UNIVERSITY OF CALIFORNIA

Santa Barbara

Integrating Condor and Queue Bounds Estimation from Time Series (QBETS) Into the UCLA Grid Portal

A Thesis submitted in partial satisfaction of the

requirements for the degree Master of Science

in Computer Science

by

Kerby Obadiah Johnson

Committee in charge:

Professor Rich Wolski, Chair

Professor Chandra Krintz

Professor Amr El Abbadi

June 2007

The thesis of Kerby Obadiah Johnson is approved.

Amr El Abbadi

Chandra Krintz

Rich Wolski, Committee Chair

May 2007

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support, encouragement and collaboration of my family, friends, colleagues, and mentors. Throughout the entire work I have been surrounded by people encouraging, teaching and challenging me to go deeper and learn more, while also providing laughter, joy, and a multitude of enjoyable and necessary distractions.

My family has always both supported me and provided a necessary outlet to discuss interesting topics. The daily phone calls to discuss the stock market, investments and who is currently winning in any number of competitions among family members have provided that lighthearted spark of competition and necessary breaks. Thank you.

My friends and housemates have also been extremely helpful in my efforts throughout this thesis. Coming home to *good* home-cooked meals and being able to eat and then go upstairs and work has proven a tremendous boon, both to my health by being able to avoid fast-food and the time that it saved me. And who could forget the Smash Brothers sessions that always seemed to last longer than anticipated but were endlessly enjoyable. Thank you Megan, Chris, and Eric for everything.

I also want to thank my girlfriend of over three years, Lizzie, for all the love, support and encouragement you have shown me throughout this process. I know it has been hard on you with how busy we have both been, but you have always been supportive and understanding and more than willing to take me out to dinner when I need a break or politely hint that it would be a good idea for me to take you out. Thank you for everything.

My colleagues and lab mates, it has been an incredible experience and you guys have always made the lab an enjoyable, fun and yet somehow productive work environment. The multitude of coffee breaks, shared experiences and random, irrelevant and delightful conversation has truly been a lifesaver throughout this process. Thank you everyone.

Finally, I want to thank my mentors Dmitrii Zagorodnov and Rich Wolski. Thank you both for teaching me how to conduct research, ask the right questions and investigate matters critically. This has been a learning and growing experience in my life and your support and guidance have definitely helped me make the transition from interested college student to productive researcher and developer. Not only have you taught me how to conduct interesting research, but you have also guided my ability to critically express ideas in writing. Learning how to organize, present, and explain detailed scientific content from both of you has been both a challenge and a blessing. I know that the skills I have learned while writing this thesis will benefit me throughout my life and I appreciate the effort and patience you both have shown throughout the process. Thank you my mentors, and my friends.

This work was supported by a grant from the National Science Foundation numbered 0093166.

ABSTRACT

Integrating Condor and Queue Bounds Estimation from Time Series (QBETS) into the UCLA Grid Portal

By

Kerby Obadiah Johnson

Grid Computing facilitates the sharing of distributed, heterogeneous resources owned by different organizations, each with their own goals, policies, and security requirements. One of the challenges facing Grid Computing is finding an optimal way for users to interact with Grid resources without needing to know the underlying details of the Grid. Grid Portals have been proposed as a means to address this challenge by providing transparent access to the Grid through a Web Portal interface.

The UCLA Grid Portal is being developed to create Grids for the UC Campuses. The Portal provides cluster scheduler information, data management and job management to the user. It supports clusters running PBS, SGE and LSF as cluster schedulers. To increase the resources available in the UCLA Grid Portal and to provide better support for some types of jobs, this thesis adds support for Condor, a prominent cluster scheduler, into the UCLA Grid Portal.

Condor was added by using a Condor job manager (provided by Globus) and by adding scripts for collecting cluster status information.

The UCLA Grid Portal provides limited feedback to its users; in particular, it is difficult for users to know where they should submit jobs for fastest execution. This thesis enhances the UCLA Grid Portal by adding QBETS (Queue Bounds Estimation from Time Series) as a standalone portlet to the Portal. QBETS provides an upper-bound prediction of the queue delay a job will experience at a cluster or the probability that a job will start at a cluster by a deadline. This thesis integrates QBETS into the UCLA Grid Portal by adding monitors that communicate job accounting information to the QBETS backend database and a standalone portlet in the Portal that uses a web service query to retrieve predictions for the clusters available to a user.

Table of Contents

1. Introduction

1.1. Grid Computing

Computational Grids (1) (2) (3) (4) have become popular due to their ability to solve large-scale problems by harnessing the power of massive distributed systems. The term *Computational Grids* refers to the infrastructure that enables the increases in shared resources such as computer cycles, storage, sensors, etc. Grid Computing is an important research area because of the challenges of enabling the sharing of heterogeneous resources from multiple organizations, each with its own goals, policies and security requirements.

1.2. The Grid Problem: Management and Organization

The access to shared resources does not come without its challenges. Specifically, a main challenge of Grid Computing is how to manage a variety of heterogeneous Grid resources from different organizations with interfaces that do not provide a customized problem solving environment (PSE) (3). "A primary barrier in the widespread acceptance of monolithic client side tools is the deployment and configuration of specialized software. Scientists and researchers are often required to download and install specialized software libraries and packages" (3). The resources shared by these organizations and the policies governing access to the resources are optimized for the specific organization's goals, policies and security requirements. Each resource follows its own policies about who can access the resource, what kind of jobs can be run on the resource, how long jobs can be run, and many other requirements that vary widely among resources. An additional complication arises because the allocation allotment of some organizations might be insufficient to meet a user's job requirements. Thus, it becomes hard for the user to manage jobs and resources in the face of differing policies, heterogeneous clusters, and non-customizable interfaces. Users need to keep track of where their executables are stored, which jobs were submitted to specific resources and what the output of the jobs was. We believe that a primary goal of Grid interfaces is to allow the user to focus on their problem by making the Grid transparent to the user.

1.3. A Grid Solution: Web Portals

Web-based portals provide a possible solution to the Grid management problem by providing a web portal interface to Grids and allowing users to manage the Grid resources from a single user-friendly interface. A *Web Portal* is a website that provides a variety of resources and services to users inside a Portal framework, such as the Yahoo.com Web Portal (5) (6). With a web portal interface, users can quickly find the clusters and resources most suited for their specific requirements, without having to login or look at each resource individually. Web portals are made up of smaller parts, called *portlets*, which provide the information for a section of the total page. For example, in an email portal page there might be a portlet for sending email, one for viewing email, and a portlet that actually does the technical details of sending and receiving mail.

A *Grid Portal* is a Web Portal that provides access to Grid Services. These services include: security services, remote file management, remote job management, access to information services, application interfaces, and access to collaboration (7). The advantage of Grid Portals is that they provide a “natural way to incorporate ‘user-facing’ Grid services into the Portal environment” (7). “Although client tools are capable of providing the most direct and specialized access to Grid enabled resources, we consider the web browser itself to be a widely available and generic problem solving environment when used in conjunction with a Grid Portal” (3). Grid Portals hide the specific details of the Grid from the user and present an interface where the Grid is transparent to the user.

The UCLA (University of California, Los Angeles) Grid Portal (8) is a Grid Portal developed for the UC Campuses. The UCLA Grid Portal hardware consists of a *portal* machine and one or more *appliance* machines. The portal machine hosts the website through which the Grid Portal is accessed. The appliance machines are each connected to a cluster and to the portal machine. The portal is therefore able to submit jobs and manage data on each cluster. The UCLA Grid Portal supports PBS (9), LSF (10) and SGE (11) as cluster schedulers, which will be described in Chapter 2. The UCLA Grid Portal provides data management, job submission, and basic load and job information about each cluster. The UCLA Grid Portal further provides the user with access to all of their shared clusters and jobs in a single location so they can better manage files and jobs. However, the UCLA Grid Portal suffers from two limitations, which leads to our thesis questions.

1.4. Thesis Questions

This thesis addresses the following two research questions: 1) How should we extend the UCLA Grid Portal in order to provide more resources and better support independent jobs, and 2) How should we enhance the feedback that the UCLA Grid Portal provides to users?

1.5. UCLA Grid Portal: Limited Resources

The UCLA Grid Portal is a great start towards providing Grid Computing to the UC Campuses, but the resources it supports is limited. The UCLA Grid Portal supports some cluster schedulers, e.g. PBS, SGE and LSF, but support for other cluster schedulers is currently missing. One prominent cluster scheduler not supported is Condor (12). Condor clusters, also called *Condor Pools*, can grow to manage thousands of machines. This is primarily because a user can join a Condor Pool and the machine will be available in the Pool only when it is not in use. The

UCLA Grid Portal could substantially enhance the resources it has available by adding support for Condor as a cluster scheduler.

Additionally, there are some types of jobs that Condor manages better than other schedulers, namely mutually independent serializable jobs. Condor is capable of transparently check-pointing jobs for failure recovery or migration, supports more types of resources than other schedulers (desktops, clusters, etc), and has a dynamic resource pool that grows and shrinks transparently. Therefore, adding Condor clusters to the UCLA Grid Portal would both increase the amount of resources available in the Portal and enhance its functionality. Furthermore, Condor has many users that are currently using Condor, have executables optimized for Condor, and want to continue using Condor in the UCLA Grid Portal. Without Condor available in the Portal, these users would be limited in their ability to benefit from the UCLA Grid Portal.

Our first goal was to extend the UCLA Grid Portal by increasing the resources available in the Portal and adding functionality to the UCLA Grid Portal by providing support for Condor as a cluster scheduler.

1.6. UCLA Grid Portal: Limited Feedback

The UCLA Grid Portal enables users to submit jobs to many clusters, each with its own scheduling policy and current load on the cluster. The user faces the question: how do I decide where to submit my jobs?

Currently, the user would look at the Resources section of the Portal to find out the cluster load, number of jobs running, and number of jobs queued on each cluster. From there, the user would take his or her best guess about which cluster to submit jobs to. This is not a good solution because the user has to make job scheduling decisions with insufficient information, limited feedback from the UCLA Grid Portal, and no way to know which cluster is best for the job. Because users can not adequately decide where to submit jobs, this can also lead to load imbalances among the clusters connected to the UCLA Grid Portal. Finally, users have no way of knowing when their job will begin running other than to periodically check the Web Portal to see if the job has started. On many resources, jobs from different users are placed in a *queue* until scheduled.

The Queue Bounds Estimation from Time Series (QBETS) system, a member of the Network Weather Service (NWS), provides job monitoring and queue delay prediction capability to clusters. QBETS is capable of returning either an upper bound on the time a job will wait in a queue before running (queue delay) or the probability that a job will be scheduled before a deadline. Adding QBETS to the UCLA Grid Portal provides the user with an upper bound regarding when their job will start, which the user can use for job scheduling decisions and submit jobs to the clusters that will start their jobs the soonest.

Our second goal was to extend the UCLA Grid Portal by integrating QBETS into the Portal in order to provide queue delay predictions to users to help them decide where they should submit jobs based on *current* load conditions.

1.7. Approach To Integrating Condor with the UCLA Grid Portal

In this thesis we will show that it is possible to support clusters using Condor as a cluster scheduler in the UCLA Grid Portal. Our approach to integrating Condor as a cluster scheduler for the UCLA Grid Portal is the following. We first investigated an approach based on using a standalone Condor portlet that would create a Condor specification file (called a *description* file), transfer the description file via GridFTP to the appliance and then submit the job to the Condor scheduler from the appliance. The primary advantage of this approach was that the user had complete control over the Condor description file, so advanced Condor users would find the interface and control over their jobs to be similar to how they were currently creating Condor description files. However, the UCLA Grid Portal is based around using job managers provided through Globus (13), a Grid development toolkit. Using a Globus-based job manager to submit Condor jobs meant sacrificing some of the flexibility of Condor in the Portal in exchange for consistency with the rest of the Portal and ease of development. This was the approach we ultimately chose.

Cluster load, job, queue and machine information is provided by the appliance to the portal for all job schedulers. It was necessary to write index scripts in the appliance to report this information from Condor to the portal. We wanted the Condor integration to be transparent to the user, so the resource and job information from the index scripts must provide the same information to the portal as the scripts for the other cluster scheduler index scripts. Finally, we modified portal code to support Condor as a cluster scheduler.

1.8. Approach to Integrating QBETS into the UCLA Grid Portal

In this thesis we will show that it is possible to integrate QBETS into the UCLA Grid Portal. Our approach to integrating QBETS into the UCLA Grid Portal is separated into two distinct parts: what was necessary to integrate QBETS on the appliance machines and what was necessary on the portal machine. On the appliance, we find any new job information and send it to the QBETS backend prediction system. In the portal, we create a QBETS portlet. A portlet is a small part of the total portal with limited HTML capabilities. The QBETS portlet gets the list of clusters that the user has access to, intersects those clusters with the clusters known in the QBETS database, and gets a prediction from each matching cluster through a web service call to the Batch Queue Prediction Web service. The portlet then partitions the clusters into clusters that are currently alive in the portal (and clusters not alive) and then sorts the resulting sections by the prediction result.

1.9. Outline

Chapter 1 of this thesis introduces and motivates Grid Computing, what it is and why it is important. It discusses current Grid problems and how Web portals are helping to solve some of the Grid challenges. Next, we introduce the UCLA

Grid Portal, outline the approach of our work in this thesis, and explain how our work will enhance the UCLA Grid Portal.

Chapter 2 presents the background and related work in Grid Computing. We expand on Grid Computing and Web Portals and discuss the current state of the art in Grids and Web Portals. We review some necessary background for this thesis, including Globus, the UCLA Grid Portal, Condor and QBETS.

