University of Victoria
Faculty of Engineering
Summer 2006 Work Term Report

# Building a Scheduler Adapter for the GridWay Metascheduler

Department of Physics and Astronomy
University of Victoria
Victoria, Canada

Patrick Armstrong
0532342
Computer Science
patricka@uvic.ca

August 29, 2006

In partial fullfillment of the requirements of the
Bachelor of Science Degree

Supervisor's Approval: To be completed by 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: NOT CONFIDENTIAL CONFIDENTIAL

| Signature | Position | Date |
|---|---|---|
| Name | Email | Fax |

# Report Specification

## Audience

This report is intended for, and (hopefully) should be interesting to those involved or interested in distributed and high performance computing, applications of Web services, and computer science and software engineering in general.

## Prerequisites

The reader should have some knowledge and understanding of general computer-related topics. Most of the grid related terminology and jargon are explained prior to their use; however, this document does assume that the reader is comfortable with computers, writing software, and Unix-like systems.

## Purpose

The purpose of this document is to familiarize the reader with grid computing in general, as well as outline some of the work done for the grid computing projects collaborated on by UVic and NRC, specifically the Metascheduling options examined by the team.

# Contents

# 1 Introduction: Grid Computing

In the last few years, many tasks that were seen as computationally unfeasible, such as protein folding, the simulation of particles studied in high energy physiscs and breaking cryptographic cyphers, has become feasible by applying the principles of distributed computing. Generally, distributed computing is accomplished by breaking one large task into smaller tasks that can easily be distributed to machines dedicated to accomplishing these tasks.

Grid computing is a specific type of distributed computing that is used to make use of a wide variety of distributed, heterogeneous resources to solve problems that are too computationally intensive for any one supercomputer, or cluster of computers. A useful analogy is to compare the grid to the power grid. Users do not need to think about which generator will produce the power that will run their appliance, and they assume that the power from the power grid will always be available. In a computational grid, one would simply submit a job to "The Grid", and then the job would be completed on one of the grid's resources, and then be sent back to the client. Ideally, any user could have as much computational power as he needs from the grid, in the same way any client can get as much electrical power as he requires from the power grid.

## 1.1 The Globus Toolkit and the Grid

The Globus Toolkit (GT) is a free software (Apache License, Version 2.0) middleware suite used to create grids. It is considered the standard grid middleware suite, and is a reference implementation for a number of grid standards, such as the OGSI. The aim of the Globus Alliance, the makers of Globus, is to produce a standard, secure mechanism for grid components to interact with one another.

Put simply, the Globus Toolkit provides a mechanism to send a *job*, generally an executable and the data files it requires, to a *grid resource*, generally a cluster of computers dedicated to processing computationally intensive jobs. After the job is complete, the output of the job is sent back to the client machine.

## 1.2 Grid Resource Allocation and Management

The *Grid Resource Allocation and Management* (GRAM) interface is the Globus component for handling the "initiation, monitoring, management, scheduling, and/or coordination of remote computations[2]." GRAM allows users to create and control their jobs through a standard API, making development of grid-enabled applications simple.

GRAM can refer to both the Web services implementation of Globus, as well as the pre-Web services version, implemented in both GT 2 and GT 4. The current version of GRAM does all of its communications through Web services, making it very simple for applications to talk to GRAM and leverage its capabilities.

## 1.3 Web Services

Web services provide a standard way for services to interact with and access one another over a network[1]. The *Web Services Architecture* was defined by the World Wide Web Consortium (W3C) to standardise this interaction using a common XML-based description language called the *Web Services Description Language*, or WSDL. Web services communicate this information using *SOAP*, a protocol for exchanging XML over a network.

Put simply, Web services allow a software program to communicate over a network with another program, similar to the way in which processes can communicate with one another on a single computer system. All that is required of both programs is that they use a SOAP binding for network access, and that they pass messages in the WSDL format. Other aspects of the program, such as the programming language used in implementation, the architecture it runs on, or the internal workings of the software are left up to the developer. This indifference to implementation details makes Web services ideal for communication between grid components, as there are many fewer constraints on grid application developers, encouraging the creation of interoprable, grid-enabled software.

