Building a Grid Portal for Teragrid's Big Red

Mehmet A. Nacar, Jong Y. Choi, Marlon E. Pierce, and Geoffrey C. Fox

Abstract— We describe the Big Red Portal, which builds on the Open Grid Computing Environment (OGCE) portal software. In addition to standard OGCE capabilities, this portal includes MEME job submission and job dashboard portlets that are built using OGCE and related portlet libraries. To simplify the development of such portlets in the future, we introduce an XML tag library approach that encapsulates common Grid operations for rapid development

1 Introduction

TeraGrid Science Gateways are science portals that are designed to provide high level, user-centric services and interfaces to the National Science Foundation's TeraGrid. Several of these efforts are described in an upcoming special issue of *Concurrency and Computation* [1]. Although not strictly speaking a "science gateway", the TeraGrid User Portal is a related and similarly architected portal system [2] that shares many common features with the gateway project. TeraGrid gateway development tends to be a heroic effort, requiring full time developers. However, it is possible to start from a base of reusable software that encapsulates basic gateway operations, such as accessing Grid resources. The Open Grid Computing Environments (OGCE) project [3] provides one such toolkit for gateway building: several JSR 168 compliant Grid portlets and gateway services are deployed into the GridSphere portlet container [4]. OGCE Grid portlets are built on top of Java CoG Grid programming abstractions [5].

This paper describes our efforts using the OGCE portal software to build a portal that combines characteristics of a "system" portal (incorporating some TeraGrid User Portal features such as GPIR resource browsing) and a science gateway for bioinformatics applications. Our primary backend resource is Indiana University's Big Red supercomputer (part of TeraGrid), but our portlets have been tested across other TeraGrid resources. The second half of the paper reviews our work to simplify the development of these portlets by using Grid tag libraries, which encapsulate common Grid tasks.

2 Extending OGCE to Support Big Red Portal

Big Red is a major new TeraGrid resource and one of the most powerful computers in the world. As with all TeraGrid resources, it runs the Coordinated TeraGrid Software and Services, which includes Globus services. One of Big Red's initial applications is Mutiple EM for Motif Elicitation (MEME) [6]. MEME is used to discover common motifs in groups of DNA or protein sequences. Due to its computational complexity, MEME should be executed in a rich resource environment such as Big Red. However, to execute MEME on Big Red, a user must not only be familiar with the application itself: he or she must also understand various network tools such as FTP for uploading and downloading input and output files, and he or she must understand Big Red's LoadLeveler and MOAB-based scheduling and queuing system in order to submit, monitor and control jobs. This kind of inconvenience can be easily overcome by making a specific portlet that allows a user to execute the MEME application by using a science portal based on the OGCE [3]. In addition to MEME execution, we can add file management and job control functionality into the portlet by using Java Commodity Grid (CoG) toolkit [5] to utilize Big Red's Grid infrastructure.

2.1 Making MEME Portlet Out of GRAM and GridFTP Libraries

The main function of the MEME portlet (Figure 1) is to submit a MEME job to a remote TeraGrid resource such as Big Red. This can be done either interactively (for very small jobs) or through Big Red's queuing system. To provide a more convenient and user-friendly interface, our MEME portlet also enables a user to transfer input and output files to/from the remote server and track the status of a submitted job. For example, a user can upload a gene sequence file to a remote server and submit a job to execute MEME application with the file as an input. After submission, a user can check whether the job is completed or not. Once the job is completed, a user can download output files from the server to his local machine. From the user's perspective, such operations can be done by simply clicking a few buttons. Under the user interface, the main Grid tasks that the MEME portlet performs are submitting the job, running the GridFTP client, and checking the submitted job's status.

We will explain how MEME portlet execute these Grid tasks with a typical use case. First, the user needs to transfer a gene sequence file from his or her desktop machine to the desired resource (i.e., Big Red). Since a web portal service (i.e., OGCE) is running between a user and a remote server, MEME portlet should transfer the file from the user's local system to the remote server, where MEME application will be executed, via the server where a web portal service is in operation. For this end, the MEME portlet executes two consecutive file transfers by using two different classes: PortletFileUpload class, a library from Apache

- *Mehmet A. Nacar, Department of Computer Science, Indiana University*
- *Jong Y. Choil, Department of Computer Science, Indiana University*
- *Marlon E. Pierce, Community Grids Lab, Indiana University*
- *Geoffrey C. Fox, Department of Computer Science, Indiana University*
- *Author emails respectively: {mnacar, jychoi, marpierc, gcf}@indiana.edu*

Commons, for form-based file upload and the CoG's GridFTPClient class. While the PortletFileUpload class enables the upload of a file from the user space to the server where web portal service is in operation, the GridFTPClient class can send a file from this server to the remote gateway where MEME will be executed.