In Chapter 3 we cover the specifics of the thesis, with an in-depth discussion of the experimental appartus and the challenges that we faced integrating Condor and QBETS into the UCLA Grid Portal.

Chapter 4 describes our approach to integrating Condor and QBETS into the UCLA Grid Portal. We explain in detail how the integration works and how we overcame the challenges raised in Chapter 3.

Chapter 5 presents the results of this thesis, complete with screenshots of the UCLA Grid Portal in action and an explanation of what the main pages of the UCLA Grid Portal do and how to use each one.

In Chapter 6, we discuss the lessons learned from this thesis and future work. We cover a few of the behind-the-scenes technical problems that we faced and how we solved them and discuss the possibility of introducing Condor glide-in into the UCLA Grid Portal and enhancements to QBETS.

In Chapter 7, we conclude the work we have done in this thesis and review how our work enhances the UCLA Grid Portal.

2. Background and Related Work

In Chapter 2 we present the necessary background information and related work for this thesis. In 2.1 we discuss Grid Computing and in Section 2.2 we present a tool for building Grids, the Globus Toolkit. In 2.3 we cover Web Portals as a way to provide web interfaces for Grids. In section 2.4 we look at related work in Grid Computing. Section 2.5 introduces the UCLA Grid Portal. In 2.6 we present the details of Condor. And in section 2.7 we discuss the QBETS system.

2.1. Grid Computing

Large-scale computational Grids facilitate the sharing of resources by connecting clusters, supercomputers, storage and other resources together through the Internet. These resources are typically heterogeneous and are provided by different organizations. The term *computational Grids* refers to the infrastructure that enables the increases in shared resources such as computer cycles, data, sensors, etc. The important features of computational Grids are the Grid infrastructure, dependable service, consistency of service, pervasive access, and inexpensive access to the Grid. Together these features will "cause computational Grids to have a transforming effect on how computation is performed and used" (1). Computational Grids have the potential to dramatically increase the amount of storage and computational capability available to their users.

The architecture of a Grid is illustrated in Figure 1. The Grid in Figure 1 consists of three organizations connected together by the Internet. Each organization has its own resources shared on the Grid. The shared resources are optimized for each organization's goals, policies and security requirements and access to those resources is through the Internet.

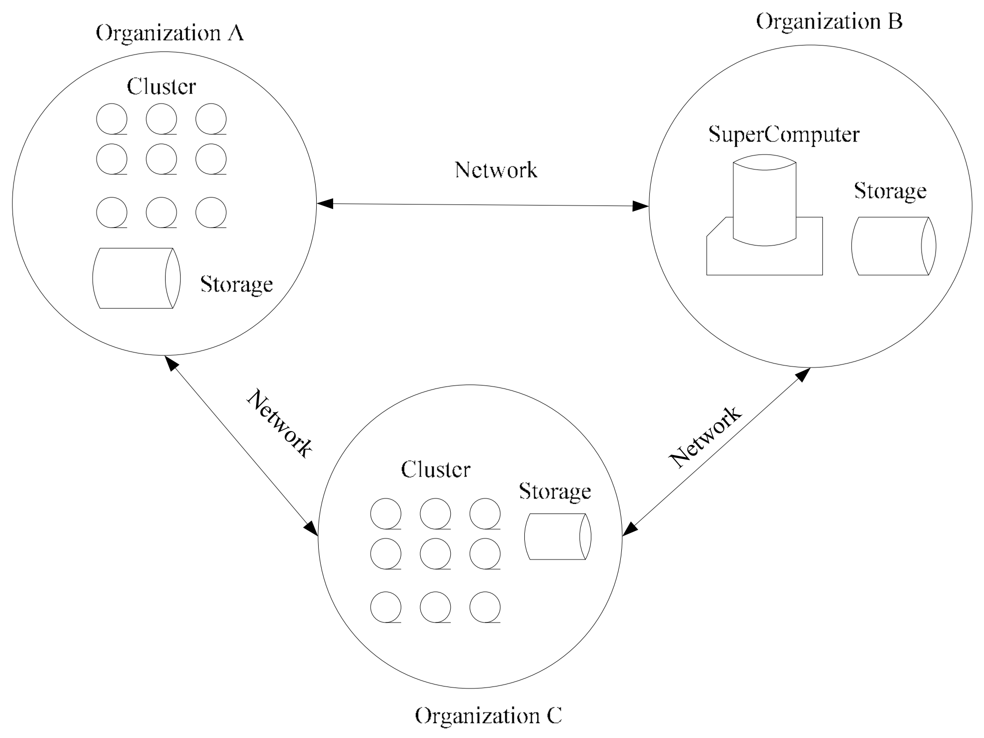


Figure 1: Architecture of a Grid

Computational Grids will be more useful and in higher demand in the future due to innovations in five key areas: technological improvements, increase in demand-driven access to computational power, increased utilization of idle capacity, greater sharing of computational results, and new problem solving techniques and tools (1). Behind each of these technical advances are the “synergistic use of high performance networking, computing, and advanced software to provide access to advanced computational capabilities, regardless of the location of users and resources” (1) Based on experiences in gigabit testbeds, the five major application classes that will be popular for computational Grids are: Distributed Supercomputing, High-Throughput Computing, On-Demand Computing, Data-Intensive Computing, and Collaborative Computing (1).

Computational problem solving has already proven effective for a wide variety of interesting problems: from modeling and simulations to chemistry, biology and medicine. However, scientists continually need access to more computational facilities to solve newer and bigger problems. Many projects get scaled down from the original design due to a lack of available computing power or resources (1). Grid computing provides a means for organizations to share resources without giving up autonomy over the resources.

2.2. Globus

Globus is a standardized Grid development toolkit that is used “as a fundamental enabling technology for the Grid. This allows people to share computing power, databases, and other tools through a secure online mechanism without sacrificing local autonomy” (14). The Globus Toolkit was first released in 1998 and was initially developed at Argonne National Laboratory, University of Southern Califronia’s Information Sciences Institute and the University of

Chicago. The Globus Toolkit provides a set of libraries and programs that address problems that occur when building distributed system services and applications (15).

Globus provides three broad features to the Grid developer. The first feature is a set of service implementations. These services address execution management, data access and movement (GridFTP, RFT), monitoring and discovery, and credential management. GridFTP is a File Transfer Program (FTP) designed specifically for Grids. The credential management feature of Globus provides support for storing, managing, transferring, and verifying security certificates. The notion of *GSI* (Grid Security Infrastructure) *certificates* in Globus is important, because every user and service is identified via a certificate. The motivations behind GSI certificates are a need for secure communication (authenticated and confidential) among elements of a computational Grid, a need to support security across organizational boundaries (prohibiting a centrally-managed security system), and the need to support "single sign-on" for users of Grid, including delegation of credentials for actions that involve multiple resources and/or sites (16). GSI provides four pieces of information: subject name to identify the person or object the certificate represents, the public key of the subject, the identity of the Certificate Authority (CA) that has signed the certificate to certify that the public key and the identity both belong to the subject, and the digital signature of the named CA. Certificates provide identity management and allow services to accept users in groups instead of having to individually accept users.

For example, a resource could choose to grant access to all users that have their certificates signed by a specific CA instead of having to individually grant access to each user that fit specific qualifications.

The second feature Globus provides is containers to host user-developed services written in Java, Python, or C. "These containers provide implementations of security, management, discovery, state management, and other mechanisms frequently required when building services" (15). The third feature Globus provides is a "set of client libraries [that] allow client programs in Java, C, and Python to invoke operations on both GT4 and user-developed services" (15).

The Globus Toolkit is useful because it provides a standardized set of Grid development tools to Grid developers. This means that different Grids will tend to have a similar underlying infrastructure which will allow Grids to be interconnected easier and expedite Grid development. Because of the advantages Globus presents, many Grids utilize the Globus Toolkit.

2.3. Web Portals

A Web Portal is a web site that offers a variety of resources and services such as email, search engines, online shopping, or news articles. More specifically, a Web Portal is a "Web application that provides every user their own individualized and personalized single place to access the information, applications and processes of an organization" (6). Examples of well known Web Portals are

Yahoo.com, Usa.gov and Amazon.com. Web Portals are "characterized visually by the dynamic arrangement of one or more windows of information and functionality on a single Web page" (6).

Web Portals are made up of individual parts, called *portlets*, which provide visual and functional support for the Portal. Portlets often consist of *Java Server Pages* (jsp) that typically handle the visual presentation of the portlet information and Java Classes that provide portlet functionality. Each portlet handles a small section of the Web Portal. The term Portal generally refers to the Portal Server or Engine which is responsible for "aggregating and laying out content from portlets on a page" (5). Portlets process requests and generate dynamic content on web pages and are collectively managed by a Portlet Container. Portlets generate "Markup Fragments" for display within the encompassing Portal page.

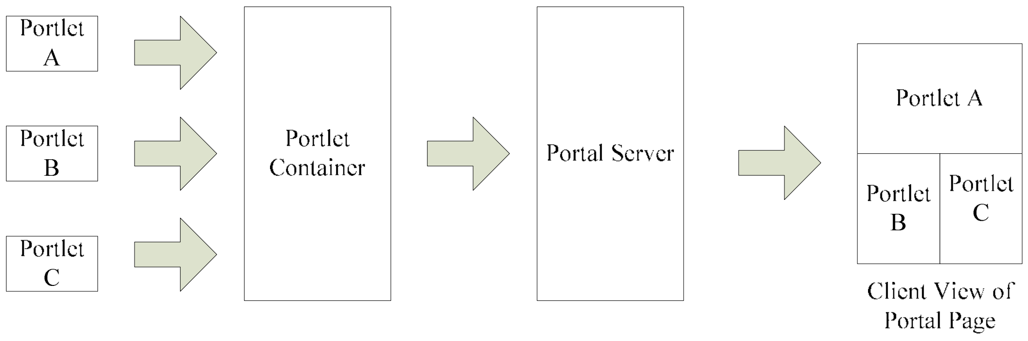


Figure 2: Creation of a Portal Page

Figure 2 presents the steps required for the creation of a page within a Portal framework. The content generated by the portlets is sent to the Portlet Container.

The Portlet Container then sends the merged content to the Portal Server. The Portal Server generates the page on the website, which is what the client sees. In Figure 2, each portlet provides the content for a small portion of the entire page.

Web Portals are designed to simplify access to diverse resources. Portals are readily extensible, easy to develop for, and easy to modify or change. Individual portlets can be developed and tested in isolation and then added to the overall Portal. In the commercial space, the Yahoo.com and Amazon.com Portals have proven successful at providing a variety of information to their users.

We have seen that Grids provide access to additional resources for their users. The following question should then be raised: Why are there not more Grids in production or development? A main problem facing Grid Computing is resource management and this is a problem that Web Portals can provide a solution for.

Grids are hard to manage because they are a conglomeration of heterogeneous resources provided by many different organizations, each of which has their own organizational policies, requirements and goals. Additionally, the interfaces provided to manage Grid resources provide specialized and direct access to complicated tools within a non-customizable interface. Thus, it becomes difficult for the user to have adequate job and resource management in the face of these difficulties (3).

Web Portals have the potential to solve Grid Management problems because they can provide users with an easy-to-use interface that can hide Grid-specific details and present information about the entire Grid in a single page accessible from the Internet. “The Grid has not yet become a disruptive technology in its own right. However, with the use of Web-based portals for the delivery of scientific, informational, experimental and computational services using a Grid infrastructure it may have the potential to be one” (5). A Web Portal that is used for Grid Services is known as a *Grid Portal*. “A Grid Portal is a problem solving environment that allows scientists to program, access and execute distributed ‘Grid’ applications from a conventional Web Browser and other desktop tools…The goal is to allow the scientist to focus completely on the science problem at hand by making the Grid a transparent extension of their desktop computing environment” (7).

Grid Portals provide the functionality of Grid Computing without the management and organizational difficulties common to Grids. Grid Portals make distributed, heterogeneous Grid environments more accessible to users by utilizing common Web and UI conventions and provide users with the capabilities to customize the content and presentation (e.g. page layout, level of detail) for the set of tools and services provided in the portal (7). Due to their advantages, Grid Portals have become a popular way to present Grid resources to users in a single, comfortable interface. Some of the contemporary Grid projects will be discussed next.

2.4. Grid Computing Projects

The most ambitious and well known Grid in the United States is the TeraGrid (17), which includes resources from Indiana University, Oak Ridge National Laboratory, National Center for Supercomputing Applications, Pittsburgh Supercomputing Center, Purdue University, San Diego Supercomputer Center, Texas Advanced Computing Center, University of Chicago/Argonne National Laboratory, and the National Center for Atmospheric Research. The TeraGrid joins resources at nine sites through an "ultra-fast optical network, unified policies and security procedures and a sophisticated distributed computing software environment" (18). "The TeraGrid provides more than 102 teraflops of computing capability and more than 15 petabytes (quadrillions of bytes) of online and archival data storage" (19). The TeraGrid was designed according to the following core policies: users should have a single identity even with multiple computer accounts, a single accounting currency, the "TeraGrid Service Unit", a unified support team, a unified infrastructure, and a unified documentation and training "programme" (18). The single identity per user is provided by the use of the certificates that were discussed with Globus in 2.2. The TeraGrid uses a "coordinated set of Grid middleware, including the Globus Toolkit, OpenSSH, SRB, Myproxy, and Condor-G" (18). The TeraGrid is a large and successful Grid, however it requires a unified architecture and a single currency. The TeraGrid is appropriate for large supercomputing sites, but its model is not feasible for a diverse group of heterogeneous clusters that each want to keep cluster architecture and scheduling the same, but also share resources on the Grid.