## 1.4 CANARIE Project

The project that the author worked on, and the purpose of the work described in this document was to evaluate the use, creation and deployment of a Web services-based computational grid. This project was funded by

CANARIE, Canada's advanced Internet development organization, and is a joint project between the University of Victoria and the National Research Council.

# 2 Metaschedulers and Resource Brokers

Metaschedulers and resource brokers are a core component of many grid projects. Their job is to pick appropriate grid resources to execute jobs they have recieved from grid users. A metascheduler is an essential component if grid computing is to follow the electrical grid model, where a user can easily send a job to "The Grid" for computation, and expect to have his job completed on one of the grid's resources without his having to think about where it is going. To accomplish this, a metascheduler will fetch information about the state of the grid from registry, then compare the abilities of each grid resource to the requirements of the job. Once this process has completed, the metascheduler submits the job to a resource, and manages the process of transferring data from the client to the resource, and the output from the resource to the client.

The distinction between a metascheduler a resource broker lies in how the machines are set up to be used. Generally, a metascheduler runs on a centralized server, and users submit jobs to this machine from their client machines. The metascheduler then carries out the matchmaking process and submits the job to the resource. This is different from how a resource broker works, where they are usually run on a client machine, which carries out the matchmaking process locally, then submits the job directly to the grid resource. The CANARIE project aims to deliver a grid based around a metascheduler, however many grid projects, such as NorduGrid and Gridbus use a per-client resource broker.

## 2.1 Local Resource Management Systems

A *Local Resource Management System* (LRMS) is a suite of software that manages a cluster of computers, and allows a user to use a cluster of computers with relative ease. In general, an LRMS automates the task of picking a resource with which to run a job, transferring files to worker nodes, managing the execution of the process, and bringing the output back to the

controlling machine. The two types of LRMS currently in use in the Canadian GridX1 grid are called the Portable Batch System (PBS) and Condor.

**The Portable Batch System** The Portable Batch System is an implementation of a simple concept: it takes a computational job that a user has been submitted on a *head node*, a computer that controls the operation of a cluster, then picks one of its dedicated worker node machines to run the job. Once the job is complete, the output of the job is sent back to the head node, where the user can read his output.

**Condor** The Condor Cycle Scavenger works similarly to PBS: jobs are submitted to a controlling central node, called a *Condor Collector*, which then picks a machine under its control to run the job. While Condor can use dedicated compute nodes to run its jobs, it main attraction is that it can make use of idle computing resources to run its jobs, much like the Folding@Home protein folding software. The Condor Collector maintains a pool of resources, which make themselves known after they sit idle. A job can then be submitted and run on the machine, but execution will halt if a key is pressed, a mouse is moved, or there is significant non-Condor CPU usage. In this way, an organization can make use of the many idle hours of CPU time from its workstations, lab machines and under-utilized servers, with minimal additional infrastructure cost.

## 2.2 GridWay

The GridWay Metascheduler is a new Globus "incubator project" that is able to accomplish many of the tasks you would expect of a metascheduler, such as submitting and monitoring jobs and matching using rank and requirements. GridWay also has a number of interesting features. For example, GridWay is able to dynamically acquire and monitor resources using the standard Globus MDS4 Index Service, Which means that a grid resource administrator could easily add his resource to a grid system by forwarding his local registry to the central grid registry, or a registry administrator can add a resource by pulling information downstream from a grid site. GridWay also allows for the use of advanced scheduling policies through the use of "Scheduling Drivers", which can be written in any language using any algorithm. The scheduler simply needs to be able to

output its commands as text, which the central GridWay daemon will execute.

# 3 Metascheduler Deployments

Currently, there are a number of metaschedulers that have been deployed in both production and development evironments. This section will highlight a few of them to explore a couple of the solutions to the metascheduling problem.