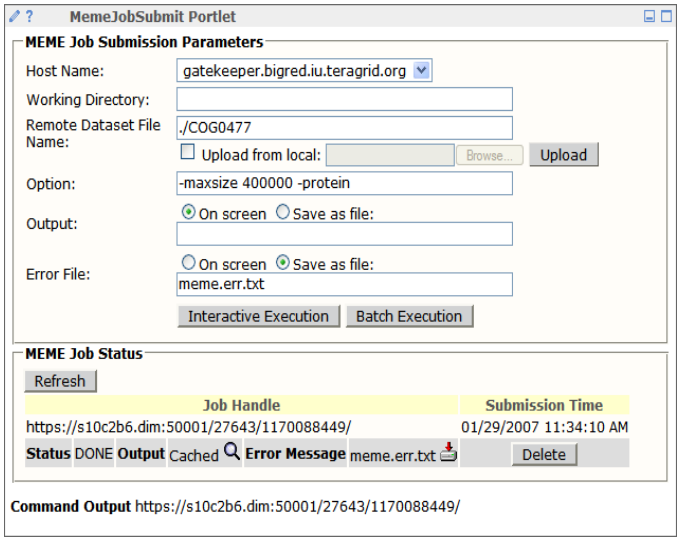

Figure 1 The MEME portlet uses OGCE portal libraries to upload and download files, submit jobs, and monitor their progress.

Secondly, the user will fire a job submission by clicking a button. Receiving this event, the portlet will submit a job to the remote server's job scheduler, which will pick the MEME application to run. To submit a job, we use two Java CoG classes: GramJob for managing job submission and GassServer for receiving outputs from the remote server. More specifically, before submission of the MEME command, a Resource Specification Language (RSL) [7] string is created as an input of GramJob class to define a MEME job with its working directory, location of input file, and location of outputs from stdout/stderr. If a user wants to see outputs on a screen instead of saving as a file, the MEME portlet will run a GASS server, which is designed to receive outputs from a remote server in an on-line manner, and include the GASS server information in the RSL string so that the user can see outputs directly through the portlet. Regarding the submit options, a user can choose to submit a job in two modes: interactive mode and batch mode. While the user should wait to receive results in interactive mode, in batch mode a user can check the result later instead of waiting for immediate output. To enable a user to access the result later in batch job submission, the portlet saves a job handle string returned by getIDAsString() of GramJob after job submission into a persistent storage. In the MEME portlet, job handles are saved by using PortletPreferences's save() function so that a user can retrieve anytime even after logging out.

Thirdly, after submitting a job in a batch mode, the user can check from the MEME portlet whether the job is finished or not. By retrieving the job handles saved in the previous step, the portlet will check the status of the job by calling jobStatus() function of Gram class with a job handle as an input.

Finally, when the batch job is completed, the user can download outputs from MEME. If the output is saved as a file in remote server's file system, we download the file by the GridFTP protocol, using the GridFTPClient class. Otherwise, we execute a remote command to retrieve the output by using the job handle. For this end, our MEME portlet will submit an interactive GRAM job to query output by giving the job handle as an input.

2.2 Job Tracking with a Dashboard Portlet

The Dashboard portlet (Figure 2) is designed to provide information about job status by using Big Red's job manager. As a default job manager, Big Red is using the MOAB job scheduler. Thus, a submitted job to Big Red can be monitored by querying MOAB's job queue status, and this can be done by using a command line tool, called *showq*, provided by MOAB. By executing the *showq* command on the behalf of a user in Big Red, our dashboard portlet can display job queues and status so that a user can easily access the information about the submitted job. We have adopted this simple approach to work on other TeraGrid resources as well.