The Java CoG kit (20) is a project to help develop *Commodity Grids (CoGs)* with "twin goals of (a) enabling developers of Grid applications to exploit commodity technologies wherever possible and (b) exporting Grid technologies to commodity computing (or, equivalently, identifying modifications or extensions to commodity technologies that can render them more useful for Grid applications)" (20). The Java CoG kit is a Grid Toolkit that "defines and implements a set of general components that map Grid functionality into a commodity environment/framework" (20). The mappings provided by the Java CoG Kit are: Low-Level Grid Interface Components such as resource management services and data access services, Low-Level Utility Components such as information service functions or the Globus Job submission language, Common Low-Level GUI Components such as editors, browsers and search components, and Application-specific GUI components to simplify the "bridge between applications and basic CoG kit components, examples of which are a stockmarket monitor, a graphical climate data display component, or a specialized search engine for climate data" (20). The Java CoG kit is similar to the Globus Toolkit in purpose, however it is developed to bring the commodity environment onto the Grid instead of connecting Grids together.

Legion (21) (22) is an object-based metasystem developed at the University of Virginia. Legion provides software infrastructure to allow a system of heterogeneous, geographically distributed, high performance machines to interact seamlessly. Legion attempts to provide users with a single coherent virtual machine at their workstation. The Legion Grid Portal (23) depends on the Legion infrastructure for managing a Grid. It presents the entire Grid as a single virtual machine to users and presents a "truly distributed file system" (23). A unique feature of Legion is that every "first-class entity" in Legion, such as files, directories, machines, disks, users, consoles, programs, etc. is an object. Legion is an interesting Grid Portal, however we believe that it is desirable for the user to know where their jobs are running and details about the underlying architecture. We present Legion here as an example of one of many Grid Portals in development and discuss the Grid Portal we chose to work with, the UCLA Grid Portal, next.

2.5. UCLA Grid Portal

The UCLA Grid Portal is a Web Portal interface to a Grid that will be set up on each UC Campus. Additionally, there will be a UC-Wide Grid that will be shared with each campus Grid. The UCLA Grid Portal is being developed for the UC System by the UCLA Grid Team with Academic Technology Services at UCLA.

The UCLA Grid Portal increases the amount of shared resources available to students and faculty members by allowing convenient access to a variety of different resources with a single login and from an easy-to-manage website. The

UCLA Grid Portal will allow departmental clusters to be shared to a specific department, UC campus, or UC wide through security certificates. The department can specify which certificates they want to give access to and only users with those certificates may access the resource. The UCLA Grid Portal allows users to access shared clusters, create, transfer, upload and delete files, move files between clusters, submit and check the status of jobs, check the load and status of clusters, and otherwise manage job submission and data among a variety of different clusters.

The UCLA Grid Portal architecture consists of a single *portal* machine and one *appliance* machine for each cluster that is connected to the UCLA Grid Portal. Figure 3 shows this architecture. The users communicate directly with the portal machine, which communicates with each appliance machine. Each appliance is connected to its respective cluster headnode to access the cluster scheduler and the users' home directories. A cluster *headnode* is the "entry point into the cluster [which] receives all jobs submitted to the compute cluster. The headnode hosts the cluster's scheduler service, whose role is to match the resources requested by the submitted jobs to resources available across the cluster" (24). To add another cluster to the UCLA Grid Portal requires connecting an appliance machine to the portal machine and cluster headnode.

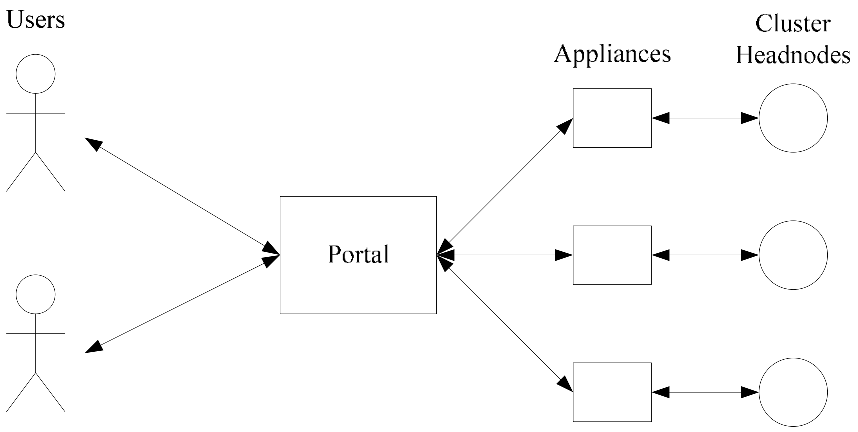


Figure 3: Architecture of UCLA Grid Portal

The communication between the portal machine and appliance machines is accomplished through the Globus, which runs on both the portal and the appliance machines and provides communication through GridFTP. New clusters are added to the Portal in the Web Portal interface by providing the Web Portal with the appliance node, name of the cluster, cluster headnode, and scheduler used by the cluster.

We see the unique contribution of the UCLA Grid Portal as its ability to hide the complexity of the Globus installation from the user and the cluster administrator through its portal and appliance architecture. If a cluster wants to join the UCLA Grid Portal, it only needs to setup an appliance machine, connect it to the cluster headnode, and then add the cluster to the UCLA Grid Portal. Moving the Grid software off the cluster headnode and onto the appliance machine allows the UCLA Grid Portal to provide a "plug-n-play" installation. Outside the UCLA

Grid Portal, to install a Grid Portal or other Grid often requires installing Globus on the cluster headnode, a time consuming and often undesirable task, in addition to installing the Grid software itself.

The difficulty with installing Globus on a cluster headnode is due to the fact that it is usually not a “fresh” install and the headnode could be using many possible hardware configurations. The cluster headnode is also hosting the cluster scheduler, storing scheduler log files and managing job submission. Therefore, it is not desirable to remove all files from the cluster headnode and setup Globus and the headnode on a fresh machine. Because of this, trying to install Grid software into a cluster headnode involves trying to merge the Grid requirements with the pre-existing cluster scheduler installation. Additionally, some secure clusters do not want to expose the cluster headnode to potentially unsafe code and would not want the Grid software to run on the headnode. With the UCLA Grid Portal this requirement does not present a problem; the cluster only has to give the appliance limited access to the cluster headnode. Furthermore, in the UCLA Grid Portal, the appliance-style installation comes with Globus already installed on the portal and appliance machines and it is relatively easy to add clusters to the UCLA Grid Portal. Now that we have seen that it is easy to install the UCLA Grid Portal and add clusters to the Portal, we present the details of how the portal and appliance relationship in the UCLA Grid Portal works for communication, job submission, and what functionality is provided by cluster schedulers.

For communication, the portal machine finds out cluster information from the appliance that is connected to each cluster. The appliance runs index provider scripts (background processes) that publish cluster-specific information to a Globus-provided URL that the portal machine periodically queries for updates. The index provider scripts are customized for each supported cluster scheduler, currently SGE, LSF, and PBS. The index provider scripts expose scheduler information to the user in the portal machine. Some specific information that is exposed is: cluster load, total nodes, number of free nodes, number of jobs running, number of jobs queued, machines in the cluster and their characteristics, any specific job requirements of the cluster, and other cluster specific information.

For job submission, the UCLA Grid Portal enables users to submit jobs to clusters through the Web Portal interface. From the user's perspective, the user selects a cluster to submit a job to and specifies the various job parameters. When the UCLA Grid Portal processes a job submission the following steps happen, as shown in Figure 4. The user creates a *GRAM* (Grid Resource Allocation and Management) file with the various job parameters. The portal machine passes the GRAM file through GridFTP to the appliance machine connected to the cluster where the job is submitted to. Globus, running on the appliance, takes the file and looks at the field specifying the cluster scheduler and determines which job manager to use (Condor, PBS, LSF, SGE). The respective job manager generates a scheduler submit file, called a *description file*, with the job details in the specific form the cluster scheduler requires and submits the description file to the scheduler.

The scheduler then schedules and runs the job according to its scheduling policies and reports the job's *stdout* or *stderr* back to the Portal. One requirement of this implementation is that the executable file has to be present on the cluster where the job is submitted to. The Data Manager inside the UCLA Grid Portal allows users to move files between clusters to satisfy this requirement. The Data Manager also provides a way for the user to create file files, delete files, upload files from the current computer, and otherwise provides file management among the clusters that a user has access to in the UCLA Grid Portal.

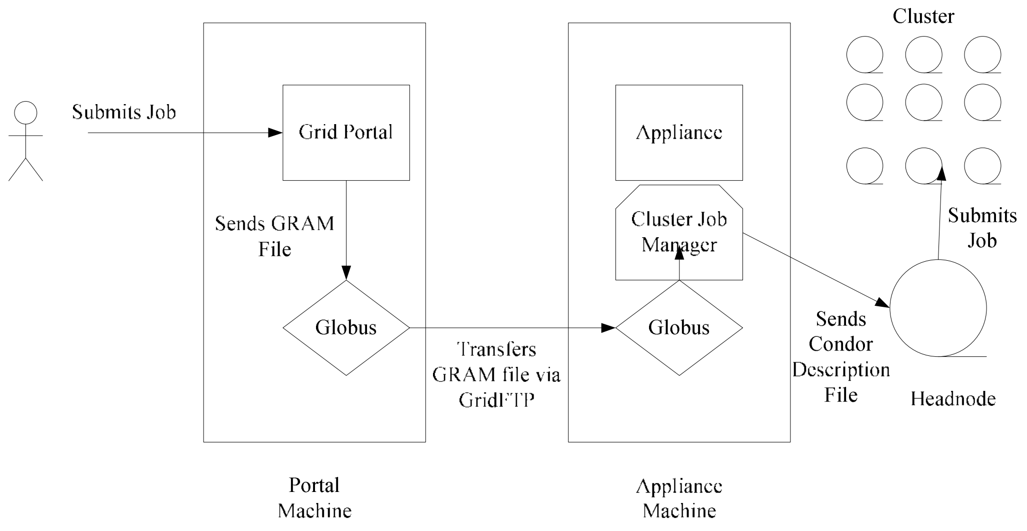


Figure 4: Job Submission in the UCLA Grid Portal

The UCLA Grid Portal supports three cluster schedulers: PBS (Portable Batch System), SGE (Sun Grid Engine), and LSF (Load Sharing Facility). All three cluster schedulers, also called *batch* schedulers, are similar. We will describe the functionality provided by PBS, but these features are also found in SGE and LSF.

PBS is a batch job and resource management system. By *batch*, we mean jobs that can be executed on a computer without human interaction. It will "accept batch jobs (shell scripts with control attributes), preserve and protect the job until it is run, run the job, and deliver output back to the submitter" (25). PBS provides the following features: automatic load-leveling, file staging, job interdependency (execution order, conditioned execution, synchronization), security and authorization, username mapping, parallel jobs support, job accounting, and a comprehensive API to write new commands, integrate PBS with applications, and implement scheduler policies (25). PBS is primarily used to manage parallel jobs run on large clusters but it is equally capable of managing serial jobs.

PBS provides two main commands to get cluster information: `qstat` and `pbsnodes`. `Qstat` provides queue and job information about the PBS cluster and can be used to find out about queue load and the jobs that are either executing or waiting to execute. `Pbsnodes` provides machine specific information about each machine in the cluster and can be used to see the status and architecture of specific machines.

In short, PBS allows the user to submit batch jobs to a scheduler and PBS will schedule the jobs on the machines it manages and return the output of the jobs to the user. PBS is typically used in a run-jobs-for-specified-time scheduler paradigm, where the user requests a PBS timeslot for a certain time, and at the end of the timeslot the job is killed if it has not finished. In contrast to PBS, the functionality provided by Condor is discussed next.

2.6. Condor

Condor is a specialized cluster scheduler and job manager that is optimized for multiple independent serializable jobs and compute-intensive jobs. Condor was developed by the University of Wisconsin-Madison and the first Condor production system was deployed in the late 1980s. Currently, in the UW-Madison Computer Science Department, Condor manages more than 1000 workstations (26) and has been installed on thousands of machines elsewhere.

Condor is different from other schedulers such as PBS, LSF, and SGE because it was originally developed to utilize CPU cycles from idle workstations, whereas the other schedulers are primarily batch schedulers for parallel jobs that run on clusters. "Like other full-featured batch systems, Condor provides a job queuing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management" (26). Condor operates under a run-when-free paradigm versus the run-for-a-specific-time paradigm that other batch schedulers follow. This means that when a job is submitted to a Condor Pool it will run whenever there are free resources, and wait when all resources are in use. Condor jobs running in the Condor *Standard* universe, a universe that requires executables to be compiled by the condor_compile command, can transparently checkpoint jobs when resources are no longer available and restart the job from the checkpoint as resources become free.

Condor submits jobs to a group of machines that are connected together by Condor, called a Condor Pool. A Condor Pool consists of a central manager and any number of other machines that have joined the pool. Conceptually, a Condor Pool can be viewed as a collection of resources (machines) and resource requests (jobs) (26). There are three roles that a machine can have in a Condor Pool: central manager, execute node, and submit node. Machines (nodes) can possess multiple roles, i.e. many machines will probably be both execute and submit machines. The first role is the *Central Manager*, of which there is only one in the Pool. The Central Manager is the collector of job and node information and the negotiator between resources and resource requests. The second role is *execute machine*, which are the machines that are configured to accept and execute Condor jobs. There must be at least one execute machine in a Condor Pool in order for jobs to run. The third role is *submit machine*, which are the machines where Condor jobs can be submitted from.

For job submission, a user submits a job at a submit machine. The submit machine sends the job information to the Central Manager, which then acts as a matchmaker between jobs and free nodes. When it finds a match it tells the submission node and the free node about each other and then those two nodes communicate to execute the job. The user submits a job specification file at the submit machine, called a *Condor Description* file. The description file specifies the job name, location of the executable, where the *stdout* and *stderr* should be written, job arguments, the number of times the job should be run, and other job submission parameters. The job then runs on the Condor Pool as long as resources are available.