## 3.1 GCGate01: Production GT2 Metascheduler

GCGate01, a part of the GridX1 project[4], is a general-purpose metascheduler, and has been used as an interface to jobs from the LHC Computing Grid (LCG) and is now being used for jobs from the BaBar experiment. GCGate01 uses Globus 2.4 middleware to receive and submit jobs to a number of different grid-enabled clusters in Canada.

GCGate01 uses Condor to schedule jobs it recieves, and is able to submit jobs to Globus resources using the Condor-G interface[5]. The creators of Condor-G assumed that Condor-G would act as a front end to the grid, but GridX1 project wanted to use GCGate01 as a central metascheduler, being able to submit jobs to it via Globus. This additional functionality is not supported by default with the Globus Condor job manager, so the GridX1 project modified the stock Condor job manager to be able to use Condor-G grid functionality. Some credential management modifications were also made so that the metascheduler is able to fetch fully proxy credentials from a MyProxy server using the partial proxy that GT receives from the client.

## 3.2 Ugdev06: GT4 Condor Metascheduler

When the CANARIE project was attempting to implement a Web services metascheduler, the team decided that a similar solution to the one used in the GridX1 project could be used. Condor-G now supports submission of jobs to GT4 WS-GRAM resources, and WS-GRAM can still submit to the Condor LRMS.

There are a few differences in implementation, however. First, rather than using MyProxy to manage credentials on the metascheduler, they decided to use Globus's credential delegation facility, the *DelegationFactoryService*. This allows a grid user to simply delegate his proxy to the metascheduler, which that machine can use for any grid operations it does for that user. This way, there is no need to set up a MyProxy server for the metascheduler to use.

### 3.3 Ugdev08: GridWay Metascheduler

An alternate solution to the Condor-G-based solution above is one based on GridWay. The GridWay metascheduler provides a fully functional resource brokering system. One limitation of the default install however, is that it provides no facility for submitting jobs remotely, that is, a user must log on to the machine hosting GridWay to submit jobs.

As a solution to this problem, the GridWay project has created a proof of concept GRAM scheduler interface called GridGateWay. The 0.9 release of this software allows a user to submit simple jobs that do not require any staging in or out of files to the execution host. This is unfortunate, as many jobs will require staging in of custom executables, scripts, and data, as well as the staging out of output created by the job. The remainder of this report will explain the details of implementing these features in the GridWay scheduler interface.

## 4 Improving the GridWay Scheduler Interface

The purpose of a GRAM scheduler interface is to provide a way for the GRAM Managed Job Service to submit, check the status of, and cancel jobs on the LRMS. The most difficult of these tasks is mapping job description metadata written for Globus into a format acceptable for an LRMS. Figures 1 and 2 show an example of such a mapping. The mapping process from Globus metadata to a LRMS like PBS or Condor can be a difficult task, often because of the solutions of the common problems in distributed computing, like file staging and execution management are often solved in different ways, which means there can not always be a one-to-one mapping between the two descriptions. This translation can be especially difficult when the scheduler interface is submitting jobs not to an LRMS, but a

```
<job>
 <executable>/bin/uname</executable>
 <argument>-a</argument>
 <directory>/tmp</directory>
 <stdout>/tmp/stdout</stdout>
 <stderr>/tmp/stderr</stderr>
 <fileStageIn>
  <transfer>
   <sourceUrl>
    gsiftp://machine.ca/tmp/dat.dat
   </sourceUrl>
   <destinationUrl>
    file:///tmp/dat.dat
   </destinationUrl>
  </transfer>
 </fileStageIn>
 <fileStageOut>
  <transfer>
   <sourceUrl>
    file:///tmp/out.dat
   </sourceUrl>
   <destinationUrl>
    gsiftp://machine.ca/tmp/out.dat
   </destinationUrl>
  </transfer>
 </fileStageOut>
</job>
```

Figure 1: Globus job description

```
EXECUTABLE=/bin/uname
ARGUMENTS=-a
INPUT_FILES=gsiftp://machine.ca/tmp/dat.dat dat.dat
OUTPUT_FILES=out.dat gsiftp://machine.ca/tmp/out.dat
STDIN_FILE=/dev/null
STDOUT_FILE=file:///tmp/stdout
STDERR_FILE=file:///tmp/stderr
```

Figure 2: GridWay job description

metascheduler like GridWay. However, before examining the specific details of this problem, let us examine exactly what it is a scheduler interface has to do.