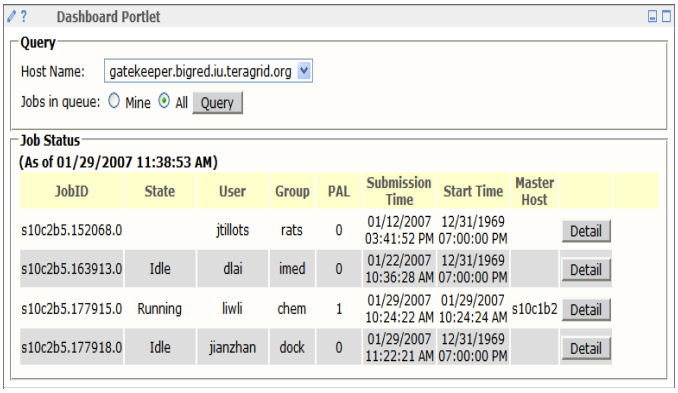

Figure 2. The dashboard portlet allows users to track jobs on the selected resource. The user can view either his own set of jobs or get information on all submitted jobs.

In many cases, it is desirable to only show a user his or her specific jobs. This can be done by remotely executing *showq* with the proper arguments such as the user ID. However, TeraGrid does not provide a global UNIX user ID system, so a user can have different user IDs on different machines, even though the user's Grid credential provides single sign on. One simple solution to evade this problem is to execute the *whoami* command to obtain the correct user ID in a gateway before executing *showq* command. This can be made into a one-time task by saving the user ID as a portlet's preference value, instead of executing every time before *showq* execution.

To execute the remote *showq* command, our Dashboard portlet follows a three step procedure: *whoami* command execution, *showq* execution, and finally output parsing. Executing *whoami* and *showq* command can be done the same way that we execute MEME commands, using GramJob and GassServer Java CoG classes. Since we need immediate

results, each execution is performed in interactive mode. Once obtained from whoami, the right user ID will be given as an input to the *showq* execution. After executing showq command, the output is parsed in order to be displayed inside portlet as an HTML document. Since the *showq* command on Big Red has an option to output in XML format, we use a XML parser, known as XML Pull Parser (XPP) [8], to convert output into a proper HTML object.

2.3 Other Portlets

The OGCE release comes with several other portlets (GridFTP, WS-GRAM, Pre-WS GRAM, MyProxy credential management) that we have adopted as-is. We have also configured the OGCE GPIR portlet to point to the TeraGrid's GPIR Web Service [9], thus providing a global view of resource load and related information (see [2] for more information). In addition to these Grid-centric portlets, the OGCE IFrame portlet provides a simple mechanism for integrating non-portlet Web pages. We used this in the Big Red portal to provide an interface to Indiana University's Knowledge Base website [10].

2.4 Integration with Other TeraGrid Resources

Although designed to work with Big Red, our portlets can be used with any other gateway in TeraGrid. To provide the same functions transparently, our portlets provide a few methods to allow a user to customize environmental settings such as execution path and working directory. Such values can be redefined by changing portlet.xml file (which must be done by the portal administrator) or by using the portlet's EDIT function, a standard interface to change user's preferences. In the case of the Dashboard portlet, discordance of user ID between a user certificate and a remote system can be a problem. To avoid this problem, we can submit a Gram job to execute *whoami* command to find a correct user ID.

3 Integrating GTLAB With Big Red Portlets

Portlets provide a common component for building portals out of reusable parts. For example, as mentioned previously, the OGCE portal has portlets for job submission, credential management, and file management that can be plugged into any standard compliant container. Often, however, as in the case of the MEME portlet described above, portlets are not quite fine-grained enough components. We would like to build portlets that combine several Grid operations in the same portlet. Our work on Grid Tags Libraries And Beans (GTLAB) provides a set of Java Server Faces (JSF) tag libraries and backing JavaBeans (called Grid beans) that attempt to solve this problem [11][12]. A full discussion of JSF is out of scope here, but briefly, JSF generates HTML from a set of XML tags. HTML form actions are associated with so-called backing JavaBeans, which in turn may act as Web Service clients or connect to databases. Developers can extend these libraries to provide their own XML tags.

The goal of GTLAB is to simplify the process of Grid portlet development by encapsulating common Grid operations as XML tags that can be embedded in portlet pages, enabling rapid development. GTLAB capabilities include credential management, remote file operations, remote job executions, and file transfers.