Condor Pools are typically large because machines can be added to a Condor Pool without degrading the user's performance. Additionally, any machine capable of running Condor can be part of the Condor Pool. Thus even personal desktop computers and laptops can be members of the Condor Pool and provide resources whenever they are not in use. The user is willing to share their workstation because they experience no degradation of performance by joining a Condor Pool. Condor facilitates cycle stealing in the following manner: when a machine is not in use then it is considered free and available to be used by Condor and Condor will schedule jobs on the machine as long as it remains free. When the computer is considered used again (the mouse or keyboard is touched), Condor will checkpoint the job if possible or kill the job if necessary and the job will be sent back to the Condor scheduler to be rescheduled. This design for cycle stealing encourages users to share their machines with Condor because users can benefit from a Condor Pool when they need additional resources, and share their own machines in the Pool without experiencing any performance loss.

Even though Condor Pools are large, users might still require access to more resources. In this case, Condor has a feature called *Flocking* that enables jobs to "flock" between multiple Condor Pools. Flocking allows a Condor master node to send jobs to other Condor Pools when the current Pool is busy. Flocking effectively merges Condor Pools together and increases the resources available to all merged Pools.

Like other cluster schedulers, Condor provides status commands to determine the state of the Pool, the individual machines on the Pool, and jobs submitted to the Condor Pool. These commands are used by Condor and its users to evaluate the load on the Pool and determine if flocking is necessary. `Condor_q` loosely compares to the `qstat` command provided by other cluster schedulers and `condor_status` loosely compares with `pbsnodes`. Condor provides the `condor_q` command to view the status of all jobs running, scheduled, and held on the cluster, the name of each job, when the job was submitted, the current state of the job, and other relevant job information. The other main Condor command, `condor_status`, provides information about the machines in the Condor Pool. Specifically, `condor_status` returns how many machines are free, used, or are running jobs and the architecture of each machine in the Condor Pool. The information reported by `condor_status` is used by Condor to find matches between the requirements of jobs and the requirements of machines.

Figure 5 shows how the Condor Matchmaking Process works. Jobs and machines advertise with the *Condor Matchmaker* (part of the Condor Master node) and the matchmaker finds a match based on a negotiation algorithm and sends a notification to both jobs and machines. The jobs and machines claim each other and communicate directly while running the job. If the owner of a machine claimed by a job returns, the job gets check-pointed, advertises its status with the Matchmaker, and waits to be scheduled with a new machine that matches the job's requirements.

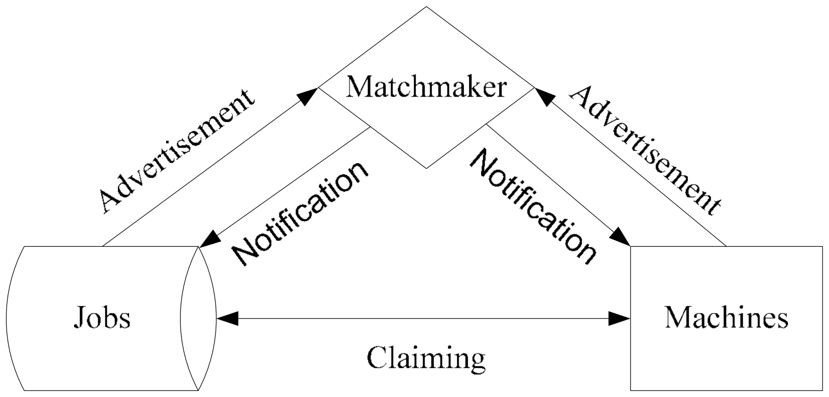


Figure 5: Condor Matchmaking Process

Machines and jobs advertise with the Condor Matchmaker through "Classified Advertisements" (*ClassAds*), which are reported by `condor_status`. ClassAds are used for describing jobs, workstations, and other resources, exchanged by Condor processes to schedule jobs, logged to files for statistical and debugging purposes, and to enquire about the current state of the system (27). ClassAd matching is used by the Condor *central manager* to determine the compatibility of jobs and nodes where they may be run (27). ClassAds are to be considered the *"lingua franca* of Condor" (27), a language that is used to communicate with others, who speak a different, unknown language.

2.7. Queue Bounds Estimation from Time Series (QBETS)

The Queue Bounds Estimation from Time Series (QBETS) system (28), a part of the Network Weather Service, consists of two parts: a *monitoring system* deployed on clusters and a *backend prediction system* that processes the data recorded from the monitors to generate predictions. The monitoring system consists of sensors and monitors that collect the job accounting information, e.g. by parsing log files of the job scheduling system, and send the information to the backend prediction system. The prediction system is constantly processing the accounting information for each cluster queue pair and updating predictions. QBETs makes queue delay predictions for jobs submitted to specific clusters and a queue for that cluster based on both past and current queue delays that similar jobs have experienced. QBETS provides two types of predictions: *wait time* prediction and *deadline* prediction. In *wait time*, given a job's characteristics, QBETS will return an upper bound on the time a job will spend in the queue before execution at the specified confidence level, either 95, 75, or 50 percent. In *deadline*, given a job's characteristics and a deadline, QBETS will return the probability that the job will start by the deadline.

Due to the architecture of Condor and the way nodes become free and used, jobs are scheduled, check-pointed, moved among machines, and priority given to individual users, QBETS cannot currently make predictions on queuing delays for Condor Pools as a whole. QBETS is currently deployed in the Teragrid, and several other Grids to provide predictions for PBS, LSF and SGE schedulers.

3. Challenges

In Chapter 3 we present the main problems addressed by this thesis and the challenges that came up in the course of the work. Section 3.1 presents and motivates the thesis questions and the high-level approach to those questions. In 3.2 we discuss the experimental apparatus used in this thesis. Section 3.3 presents the challenges involved with adding Condor as a cluster scheduler to the UCLA Grid Portal. In 3.4 we cover the challenges of integrating QBETS into the UCLA Grid Portal.

3.1. Research Questions and Motivation

This thesis addresses the following research questions: 1) How should we extend the UCLA Grid Portal in order to provide more resources and better support independent jobs, and 2) How should we enhance the feedback that the UCLA Grid Portal provides to users? First, we will consider the motivation for our research questions. Second, we will explain how we chose to address the research questions raised.

The UCLA Grid Portal currently provides support for PBS, LSF, and SGE as cluster schedulers, adding the resources managed by those schedulers into the UCLA Grid Portal. It would be ideal for the UCLA Grid Portal to support as many schedulers as possible in order to maximize the amount of resources available.

However, it is not feasible to support all possible cluster schedulers, but it is feasible to add support for additional cluster schedulers. We chose to add support for Condor both because it is a popular scheduler, because it better supports some job types than the current facilities of the UCLA Grid Portal, and because it operates under a different model than the supported schedulers, as discussed in Section 2.6. The challenges of supporting Condor as a cluster scheduler will be discussed in section 3.3.

The UCLA Grid Portal currently provides limited feedback to users regarding cluster conditions. It provides basic feedback to users trying to decide where they should submit their jobs, specifically, cluster load, running jobs, and waiting jobs at each cluster. This feedback is helpful, but limited in its effectiveness for making scheduling decisions. It would be ideal if the UCLA Grid Portal could provide users with a statistical prediction concerning which cluster is most likely to start their job the soonest. As discussed in Section 2.7, QBETS provides an upper bound on the queue delay a job will have at a specific cluster. As far as we know, QBETS is the only system available that is able to predict queuing delays based on current and past cluster conditions with a high degree of accuracy. The challenges of integrating QBETS into the UCLA Grid Portal will be discussed in section 3.4.

3.2. Experimental Apparatus

The December 2006 version of the UCLA Grid Portal was installed on two machines: one portal machine and one appliance machine. The portal machine is a 2.4GHz Pentium 4 with 1024Mb of physical memory. The appliance machine is a 2.4GHz Pentium 4 with 256Mb of physical memory. The operating system on both machines is Fedora Core 4.0 and the database is mySQL version 14.7. The UCLA Grid Portal provided Globus 4.0, Apache-tomcat-5.5.17 as a servlet container, apache-ant-1.6.2 as a compilation tool, Gridsphere (29) as a portal framework (Gridsphere is a free Portal framework similar to IBM's WebSphere), and JDK-1.50_05. We tested Condor with a cluster running Condor version 6.7.14 on the cluster machines.

3.3. Extending the UCLA Grid Portal to Provide More Resources

There were three kinds of challenges we faced while adding Condor to the UCLA Grid Portal. First, we had to decide on a general approach for adding Condor clusters to the UCLA Grid Portal. Second, we faced challenges to support Condor in the appliance. Third, we faced challenges adding Condor to the portal machine.

The first challenge we faced integrating Condor into the UCLA Grid Portal was deciding on the general approach to take. We saw two feasible approaches: using a standalone portlet or using the Condor job manager provided by Globus.

A standalone portlet would allow the user to directly create Condor description files and directly access Condor commands concerning the status of the Condor Pool, such as `condor_q` and `condor_status`. Using the Condor job manager provided by Globus would allow the user to access a Condor Pool without needing to know anything about Condor or how to submit Condor jobs. Globus would provide the tools to translate job specifications into a Condor description file and submit the job to the Condor Pool. We ultimately chose to use the Condor job manager within Globus for reasons discussed in 4.1 and then faced the challenges of supporting the Condor job manager in both the appliance and the portal.

The main challenge we faced in the appliance to integrate Condor into the UCLA Grid Portal was the creation of index provider scripts that provide cluster scheduler information to the portal machine, as discussed in Section 2.5. These scripts are customized for each cluster scheduler. The UCLA Grid Portal had created scripts for PBS, LSF and SGE, but we had to create the index provider scripts for Condor.

While creating index provider scripts for Condor, we encountered several complications. First, the scripts need to report the same information in same format as other cluster schedulers for the portal to parse the information and display it on the Web Portal. `Condor_q` and `condor_status` loosely compare to commands provided by other cluster schedulers, however, both the information and format is significantly different from the other cluster schedulers. Some information, such as the notion of queues, is not directly relevant in Condor because Condor supports Pools that dynamically, and transparently, grow and shrink as resources join and leave the Pool. The challenges creating Condor index scripts that are compatible with the portal machine was finding out what the provider scripts should report to the portal, getting the relevant information from Condor and putting the information in a form that the portal machine understands. The index scripts provide the means for information to pass from the appliance to the portal; next we discuss the challenges faced passing information from the portal to the appliance for job submission.

The main challenge for integrating Condor into the portal section of the UCLA Grid Portal is in finding the most natural way to present Condor in the Portal interface and tying the presentation logic to the back-end logic on the appliance. The answer to this question depended on the general approach chosen to integrate Condor into the UCLA Grid Portal. If a standalone portlet approach had been used, then Condor job submission would have been different than other cluster schedulers and would require a Condor portlet to be developed that could be used for job submission and running Condor commands. Because we used the Condor job manager within Globus, the challenge was providing support for the requirements of the Condor job manager that are not already handled by the UCLA Grid Portal, specifically *Condor* and *multiple* job types.

3.4. Extending the UCLA Grid Portal to Provide More Feedback

There were two kinds of challenges we faced integrating QBETS into the UCLA Grid Portal in order to provide more feedback to users. The first challenge was installing and enabling the monitors that QBETS uses on the appliance machine. The second challenge was building a QBETS portlet on the portal machine that gets predictions from the QBETS backend database and outputs those results to the user in a clear and understandable manner.

In the appliance, QBETS gets job accounting information from the cluster scheduler and sends the information to the QBETS backend database. The main challenge with the appliance integration is that we do not want QBETS to require any additional information over what is already needed to set up the appliance. This requirement came from the desire to seamlessly add QBETS into the UCLA Grid Portal and minimize the additional work required by the system administrator to support QBETS. In order to integrate QBETS in the appliance without needing more information we had to find out what cluster scheduler is used on the appliance, find the log files for the jobs submitted to the cluster scheduler even though the root location of the scheduler is controlled by the user, parse job information efficiently from the log files, and send job information to the QBETS backend database.

After the job information is sent to the QBETS backend database, QBETS generates predictions based off the job accounting information and the predictions are displayed to the user in the Web Portal. The main challenge integrating QBETS in the portal machine was building a User Interface (UI) that is fast, consistent visually with the Portal, and returns comprehensive prediction information with a single query.

4. Approach

In Chapter 4 we present our approach to solving the challenges brought up in Chapter 3. Section 4.1 discusses our approach to adding Condor to the UCLA Grid Portal and in section 4.2 we cover our approach to integrating QBETS into the UCLA Grid Portal.

4.1. Overcoming Condor Challenges

There were three kinds of challenges we faced adding Condor to the UCLA Grid Portal: the challenging of choosing the general approach, challenges in the appliance with index provider scripts, and challenges in the portal machine with job submission.

When considering our general approach to adding Condor to the UCLA Grid Portal, we saw two viable options: creating a Condor standalone portlet in the UCLA Grid Portal or using the Condor job manager in Globus. We initially built a prototype Condor standalone portlet before we had access to the UCLA Grid Portal, and planned on adding the portlet directly into the UCLA Grid Portal. Figure 6 shows how job submission works with a Condor standalone portlet. The user submits a job in the Portal, which creates a Condor Job Description file, and the Description file gets sent from the Portal to the appliance through Globus' GridFTP service. Once the Condor Job Description File is on the appliance, it is sent to the cluster headnode, which then submits the cluster scheduler.