## 4.1 Structure of a Scheduler Interface

A GRAM scheduler interface is implemented as a perl module, and needs to implement three methods: `submit`, `poll`, and `cancel`[6]. Each of these methods corresponds to some exchange of information between Globus and the local scheduler.

The `submit` method is where the actual mapping of the job metadata occurs. This method is always called with one argument, an associative data structure called a GRAM JobDescription object which is created from the job submitted by the user. The submit method takes the values in this structure and creates a job description file that it will eventually submitted to the local scheduler. This function returns the local job ID, which is then used by the `poll` and `cancel` methods.

The `poll` method is used by Globus to get the status of the currently running job from the local scheduler, which it passes on to the client who owns the job. Usually this is done by running the local scheduler's job

status command, for example the `condor_q` command, with the job ID returned by the submit method.

Finally, the `cancel` method is used to cancel a job on the local scheduler using the job ID from the `submit` method. This is done by running the local scheduler's job cancel command, for example the PBS `qdel` command.

## 4.2 The GridWay Interface: Problems and Solutions

As the beginning of this section explained, the mapping from Globus to an LRMS or a metascheduler, is not always simple. Both Globus and GridWay make a number of assumptions about the jobs they are handling, and a variety of workarounds must be devised to make the system work properly.

### 4.2.1 The Stripped fileStageIn Element

The Globus Toolkit assumes that files staged in from a client machine to a Globus host machine will never need to be re-staged to some other machine lower in the grid hierarchy. This assumption has lead to the JobDescription data structure that does not contain a list of the files which are to be staged in to the execution host. This can cause problems with software that needs to re-stage files to its compute nodes, like Condor and GridWay, because these pieces of software generally do not have homogeneous compute resources, so they rarely have shared filesystems. A workaround to this problem must then be devised to solve this problem.

### 4.2.2 The <`extensions`> Element

Globus 4 has provided the `extensions` element as a way for the Globus user to add custom metadata in XML to a job description file. This XML data is passed unparsed to the JobDescription object, and a scheduler interface can parse this XML itself to extract the needed information. As we cannot change how Globus stages files, and we can't force it to include a list of files that we need to stage in the JobDescription object, we must put our list of files to be staged inside this tag, and handle it ourselves in the scheduler interface.

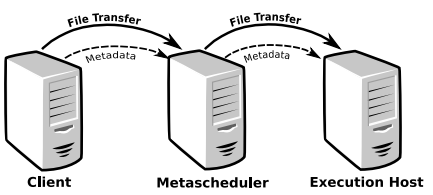


Figure 3: Two-hop file staging

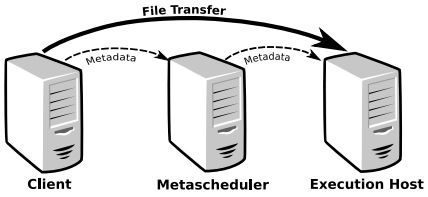


Figure 4: One-hop file staging

Since Globus job description files already have a syntax for describing a file transfer to a directory on the execution host, it makes sense to re-use this syntax when submitting a job. Essentially, this means that we should place our file staging elements inside an `extensions` tag. As a side effect, this also means that Globus will not stage files in to the metascheduler, and if we also place fileStageOut elements in the `extensions` element, it will not stage files out. This happens to be useful, as we can then stage files in and out from the client machine to the metascheduler in one hop, rather than two.