The JSF Web application framework provides us with an extensible component architecture. Each XML tag is associated with a backing Grid bean that implements the actual Grid clients, which we build with the Java CoG kit [5]. We use JSF's built-in functionality to pass attribute values from the XML tags to the backing beans. Grid beans are associated with Grid tags and their action methods are fired by our 'submit' tag. Tracking the jobs and monitoring is also part of the GTLAB framework.

3.1 How to use GTLAB within Big Red portlets

Typically a Grid portlet stages various related tasks in response to a user-generated event. These are usually the nodes of a Directed Acyclic Graph (DAG), which our Grid tags are designed to support. The DAG, or composite task, is called *multitask* in GTLAB. Currently, multitasks only allow dependent task units and prevent parallel tasks and cycles.

After building the sub-tasks, multitask and their dependencies, GTLAB then registers multitasks in the browser session. In addition, it registers their handler information within the session to track their lifecycle. All of the objects are stored in hash tables with a unique key. The job handler information can be stored persistently to a backend storage system (i.e., a database) by setting *persistent* attribute of multitask.

The following use case explains a multitask for MEME with sub-tasks and their dependencies. Assume a developer has been assigned the job of creating a portlet to do the following basic tasks. First, Task A makes a working directory on Big Red. Then, Task B transfers an input file from a remote host to the newly created directory. Finally, Task C is responsible for submitting a command script on Big Red using the input file. The following sections explain the scenario in detail through the use of Grid tags. After the portlet is finished and deployed, users will then submit and monitor jobs using the developer's portlet. Users will not see the tag libraries and will interact with standard HTML pages that get generated when the portlet is rendered.

3.2 Preparing Application Pages

The developer starts by creating a JSF form that generates the HTML interface (Figure 1). After the HTML form is prototyped, the developer can now add GTLAB tags to Grid-enable the HTML form submission components. GTLAB tags consist of non-visual page action components with one exception: we override the 'submit' button tag that propagates user events to the backend.

A full example is given in Table 1. The GTLAB tag part that specifies the Grid actions is surrounded by GTLAB *submit* tag, which in turn is contained within JSF view page. Table 1 shows the key GTLAB tags for constructing the Dashboard portlet. As explained in the previous section, this portlet basically submits two dependent jobs using GRAM service. The first one is to retrieve the user ID on the specific TeraGrid resource, and the second is to submit the *showq* command. These two jobs and their dependency are shown in Table 1. Note resulting output data must still be

formatted for display.

To generalize this portlet, we will need to associate tag ers.

After submission, the GTLAB job handlers can be used to

```
<o:submit id="track" action="list_page" />
       <o:multitask id="dashboard" taskname="track" persistent="true" >
            <o:myproxy id="proxy" hostname="gf1.ucs.indiana.edu" port="7512"
                       lifetime="2" username="dash" password="******" />

            <o:jobsubmit id="jobA" hostname="cobalt.ncsa.teragrid.org"
                         provider="GT4" executable="/bin/whoami"
                         stdout="tmp/result"
                         stderr="tmp/error" />

           <o:jobsubmit id="jobB" hostname="cobalt.ncsa.teragrid.org"
                         provider="GT4" executable="/bin/showq"
                         stdin="tmp/result" stdout="tmp/list"
                         stderr="tmp/error" />

            <o:dependency id="depend" task="jobB" dependsOn="jobA" />

       </o:multitask>
</o:submit>
```

Table 1 GTLAB example for creating Grid portlets to collect data for the dashboard portlet.

attributes with information collected from the user. These inputs (i.e., the specific computer hostname to use or the name of the task) correspond to XML tag attribute fields with dynamic parameters. In other words, attribute values should be supplied by the user in a dynamic web user interface. We have defined *resource* bean to manage these specific user inputs. The resource bean represents all properties of the GTLAB tags and supplies default values. The application programmer has to tie the user inputs with corresponding property using resource bean as follows:

```
<h:outputText value="Taskname: "/>
<h:inputText value="#{resource.taskname}" />
<o:multitask id="multi" persitent="true" task
        name="#{resource.taskname}" />
```

Here, <*outputText*> and <*inputText*> are standard JSF tags that are rendered as text and input text fields, respectively. The attribute value *#{resource.taskname}* uses JSF's Expression Language (EL) syntax. The user will be prompted to provide a name for the particular task, which will also be used by GTLAB's <*multitask*> tag as the value for its *name* attribute.