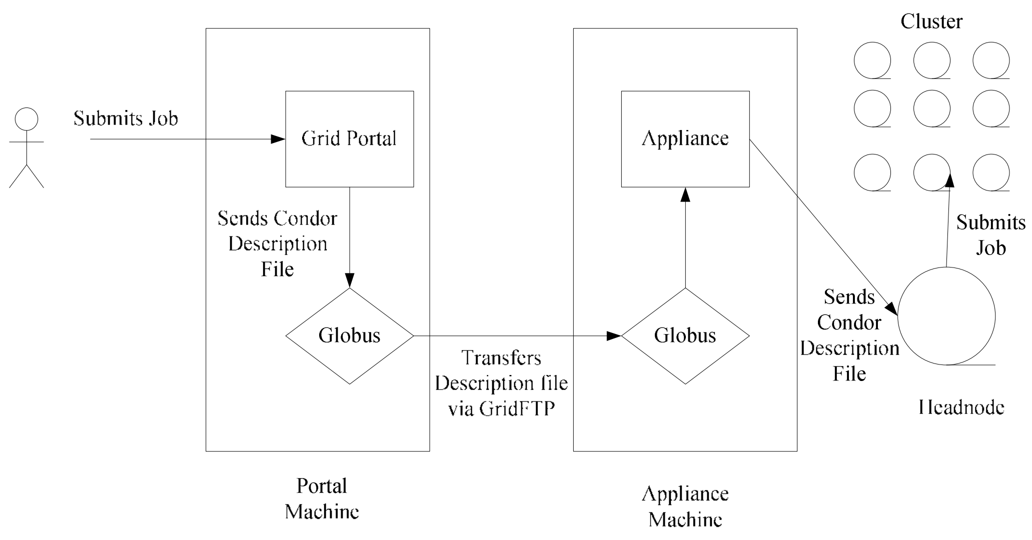


Figure 6: Job Submission using Condor Standalone Portlet

In contrast, Figure 7 shows how job submission would work using the Condor Job Manager in Globus. The user submits the job in the Web Portal and the Portal creates a GRAM file, as discussed in Section 2.2 regarding how the UCLA Grid Portal handles job submission for all cluster schedulers.   The Portal sends the GRAM file to the appliance through GridFTP and Globus passes the GRAM file to the job manager specified in the GRAM, which uses the GRAM to create a Condor Description file. The Condor Description file is then submitted to the cluster scheduler for execution.

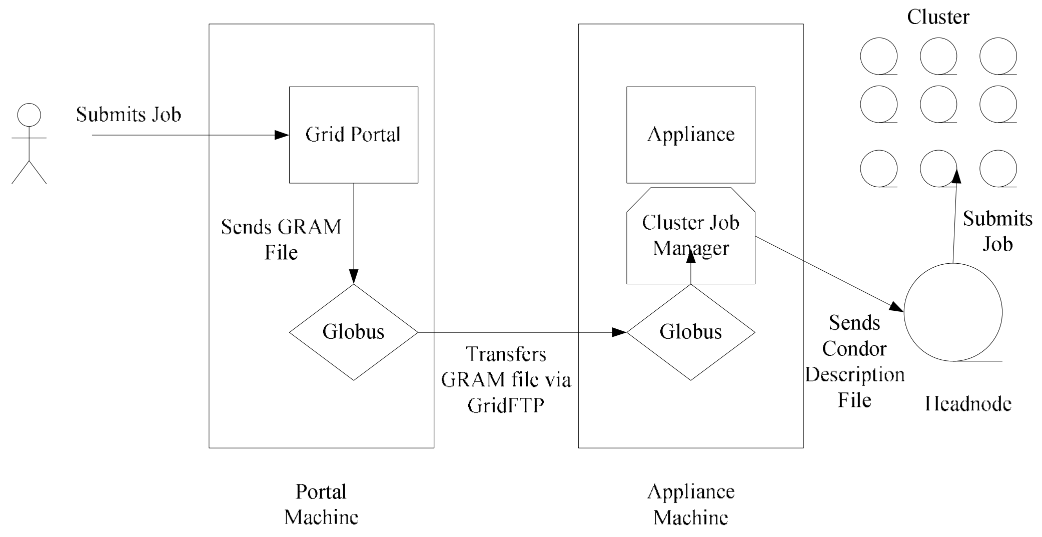


Figure 7: Job Submission using Condor Job Manager

Using the Condor job manager provided by Globus sacrificed some of the flexibility of allowing the user to create their own Condor description files, but enables Condor job submission to be consistent with other cluster schedulers and is easier for the novice user to use. As a result of using the Condor job manager, it turned out that a lot of the work done to build a prototype standalone Condor portlet was not needed to integrate Condor into the UCLA Grid Portal because the Condor job manager and the preexisting Portal code provided many of the necessary features for integration. In the appliance, the only changes needed to support the Condor job manager, other than the actual installation of the Condor job manager within Globus, was the creation of Condor index provider scripts to communicate cluster scheduler information from the appliance machine to the portal machine.

The Web Portal displays cluster information, provided by the appliance machines, in the Web Portal interface. Figure 8 shows how the Portal gets information about shared clusters. The index scripts running in the background on each appliance get cluster information from the cluster headnodes and then publish the information to a URL. The Portal periodically queries the URL for each appliance and displays the information to the user.

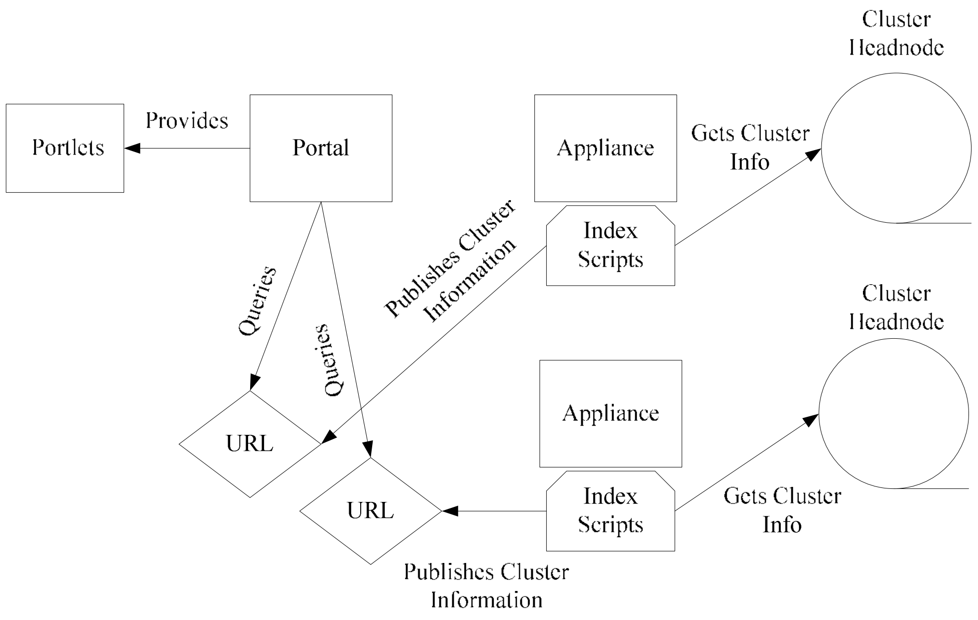


Figure 8: How the Portal Gets Information about each Cluster

The information that the index scripts publish to URLs is formatted with XML tags. This method of using XML tags enables the UCLA Grid Portal to ignore formatting differences among cluster schedulers and focus on providing the correct information to the Portal through index provider scripts.

There are seven index provider scripts that we needed to create for Condor: globus-info-provider, globus-load-provider, globus-job-provider, globus-jobdetail-provider, globus-local-provider, and globus-queue provider. For each script, we used a corresponding PBS script as a template. The PBS commands pbsnodes and qstat roughly match up with condor_status and condor_q. Condor_status returns information about the machines in a Condor Pool and condor_q returns information about jobs scheduled in the Pool. Each script provides information for a specific part of the UCLA Grid Portal Resource Page, thus there is some overlap between index scripts.

Globus-info-provider returns the CPU load on a cluster, total nodes, free nodes, downnodes, running jobs, pending jobs, and peak performance. In order to get this information, we used Condor_status to get the total nodes, free nodes, and cluster load. One difficulty with using condor_status was that it returns an entry for each CPU in the pool, and we only want the number of nodes in the Pool. To account for this we hash each new machine to a hash table and check to see if the machine has already been seen. If it has, then we skip to the next entry. Any nodes that are down are not returned by condor_status, so the down nodes field is always of value 0. Running and pending jobs are found directly from condor_q. Peak performance, measured in peak Gflops, is statically set in the PBS provider script so we also statically set peak performance with the Condor version of globus-info-provider.

`Globus-load-provider` returns the CPU load, architecture, number of processors, physical memory, memory-in-use, total swap space and free swap space for each machine in the cluster. We used `condor_status` to get this information from the Condor scheduler. However, Condor does not report actual memory-in-use so we looked at the memory that was used on the machines with the unix *free* command and in almost all cases the memory-in-use was close to all physical memory available. Additionally, by default, Condor considers all physical memory of a machine as available to be used even though some memory is used for the operating system and background processes. We decided to report the used physical memory as being equal to all available physical memory as reported by *free*, as this was the most common case when evaluating the memory usage of machines on our experimental cluster. This memory usage information is only used when the user looks at the load link in the Resource Page of the UCLA Grid Portal.

`Globus-job-provider` returns information about each job submitted to the cluster, specifically the job ID, job name, queue name, requested memory, requested time, status of the job, start time, and elapsed time. This information can be readily obtained from `condor_q`.

`Globus-jobdetail-provider` returns all information from `condor_q` with the command `-long` option on a specific jobID to the user. It is used to get detailed information about a specific job. We return all information reported by `condor_q -long` directly to the user.

`Globus-local-provider` returns the hostname of the appliance machine, the home directory of the cluster scheduler and the binary directory of the scheduler. The information is found from the Linux commands: `hostname`, `which`, `dirname`, and the Globus user's environment.

`Globus-queue-provider` returns information about the queues that are available in a cluster. Because Condor does not have queues, there were two available options of simulating queues in Condor. One is to treat a Condor Pool as a queue; the other is to treat each individual machine as a queue (because you can specify which machine to submit jobs to). We decided that we should represent each Condor Pool as a queue to stay consistent with what was represented for other schedulers and because the average user is more concerned with information about the Condor Pool as a whole than individual machines. We believe it would not be as helpful to provide a list of individual machines as queues to the user and instead believe the user is more interested in knowing what machines and processors are available to a Condor Pool as a whole. We report the queue name, max CPU time, max wall clock, max required CPUs, max total memory, max single memory, total and free cpus, total nodes, running jobs and waiting jobs. We set the max required CPUs to one because the Condor job manager with Globus 4.0 does not support parallel jobs. We set the various job requirements (Max WallClockTime, Max Total Memory, etc) to be unlimited (the default values) because those parameters are specified on a per machine basis and not as one setting for the entire Pool. We get the queue name from the Condor config file and use the Condor Pool name. We get the other parameters directly from `condor_q` and `condor_status`.

Now that the UCLA Grid Portal is capable of providing cluster scheduler information from the appliance to the Portal, we need a way to submit jobs to Condor clusters from the Web Portal interface. Because we used the Condor job manager provided by Globus, we are able to use the same interface for Condor job submission that the Portal uses for other cluster schedulers. Additionally, the majority of the code to support a Condor job manager is already provided in the Portal because specifying the cluster scheduler a job should use is a parameter in the GRAM file that is sent from the Portal to the appliance.

There are four types of jobs supported by the UCLA Grid Portal: single, multiple, MPI parallel, and Condor. The Condor job manager initially worked with single and multiple job types with no modifications needed in the Portal. MPI parallel jobs are not supported by the Condor job manager in Globus 4.0. The Condor job type was not previously supported in the Portal, so it was necessary to discover why it was not working.

The Condor job type initially reported two types of errors, a ProcessDied error and an InvalidPathType error, depending on what executable was used while testing the Condor job type. The UCLA Grid Team had used the Condor job type in the past with a SGE job manager, so we initially assumed that we were mis-configuring the Condor job manager in Globus. However, we eventually discovered that the single and multiple job types run in the Condor *Vanilla*

Universe and the Condor job type runs in the Condor *Standard* Universe. The Condor Vanilla Universe does not require any changes to executables in order to run them; the Condor Standard Universe requires executables to be compiled specifically for Condor (`condor_compile` is a command that Condor provides) in order to be run. Our test process was dying because it was not compiled for Condor and after using `condor_compile` on our test process, the Condor job type worked. The InvalidPathType error came from the way that the Condor Standard Universe deals with executable paths, which requires paths to be *absolute* and not *relative*. We documented the requirements for Condor job types in the Portal in the help page of the job submission portlet and modified the job submission portlet to automatically provide an absolute path to the executable for any Condor job types submitted to a Condor job manager and our integration of Condor into the UCLA Grid Portal was complete.

4.2. Overcoming QBETS Challenges

There were two kinds of challenges we faced adding QBETS to the UCLA Grid Portal: challenges with the monitors in the appliance, and challenges adding the standalone portlet into the Portal.

The main challenge in the appliance was modifying the QBETS monitors to get job accounting information and send that information to the QBETS prediction database. Our approach can be broken up into several smaller steps, specifically:

finding out which scheduler is used on the cluster, finding the cluster scheduler log files, efficiently parsing the log files for new jobs, and sending the results to the QBETS database. We created a file called *sitescript* that is used by the QBETS monitors to get job accounting information. The QBETS installation comes with a background sensor that runs the *sitescript* file periodically, at an interval configurable by the user. The background sensor then sends any new job information to the QBETS prediction system. The appliance machine initiates communication with the QBETS prediction system to avoid possible firewall issues setup on the appliance because most firewall settings restrict incoming traffic, not outgoing traffic.