### 4.2.3 One-hop vs. Two-hop File Staging

The Globus Toolkit works under the assumption that the machine that a job is submitted to are where the files are to be accessed from, probably with a shared filesystem like NFS, as was discussed earlier. However, this assumption makes submitting a job to a metascheduler complicated, as we do not actually want the files required for the job to be sent to the metascheduler at all.

Unfortunately, when using standard Globus job metadata this is impossible, as at job submission time the client does not know where to send the input files. Only once the metascheduler has picked a resource for the job do we know where the files need to be sent. As a result of this, we must use *two-hop file staging*, where we first copy all of the data files and executables needed to run the job to the metascheduler, then after the job has been scheduled, it must be sent to the execution host. When the job has finished executing on the execution host, we must reverse the process. This double copying of data, executables, and output is inefficient, and will be especially costly when dealing with the enormous size of the data used in scientific computing. Figure 3 illustrates two-hop file staging. A superior solution to two-hop staging is to stage data directly from client to

resource, and from resource to client. This is called *one-hop file staging*.

One-hop file staging eliminates the problems of two-hop staging by transferring file staging metadata to the metascheduler, which is then passed on to the execution host. The metascheduler then initiates a third party file transfer from the client machine to the execution host. While one-hop file staging is an ideal solution, it can sometimes be difficult to implement. Condor-G, for example, requires the files it will stage be present on the metascheduler before it can transfer the files to another machine running GridFTP. There is no way to do third-party transfers.

Fortunately, GridWay *is* able to do file transfers from a remote machine to to an execution host using a *gsiftp://* URL. This means that only metadata describing the file transfer must be sent to the GridWay metascheduler, not the actual data itself. This results in faster data transfers, as there is only one file transfer operation per file, rather than two as well as less load on the metascheduler itself.

#### 4.2.4 Implementing One-hop File Staging

Earlier we discussed how moving the `fileStageIn` element inside the extensions tag prevents Globus from parsing the element, thus preventing Globus from staging the file in to the metascheduler. This is good, as we want to prevent Globus from doing this, but we still need to handle file transfers somehow. We can do this by passing off all the file transfer directives to GridWay itself. The scheduler adapter will then need to parse the file transfer elements, and format them in the syntax GridWay expects for its `INPUT_FILES` field.

One benefit of this solution is that submitting a job that stages in files requires only a tiny change to a Globus job description file intended to be used for submission directly to a Globus resource: the must put `extensions` tags around the fileStageIn and fileStageOut elements. A two-hop file staging solution using the Condor-G metascheduler required adding modified fileStageIn and fileStageOut elements to the `extensions` tag, while keeping standard fileStageIn and fileStageOut elements inside the `job` element. This resulted in rather clunky and confusing job description files.

```
EXECUTABLE=gwgramwrapper
ARGUMENTS=--fastmath --roundfloats
INPUT_FILES=gsiftp://sci.ca/tmp/science science
```

Figure 5: GridWay job with staged executable

```
#!/bin/sh
# Wrapper script for gridway

chmod 744 science
./science $@
```

Figure 6: Generated executable wrapper

### 4.2.5 Staging in Executables

Unfortunately, GridWay makes a few assumptions about the jobs that it is requested to submit as well. Contrary to the official documentation[7], one of these is that GridWay expects the executable to be run will already be present on the execution host, or that they will be staged from the metascheduler. This restriction imposed by GridWay makes it seem that we will have to make a special exception for staging in executables, and use two-hop staging while using the cleaner one-hop staging for input and output files. Luckily, because of a few specificities of this problem, we are able to use a fairly simple workaround. When GridWay submits a job to a Globus resource, it puts all of its files inside a hidden directory on the execution host, and launches any staged executable from inside that directory. Thus, since we know where the executable will be launched from, and because we know all files will be staged to this same directory, we can write a small wrapper script that will launch our main executable. All this script will have to do is launch the executable specified in the JobDescription object, and pass on any arguments it receives to the executable it launches. We can then do a one-hop stage of the executable, and dynamically generate this script to send to the execution host. Figures 5 shows a job description file that would require the use of such a wrapper, and figure 6 shows the generated wrapper.

