and stops updating the information about jobs in the queue.

The birdbath.JobHelper class provides supporting methods. It is inserted inside the University of Wisconsin-Maddison Birdbath library so it can access protected methods and variables of that library. Two methods of JobHelper are the most important. One is *getJobAttrFromJDL ()* which takes a file in plain text format, parses it, and returns a list of (name, type, value) triples which represent attributes of the JDL file. The other is technically

two methods, *getStageInFiles ()* and *getStageInFilesVector ()*. It returns the names of the files to be staged in (transfered from the client to the metascheduler) as either an array or a vector of Strings.

Conclusions

The Web Services interface now has had all the basic functionality it needs. I can submit jobs from three different accounts on two different machines, see them Run and Complete, and get the output back. I can also monitor the status of jobs on the same machines and accounts. I had created instructions for installing and using the code, and two co-workers had studied them and confirmed that they were clear. I have submitted a moderate number of jobs in close succession and seen most of them complete successfully.[10] This suggests that the new interface is robust enough to handle large numbers of jobs.

Although a basic Web Services interface has been successfully implemented, there are ways it could be expanded and improved. Possible major improvements include:

- Users often need to submit a batch of very similar jobs, such as repeating the same simulation with different parameters. Right now, CondorJobSubmitter.java is only designed to submit one job at a time, and for each job the user must interact with the Graphical User Interface. It could be automated to submit a series of jobs with some changes.
- At present, every file associated with the job is retrieved to the user's home directory. It would be useful to be able to choose which files to stage back, and where to put them. Traditional Condor supports this, but there are some difficulties doing it in Web Services.
- An interface to monitor jobs online could be added using the PHP programming

[10] To be precise, seven of nine jobs were successful when I submitted them two minutes apart.

language. Condor has added support for this in a recent release.

Acknowledgements

I would like to thank my supervisors, Dr. Ashok Agarwal and Dr. Randall Sobie, the system administrator, Howard Peng, and my co-workers, including Ron Desmerais, Patrick Armstrong, David Grundy, Greg King, and Ian Gable, for their kind help and support this summer. David Gong, who started this project as a co-op student in 2007, was generous with his time in helping me over email.

References

Agarwal, Ashok, et al. "Babar MC Production on the Computational Grid Using A Web Services Approach," *Journal of Physics: Conference Series* 119 (2008 CE) 072002

Corbatto, M. "An Introduction to Portable Batch System." Retrieved 1 September 2008 from http://hpc.sissa.it/pbs/pbs.html

The Globus Alliance, "The Globus Alliance." Retrieved 1 September 2008 from http://www.globus.org/

Gong, David. "Xen-Based Grid Computing Cluster and Condor SOAP Client", Work Term Report, Department of Engineering, University of Victoria, 2007. Retrieved from http://www.grid.phys.uvic.ca/documents/reports/reports.html in May 2008.

Ramage, Simon. "Greater Demands on Computational Grids: Can Web Services Facilitate Global Grid Expansion?" Work Term Report, Department of Engineering, University of Victoria, 2006. Retrieved from http://www.grid.phys.uvic.ca/documents/reports/reports.html in May 2008.

Sun Corporation, "Reference: API Specifications", http://java.sun.com/reference/api/

University of Victoria, "Grid Computing". Retrieved 1 September 2008 from http://www.grid.phys.uvic.ca/index.html

University of Wisconsin-Maddison, "Condor: High Throughput Computing." Retrieved 2 September 2008 from http://www.cs.wisc.edu/condor/

World Wide Web Consortium (W3C), "Web Services Glossary." Retrieved on 1 September 2008 from http://www.w3.org/TR/ws-gloss/