We did not want to require any additional information over what is required in the appliance because QBETS should be fully integrated with the UCLA Grid Portal and should not require any additional work by the cluster administrator to support QBETS. We checked what is required inside Globus to set up a job manager and what Globus does when a new job manager is added in order to see what information we had available to use.

Globus requires the environment variable `GLOBUS_LOCATION` to be set in order to run and Globus creates a job manager folder inside `Globus_Location/etc/gram-service-<job manager>` when installing a new job manager. Inside *sitescript,* we identify the job manager folders that have been created. When Globus adds a new job manager, it requires an environment variable pointing to the home directory of the scheduler to be set. The

LSF scheduler is a special case and Globus expects `LSF_ENVDIR` to point to the LSF configuration file. We use the cluster scheduler environment variable to find the root directory of the scheduler and then search inside the home directory to find the job accounting files. In the case of LSF we use the unix `grep` command to search inside the configuration file for the root directory. The job accounting information that QBETS sends to the backend database is: jobID, start timestamp, wait time, nodes, walltime, and queue. In order to efficiently get job information from the cluster accounting files, we only want to parse jobs that we have not seen before. In order to do this, we modified code that was already in deployment for QBETS and added the modifications to *sitescript*. We save all previously seen jobIDs into a *cache_db* file and all previously parsed files (LSF and PBS) to *logcache_db* file; SGE stores all job accounting information in a single file and a *logcache_db* file is therefore unneeded. We store all relevant accounting information to a *waittime_log_db* file, which gets periodically sent to the QBETS backend database. Each time *sitescript* runs and attempts to parse new jobs, we hash the contents of *cache_db* and *logcache_db* and then hash each new file or jobid to see if it is in our hash of previously seen ids. If the file or jobid has been seen we advance to the next entry, if not, we parse the id or file normally. The monitors allow QBETS to send job accounting information from the appliance to the QBETS backend prediction system, which can then be used to give predictions in the Web Portal interface.

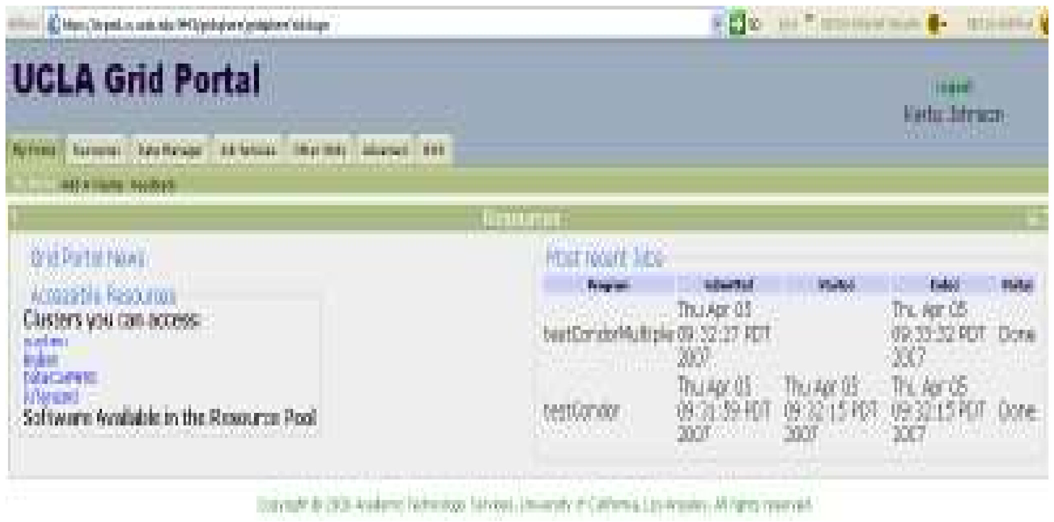
QBETS is implemented as a standalone portlet in the UCLA Grid Portal infrastructure and relies on the UCLA Grid Portal to provide a list of clusters accessible by the logged-in user. Other than needing a list of accessible clusters and their current status, QBETS has little reliance on the UCLA Grid Portal and could be integrated into other Grid Portals with minimal effort. The QBETS portlet was integrated as a standalone portlet in its own “tab” of the Portal because the information provided by QBETS did not fit into the existing UI structure of the Portal. The UI of the separate tab matches the general UI scheme of the Portal and is consistent with the information provided by the Resource Page of the UCLA Grid Portal. We separated the portlet into the frontend UI and the backend information that populates the UI. The frontend UI, or presentation logic, is handled by JavaServer Pages (JSP) and the backend information, or content logic, is handled by Java classes. We used JavaBeans to pass information between the JSP client code and the Java server code.

The goal of the UI is to make QBETS easy to use, comprehensive (a single prediction for all resources), and consistent with the rest of the UCLA Grid Portal. The QBETS portlet is easy to use because only a few pieces of readily available information is needed about each job: the prediction method desired (wait time or deadline), number of nodes, runtime, and either deadline or quantile depending on the prediction method. The user only needs one prediction per job because the portlet gets a prediction for all cluster queue pairs that are accessible to the user for the specified job. After the QBETS portlet gets all predictions, it first sorts the prediction results into clusters that the Portal knows are alive and clusters that the Portal does not know are alive and then sorts each category, alive and not alive, according to the results of the predictions. Thus, the cluster queue pair at the top of the output will have the highest prediction results for clusters that the Portal knows are currently alive. The rest of the cluster queue pairs will be shown in sorted order for each category of clusters alive and not alive. The last prediction category is made of clusters that QBETS can never make a prediction for, specifically clusters running Condor as a scheduler and, as mentioned in Section 2.7, QBETS can not make predictions for Condor Pools as a whole. Finally, all prediction results are color coordinated for easier viewing.

With the QBETS portlet getting a prediction for every cluster queue pair available to a user and checking the aliveness of each cluster, it became necessary to find a way to get all predictions within a reasonable time. We made several optimizations to speed up prediction requests. First, the portlet gets and saves an XML document from QBETS web service that lists all available clusters when the portlet is initialized and the portlet creates a list of the clusters it cannot make a prediction for at that time. When a prediction is requested, the portlet intersects the clusters that a user can access with all clusters known by QBETS and when an intersection is found, the portlet gets predictions for all cluster queue pairs involving the cluster and saves them to a list. After all predictions have been made, the portlet gets the aliveness status of each cluster/queue pair by checking a *Cluster Service* in the Portal. The Cluster Service is periodically updated when the Portal queries the index scripts published by appliances. Therefore, the aliveness reported in the Portal could be delayed by the interval that the Portal waits to check the appliance index scripts, but in return the QBETS portlet is able to immediately discover the aliveness of each cluster. The QBETS portlet gets the predictions from the QBETS database through a web service query and uses a SOAP (Service Oriented Architecture Protocol) client to perform the web service request. As a result of our approach, the user is able to get fast and comprehensive predictions from the QBETS portlet to use when making job scheduling decisions.

5. Results

In Chapter 5 we present our results in the context of the overall UCLA Grid Portal. In Section 5.1 we present the Home Page of the Portal. In Section 5.2 we cover the Resource Page and the index provider scripts that were written for Condor. Section 5.3 demonstrates the Job Services page and how to submit jobs to the Condor scheduler and in 5.4 we look at the NWS Page with the QBETS portlet.



UCLA Grid Portal

Kerby Johnson

Grid Portal News

Accessible Resources

Clusters you can access:

Software Available in the Resource Pool

Most recent Jobs

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
testCondorMultiple	Thu Apr 05 09:32:27 PDT 2007		Thu Apr 05 09:33:32 PDT 2007	Done
testCondor	Thu Apr 05 09:31:59 PDT 2007	Thu Apr 05 09:32:15 PDT 2007	Thu Apr 05 09:32:15 PDT 2007	Done

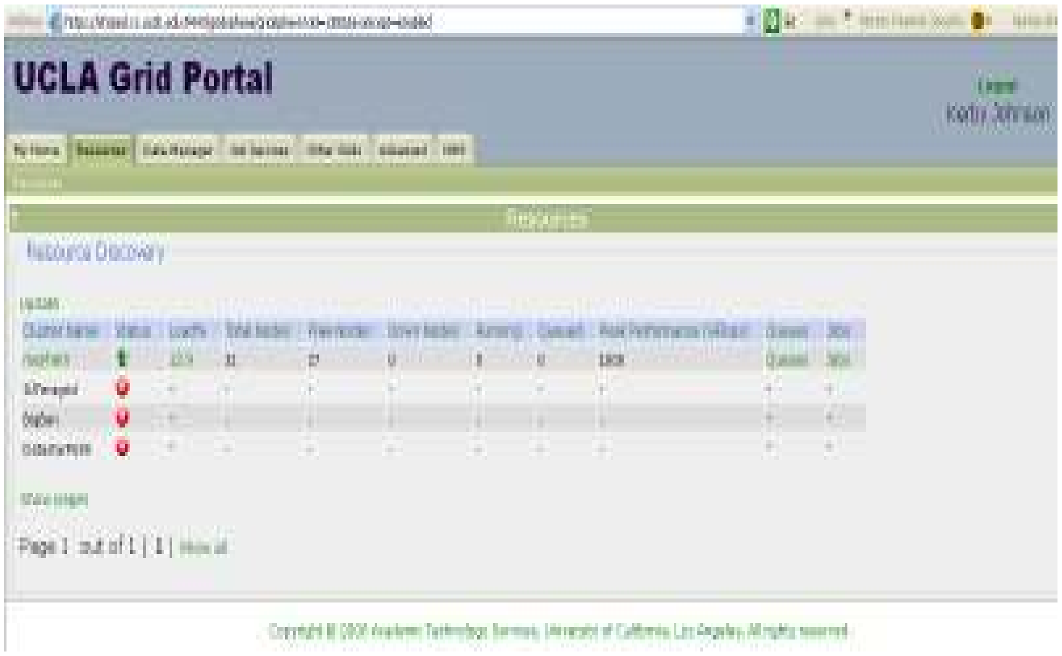
Figure 9: Home Page

5.1. Home Page

Figure 9 presents the home page of the UCLA Grid Portal. This is the first page the user sees when he or she logs into the Portal. The name of the Portal is in the top left corner of the page, the current username and logout button is in the top right corner of the page. Underneath the Portal name are the seven main tabs in the Portal: My Home, Resources, Data Manager, Job Services, Other Grids, Advanced, and NWS. Each tab handles a specific area of the Portal and presents portal pages for those sections. Underneath the main tabs are the subtabs for the currently selected main tab, in this case the subtabs for My Home are My Home, Add A Cluster, and Feedback. The subtabs navigate to specific pages within the maintab area. The subtab currently selected is *My Home*, which informs the current user the status of the clusters they can access on the left side of the page and the status of their most recent jobs on the right. The clusters on the left side are links that can be clicked to see more information about each cluster and the jobs provide current information about recently submitted jobs. The My Home page is provided by the UCLA Grid Portal and our work did not affect this page.

5.2. Resources Page

The Resources Page provides status information about each cluster accessible by the current user. The Resources Page uses the information that the index provider scripts written for cluster schedulers on the appliance publish to a URL. We did not write the code for generating the UI for the Resources Page, but our Condor index scripts provide the information in the Resources Page for clusters running Condor, in this case the *mayhem* cluster.



UCLA Grid Portal

Resources

Resource Discovery

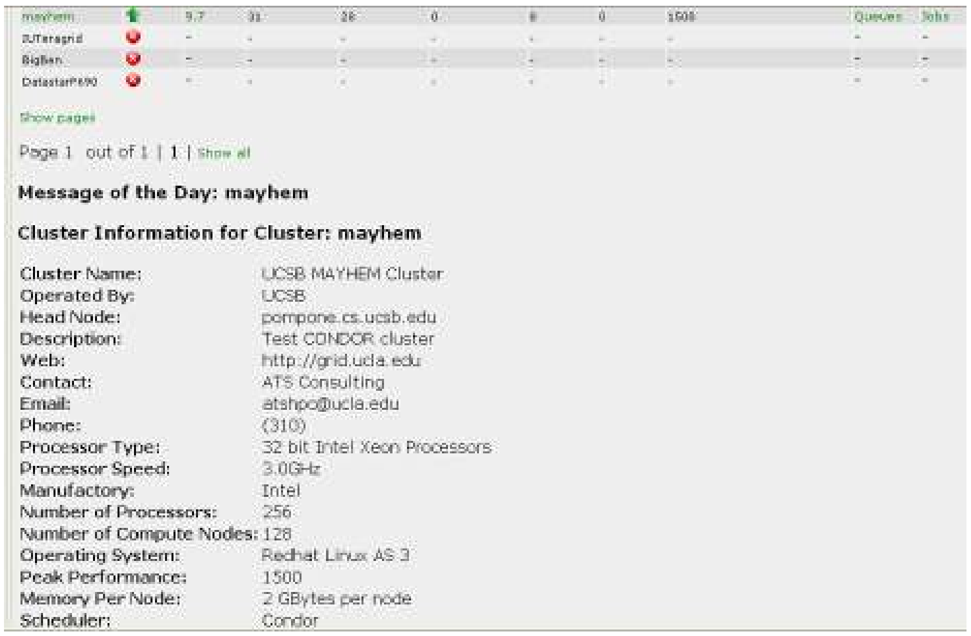
Cluster Name	Status	Load%	Total Nodes	Free Nodes	Down Nodes	Running	Queued	Peak Performance (GFlops)	Queues	Jobs
mayhem	●	12.9	31	27	0	0	0	[illegible]	Queues	Jobs
IUTeragrid	●	-	-	-	-	-	-	-	-	-
BigBen	●	-	-	-	-	-	-	-	-	-
DatastarP690	●	-	-	-	-	-	-	-	-	-

Page 1 out of 1 | 1 | Show all

Figure 10: Portal Resources Page: Main Page

Figure 10 shows the Resources Page of the UCLA Grid Portal. The page presents a table with information about each cluster that the user can access and the information for each row of the table is provided by the `globus-info-provider` script. The first column of the table has the cluster name link, second is the cluster status, third is a link to cluster [CPU] load, then comes total nodes, free nodes, down nodes, running [jobs], queued [jobs], and Peak Performance. Last, are links to queues supported by the cluster and jobs running on the cluster. In this screenshot, the *mayhem* cluster is currently alive with 12.9% load, 31 total nodes, 27 free nodes and 0 running jobs. The clusters *IUTeragrid*, *BigBen* and *DatastarP690* are currently down, which means the appliance running on the cluster is either down or not communicating correctly with the cluster headnode.