3.3 Tracking and Managing Jobs with GTLAB

Grid applications typically must submit jobs to batch queues, and even interactive jobs may take a several minutes to finish. Thus we must provide a callback system that lets jobs run while allowing the portal to return control to the user. Thus the GTLAB tags need to track the jobs' lifecycle and monitor their status, displaying this information back to the user.

GTLAB creates a handler for every submitted job and displays status information using JSF data tables (which are rendered as HTML tables for display). These data tables are fed by job handlers that are saved in hash tables within the session. The visual design of the job tracking's display table and filtering on the values are left to application developmanage, stop, or cancel running jobs. Permanent job archiving is also tied to job handlers. For example, users can keep good samples, remove old jobs or failed jobs, and otherwise organize their repository. The job's metadata features (submit time, status, finish time, output location and input parameters) are stored and can also be listed.

4 Conclusions

We have described in this paper our work to build a simple science gateway for Indiana University's Big Red supercomputer, based on the OGCE portal software release. In our discussion, we have focused on new portlets for MEME job submission and job tracking that we developed from OGCE and related libraries. We then described our work to simplify the process for creating new Grid portlets using Java Server Faces tag library extensions.

Acknowledgment

This work was supported by the National Science Foundation's National Middleware Initiative program.

References

[1] Nancy Wilkins-Diehr, Special Issue: Science Gateways - Common Community Interfaces to Grid Resources. Published Online: 10 Oct 2006 DOI: 10.1002/cpe.1098. Available from http://www3.interscience.wiley.com/cgi-bin/fulltext/113391281/PDFSTART.

[2] Maytal Dahan, Eric Roberts, "TeraGrid User Portal v1.0: Architecture, Design, and Technologies." Second International Workshop on Grid Computing Environments GCE06 at SC06, Tampa, FL. Nov. 12-13 2006.

[3] Jay Alameda, Marcus Christie, Geoffrey Fox, Joe Futrelle, Dennis Gannon, Mihael Hategan, Gopi Kandas-

wamy, Gregor von Laszewski, Mehmet A. Nacar, Marlon Pierce, Eric Roberts, Charles Severance, Mary Thomas, ***The Open Grid Computing Environments collaboration: portlets and services for science gateways***. Published Online: 10 Oct 2006 DOI: 10.1002/cpe.1078. Available from http://www3.interscience.wiley.com/cgi-bin/fulltext/113391287/PDFSTART

[4] Jason Novotny, Michael Russell, Oliver Wehrens: GridSphere: a portal framework for building collaborations. Concurrency - Practice and Experience 16(5): 503-513 (2004).

[5] Kaizar Amin, Mihael Hategan, Gregor von Laszewski, Nestor J. Zaluzec: Abstracting the Grid. PDP 2004: 250-257.

[6] The MEME/MAST System. [Online] http://meme.sdsc.edu/meme/intro.html.

[7] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, Steven Tuecke. "A Resource Management Architecture for Metacomputing Systems". LNCS Vol. 1459, 1998

[8] Aleksander Slominski. Design of a Pull and Push Parser System for Streaming XML. Technical report, Indiana University Computer Science Department, 2002. Available from http://www.extreme.indiana.edu/xgws/papers/xml_push_pull/. last accessed in March 2005

[9] M. Dahan, M. Thomas, E. Roberts, A. Seth, T. Urban, D. Walling, J.R. Boisseau. "Grid Portal Toolkit 3.0 (GridPort)", in Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, 4-6, pp.272 - 273, June 2004

[10] Indiana University Knowledge Base. [Online] http://kb.iu.edu/.

[11] Mehmet Nacar, Marlon Pierce, Gordon Erlebacher, Geoffrey Fox. "Designing Grid Tag Libraries and Grid Beans." Second International Workshop on Grid Computing Environments GCE06 at SC06, Tampa, FL. Nov. 12-13 2006.

[12] Mehmet A. Nacar, Mehmet S. Aktas, Marlon Pierce, Zhenyu Lu and Gordon Erlebacher, Dan Kigelman, Evan F. Bollig, Cesar De Silva, Benny Sowell, and David A. Yuen VLab: Collaborative Grid Services and Portals to Support Computational Material Science Dec 30, 2005 Special Issue on Grid Portals based on SC05 GCE'05 Workshop, Concurrency and Computation: Practice and Experience.