## 4.3 Current Status and the Future

Currently the GridWay scheduler interface is installed and running on ugdev08, and is getting information about its resources from the testbed registry, ugdev05. It can run jobs that stage in and out an arbitrary number of files using one-hop staging, and can even stream output back from the jobs. Despite these successes there are a number of issues that should be dealt with in the future.

### 4.3.1 Proxy Management

The stock GridWay scheduler adapter requires that a user manually initialize a proxy on the metascheduling machine. A simple workaround was to copy a delegated proxy to the default location for a user proxy, /tmp, which GridWay could then use. There is a minor security issue with this solution when more than one grid user is mapped to the same user account, but this should not be done anyway. For now, a GridWay administrator should ensure that there no user account is mapped to more than one grid user.

### 4.3.2 Rank and requirements

Specifying rank and requirements for a Globus submitted to a GridWay metascheduler is not yet supported with the scheduler adapter. This should be easy to implement however, as GridWay job template files have a very simple syntax for their specification. The XML syntax for rank and requirements must be considered carefully however, especially if the metascheduler is to be considered a modular component, that is, we expect that at some point in the future a superior metascheduler shall be created that we might like to switch to. This new metascheduler may use a different syntax from GridWay for specifying rank and requirements, in the same way that Condor has a different syntax from GridWay.

As a solution, it would be best to make use of the strengths of XML and describe rank and requirements semantically. An example solution could have a root tag named `rank`, with subtags like `CPUSpeed`, `freeRAM` and `operatingSystem`. This would make interpretation of the XML data into different formats simple for different scheduler adapters. It is future-proof as well, as these values are likely to be important to any metascheduler, and are not tied to a specific implementation.

### 4.3.3 Integration with Registry

Currently, GridWay gets metadata about grid resources from the default GRAM information providers included with a Globus install. While these information providers do provide a minimal amount of information about the grid resources they describe, they do not provide nearly as much information as is provided by the GLUE Schema[8], nor the extended GridX1

Schema[9]. Adding an information driver that can interpret the GridX1 Schema should not be particularly difficult, as there is already example source code included with GridWay for gathering monitoring information from stock MDS4 Registries and the LDAP mapping of the GLUE schema (MDS2) as well.

# 5 Conclusion

Choosing the proper metascheduling solution for a Web services computational grid will be a difficult task. There are a number of benefits and disadvantages to both the Condor-G metascheduling system and the GridWay-based system. Overcoming the disadvantages to both solutions is one of the tasks that need completion in the next few months of the CANARIE project. The author is confident that overcoming these obstacles will be accomplished without much difficulty by the team at

# References


[1] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard. "Web Services Architecture." 11 February 2004.

[2] Foster, I. "A Globus Primer." 8 May 2005.

[3] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist. "X.509 Proxy Certificates for Dynamic Delegation." 2004.

[4] A. Agarwal, M. Ahmed, B. Caron, A. Dimopoulos, L. Groer, R. Haria, R. Impey, L. Klektau, C. Lindsay, G. Mateescu, D. Quesnel, R. Simmonds, R. Sobie, B. St. Arnaud, D. Vanderster, M. Vetterli, R. Walker, M. Yuen. "GridX1: A Canadian Computational Grid."

[5] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke. "Condor-G: A Computation Management Agent for Multi-Institutional Grids." 2 November, 2004.

[6] "WS-GRAM Scheduler Interface Tutorial (Perl Module)." http://www.globus.org/toolkit/docs/4.0/.

[7] "GridWay 5 Documentation: User Guide".

[8] S. Andreozzi, S. Burke, L. Field, S. Fisher, B. Konya, M. Mambelli, J. Schopf, M. Viljoen, A. Wilson. "GLUE Schema Specification, version 1.2". 3 December 2005.

[9] "Resource Schema Description and Definition for GridX1 Services Project". 9 June 2006.