The *mayhem* cluster in the table is using a Condor cluster scheduler, and the information provided by *mayhem* is from the index provider scripts that we wrote during this work.



mayhem	⬆	9.7	31	28	0	8	0	1508	Queues	Jobs
IUTeragrid	✖	-	-	-	-	-	-	-	-	-
BigBen	✖	-	-	-	-	-	-	-	-	-
DatastarP690	✖	-	-	-	-	-	-	-	-	-

Show pages

Page 1 out of 1 | 1 | Show all

Message of the Day: mayhem

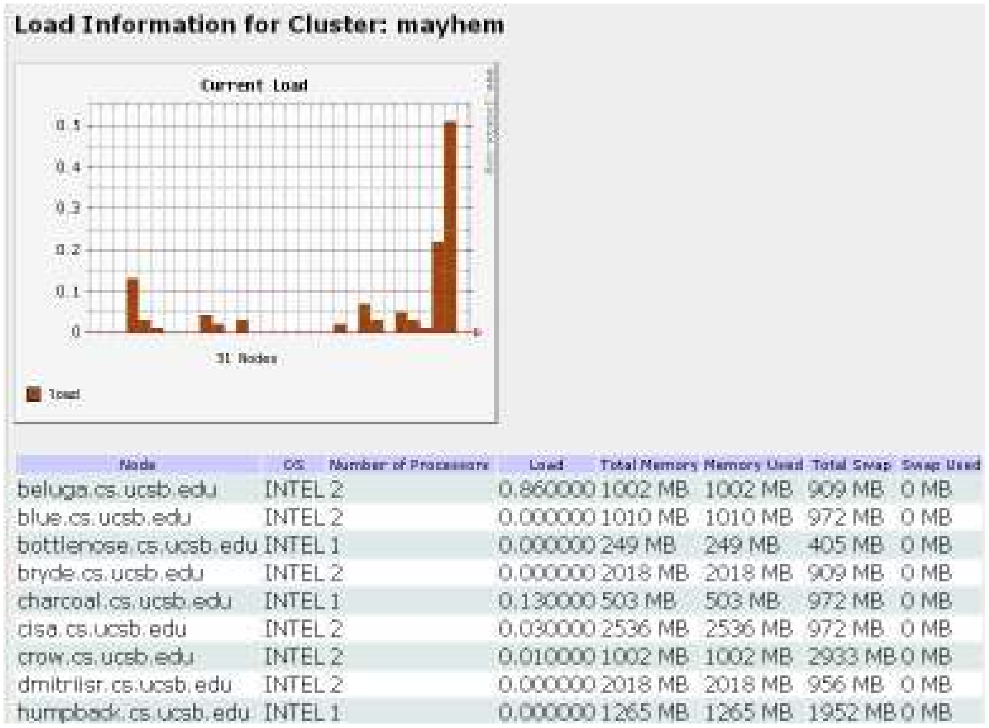
Cluster Information for Cluster: mayhem

Cluster Name:	UCSB MAYHEM Cluster
Operated By:	UCSB
Head Node:	pompone.cs.ucsb.edu
Description:	Test CONDOR cluster
Web:	http://grid.ucla.edu
Contact:	ATS Consulting
Email:	atshpc@ucla.edu
Phone:	(310)
Processor Type:	32 bit Intel Xeon Processors
Processor Speed:	3.0GHz
Manufactory:	Intel
Number of Processors:	256
Number of Compute Nodes:	128
Operating System:	Redhat Linux AS 3
Peak Performance:	1500
Memory Per Node:	2 GBytes per node
Scheduler:	Condor

Figure 11: Resources Page Cluster Information

First we present the cluster information provided by the Resources Page after the *mayhem* cluster link has been clicked. Figure 11 shows that the user is provided with cluster information and specific information about the machines in the cluster. This information is from the index script `globus-clusterinfo-`

`provider`, the only index script we did not need to modify because it only provides statically set text to the user.

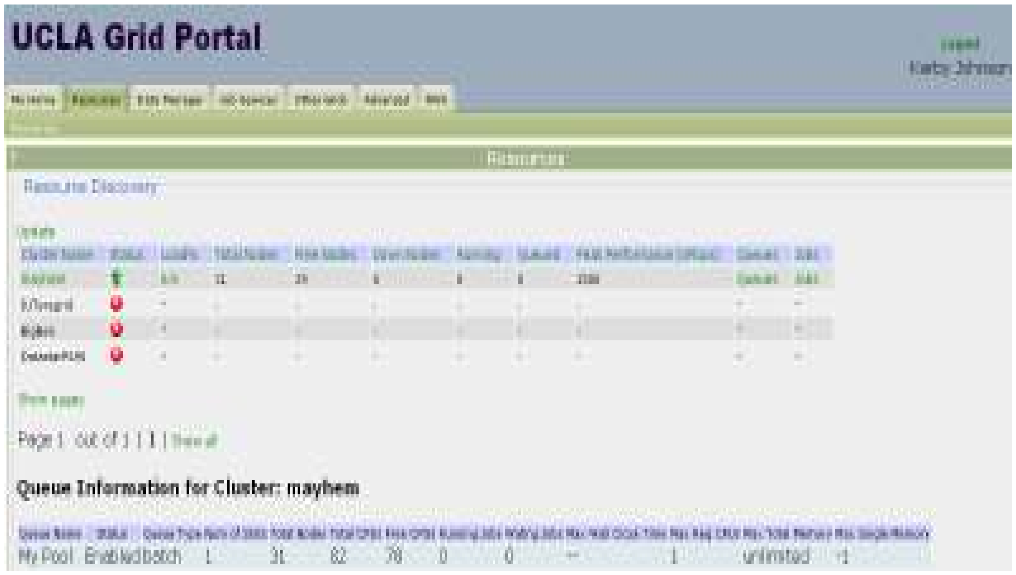


Load Information for Cluster: mayhem

Node	OS	Number of Processors	Load	Total Memory	Memory Used	Total Swap	Swap Used
beluga.cs.ucsb.edu	INTEL	2	0.860000	1002 MB	1002 MB	909 MB	0 MB
blue.cs.ucsb.edu	INTEL	2	0.000000	1010 MB	1010 MB	972 MB	0 MB
bottlenose.cs.ucsb.edu	INTEL	1	0.000000	249 MB	249 MB	405 MB	0 MB
bryde.cs.ucsb.edu	INTEL	2	0.000000	2018 MB	2018 MB	909 MB	0 MB
charcoal.cs.ucsb.edu	INTEL	1	0.130000	503 MB	503 MB	972 MB	0 MB
cisa.cs.ucsb.edu	INTEL	2	0.030000	2536 MB	2536 MB	972 MB	0 MB
crow.cs.ucsb.edu	INTEL	2	0.010000	1002 MB	1002 MB	2933 MB	0 MB
dmitriisr.cs.ucsb.edu	INTEL	2	0.000000	2018 MB	2018 MB	956 MB	0 MB
humpback.cs.ucsb.edu	INTEL	1	0.000000	1265 MB	1265 MB	1952 MB	0 MB

Figure 12: Resources Page Load Information

After viewing cluster information, we look at the information that the Resources Page provides after the “load” link of the *mayhem* cluster has been clicked. At the top of Figure 12 we see a graph showing the load on each machine in the cluster and below the graph is a table with detailed information about each machine in the cluster, in particular the machine name, operating system, number of processors, load on each machine, physical memory, memory used, swap space and swap space used. This information is provided by the `globus-load-provider` index script we wrote for the Condor scheduler.



UCLA Grid Portal

Resource Discovery

[illegible]	Status	[illegible]	Total Nodes	Free Nodes	Down Nodes	Running	Queued	[illegible]	Queues	Jobs
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	Queues	Jobs
[illegible]	[illegible]	-	-	-	-	-	-	-	-	-
[illegible]	[illegible]	-	-	-	-	-	-	-	-	-
[illegible]	[illegible]	-	-	-	-	-	-	-	-	-

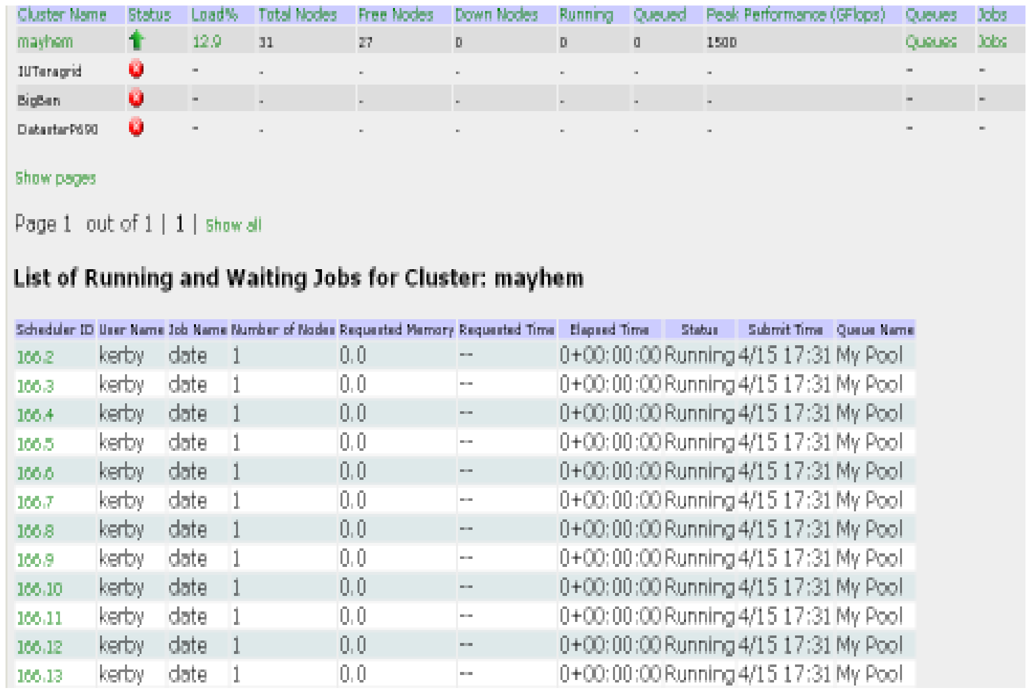
Page 1 out of 1 | 1 |

Queue Information for Cluster: mayhem

[illegible]	Status	Queue Type	[illegible]	Total Nodes	Total CPUs	Free CPUs	Running Jobs	Waiting Jobs	Max Wall Clock Time	Max Req CPUs	Max Total Memory	Max Single Memory
My Pool	Enabled	batch	1	31	82	78	0	0	--	1	unlimited	-1

Figure 13: Resources Page Queue Information

Figure 13 shows the Resources Page after the “Queues” link on the *mayhem* cluster has been clicked and presents the output of the `globus-queue-provider` script we wrote for Condor. In the case of Condor, Figure 13 provides information about a Condor Pool instead of a queue in a cluster. The “Queues” link is less useful for a Condor scheduler because there is only one “queue” in Condor, the Condor Pool. With other batch schedulers the “Queues” link can show the user what resources are available in each queue and the requirements and running jobs of each queue.



Cluster Name	Status	Load%	Total Nodes	Free Nodes	Down Nodes	Running	Queued	Peak Performance (GFlops)	Queues	Jobs
mayhem	⬆	12.9	31	27	0	0	0	1500	Queues	Jobs
IUTeragrid	✖	-	-	-	-	-	-	-	-	-
BigBen	✖	-	-	-	-	-	-	-	-	-
DatastarP690	✖	-	-	-	-	-	-	-	-	-

Show pages

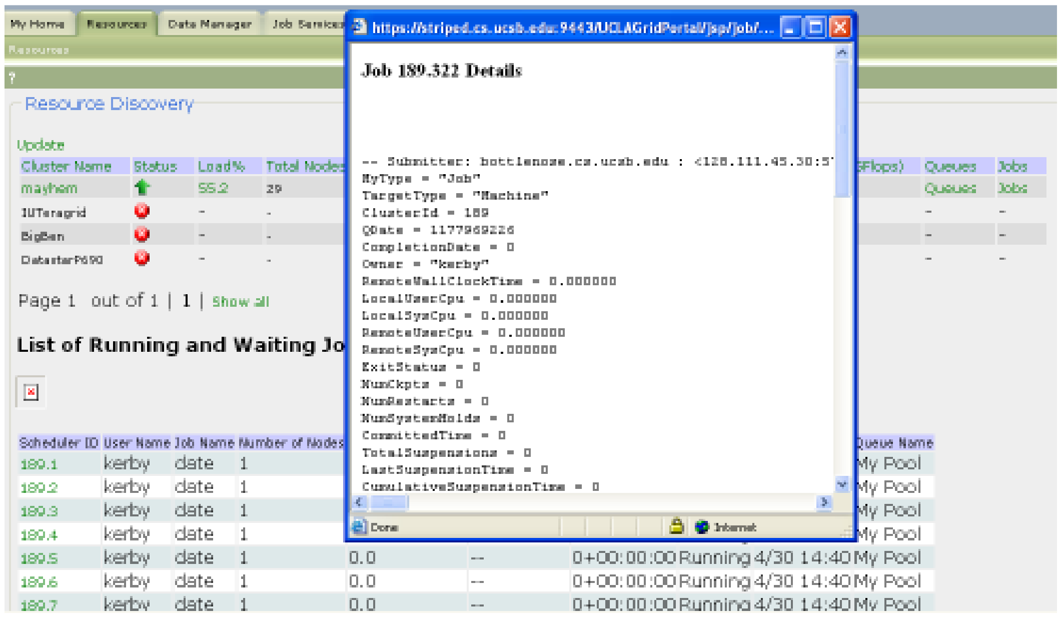
Page 1 out of 1 | 1 | Show all

List of Running and Waiting Jobs for Cluster: mayhem

Scheduler ID	User Name	Job Name	Number of Nodes	Requested Memory	Requested Time	Elapsed Time	Status	Submit Time	Queue Name
166.2	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.3	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.4	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.5	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.6	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.7	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.8	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.9	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.10	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.11	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.12	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool
166.13	kerby	date	1	0.0	--	0+00:00:00	Running	4/15 17:31	My Pool

Figure 14: Resources Page Job Information

In Figure 14, we see the Resources Page after the "Jobs" link of the *mayhem* cluster has been clicked. `Globus-job-provider` provides the information used in the "Jobs" link of the Portal, which ink provides details about all jobs currently running or queued on the cluster and the resources that each job requests, when the job was submitted and how long it has been running. On the left side of Figure 14, we see a "Scheduler ID" link for each job that is running or scheduler in the Portal. The "Scheduler ID" link creates a popup window that provides detailed information about individual jobs, as shown in Figure 15. This information is provided by `globus-jobdetail-provider` and uses the command `condor_q -long <jobid>` to provide detailed information about jobs.



My Home | Resources | Data Manager | Job Services

Resources

Resource Discovery

Update

Cluster Name	Status	Load%	Total Nodes	GFlops)	Queues	Jobs
mayhem	↑	55.2	29		Queues	Jobs
IUTeragrid	●	-	-		-	-
BigBen	●	-	-		-	-
DatastarP690	●	-	-		-	-

Page 1 out of 1 | 1 | Show all

List of Running and Waiting Jo

Scheduler ID	User Name	Job Name	Number of Nodes						Queue Name
189.1	kerby	date	1						My Pool
189.2	kerby	date	1						My Pool
189.3	kerby	date	1						My Pool
189.4	kerby	date	1						My Pool
189.5	kerby	date	1	0.0	--	0+00:00:00	Running	4/30 14:40	My Pool
189.6	kerby	date	1	0.0	--	0+00:00:00	Running	4/30 14:40	My Pool
189.7	kerby	date	1	0.0	--	0+00:00:00	Running	4/30 14:40	My Pool

https://striped.cs.ucsb.edu:9443/UCLAGridPortal/jsp/job/...

Job 189.322 Details

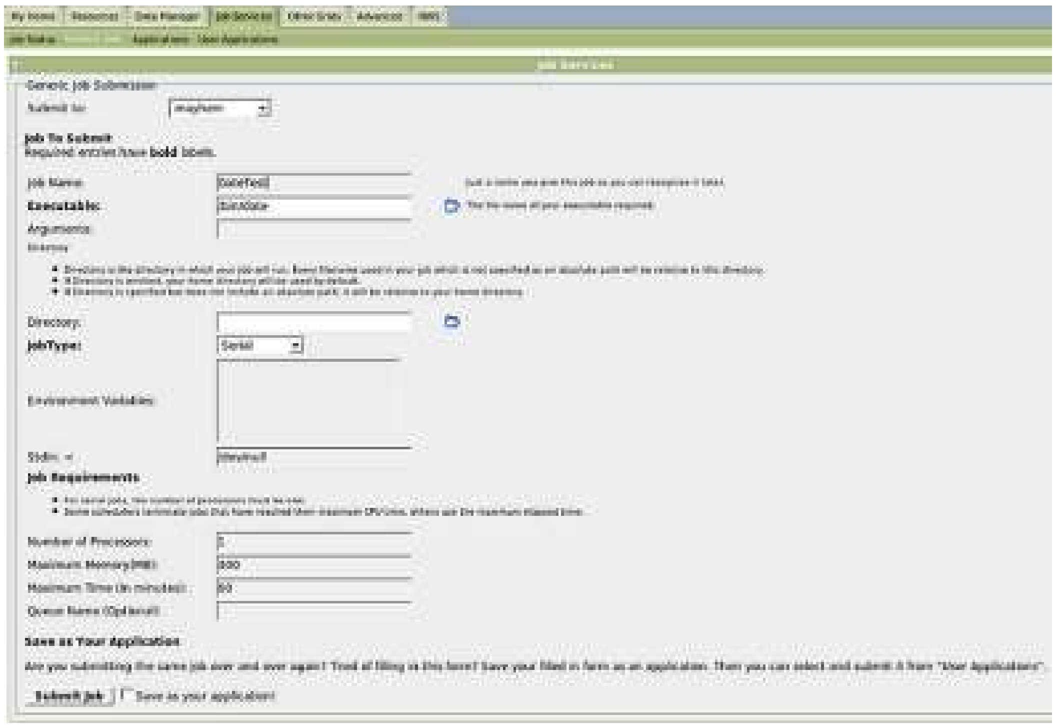
```
-- Submitter: bottlenose.cs.ucsb.edu : <128.111.45.30:5
MyType = "Job"
TargetType = "Machine"
ClusterId = 189
QDate = 1177969226
CompletionDate = 0
Owner = "kerby"
RemoteWallClockTime = 0.000000
LocalUserCpu = 0.000000
LocalSysCpu = 0.000000
RemoteUserCpu = 0.000000
RemoteSysCpu = 0.000000
ExitStatus = 0
NumCkpts = 0
NumRestarts = 0
NumSystemHolds = 0
CommittedTime = 0
TotalSuspensions = 0
LastSuspensionTime = 0
CumulativeSuspensionTime = 0
```

Done | Internet

Figure 15: Resources Page Job Details

5.3. Job Services Page

The Job Services page is where the user submits jobs and checks the status of jobs. The user submits jobs through the Generic Jobs subtab, views jobs through the Job Status subtab, and can use the Applications and User Applications subtabs to quickly submit standardized job formats.



Generic Job Submission

Submit to: mayhem

Job To Submit
Required entries have **bold** labels.

Job Name: DateTest [illegible]
Executable: /bin/date [illegible]
Arguments:
Directory
- [illegible]
- [illegible]
- [illegible]

Directory:
JobType: Serial
Environment Variables:
Stdin: < /dev/null
Job Requirements
- [illegible]
- [illegible]

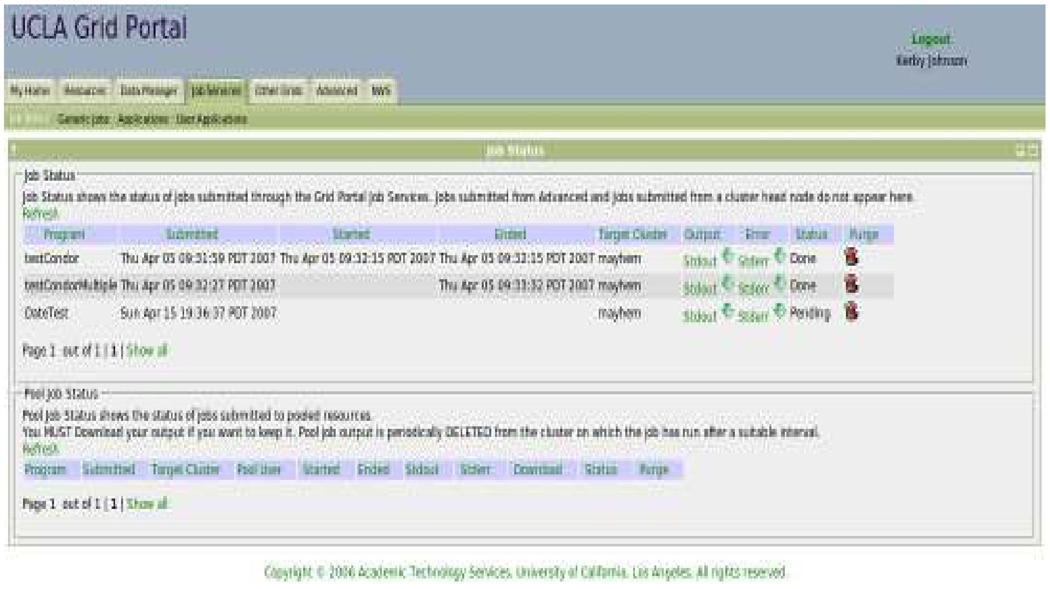
Number of Processors: 1
Maximum Memory (MB): 400
Maximum Time (in minutes): 60
Queue Name (Optional):

Save as Your Application
Are you submitting the same job over and over again? Tired of filling in this form? Save your filled in form as an application. Then you can select and submit it from "User Applications".

Submit Job Save as your application!

Figure 16: Job Submission Page

Figure 16 shows the Generic Jobs subtab of the Job Services maintab, which is the page used for job submission in the UCLA Grid Portal. The interface in Figure 16 is used for submitting jobs to any cluster scheduler supported by the UCLA Grid Portal, including Condor and the only interface change we made was to add the job types "Condor" and "multiple" to the Job Type drop down box. In Figure 16, the user is submitting a job called "DateTest" to the *mayhem* cluster. The job is the executable `/bin/date` and is a "Serial" job with no arguments or environment variables. The job has the default requirements of 1 processor, a maximum memory requirement of 400Mb and a maximum time of 60 minutes. We submit the job in the Portal and go to the "Job Status" subtab to see the status of the job, as shown in Figure 17. The job status page presents a table with the program name, time the job was submitted, time the job started, time the job ended, target cluster, `stdout`, `stderr`, and job status. The “testCondorMultiple” job does not have a start time because it was a “multiple” job type and started several jobs at once.



Job Status

Job Status shows the status of jobs submitted through the Grid Portal Job Services. Jobs submitted from Advanced and jobs submitted from a cluster head node do not appear here.

Refresh

Program	Submitted	Started	Ended	Target Cluster	Output	Error	Status	Purge
testCondor	Thu Apr 05 09:31:59 PDT 2007	Thu Apr 05 09:32:15 PDT 2007	Thu Apr 05 09:32:15 PDT 2007	mayhem	Stdout	Stderr	Done	
testCondorMultiple	Thu Apr 05 09:32:27 PDT 2007		Thu Apr 05 09:33:32 PDT 2007	mayhem	Stdout	Stderr	Done	
DateTest	Sun Apr 15 19:36:37 PDT 2007			mayhem	Stdout	Stderr	Pending	

Page 1 out of 1 | 1 | Show all

Pool Job Status

Pool Job Status shows the status of jobs submitted to pooled resources.
You MUST Download your output if you want to keep it. Pool job output is periodically DELETED from the cluster on which the job has run after a suitable interval.

Refresh

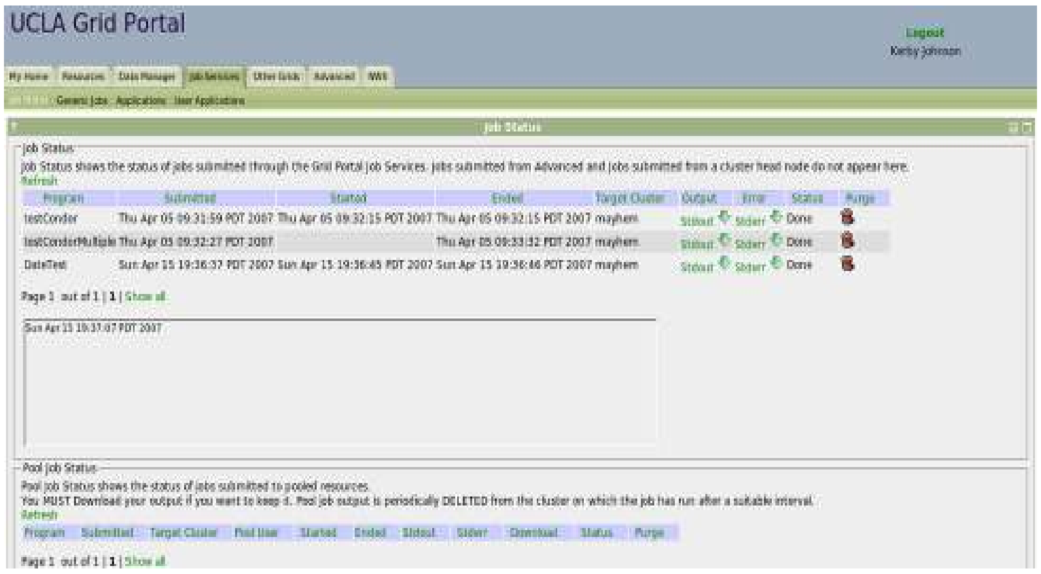
Program	Submitted	Target Cluster	Pool User	Started	Ended	Stdout	Stderr	Download	Status	Purge

Page 1 out of 1 | 1 | Show all

Copyright © 2006 Academic Technology Services, University of California, Los Angeles. All rights reserved.

Figure 17: Job Status Page

The DateTest job we submitted is currently pending, so we press the “Refresh” button and the Portal updates the Job Status page. Figure 18 shows the Job Status page after the “DateTest” job has finished on *mayhem* and the `Stdout` link has been clicked. The Portal displays the output, in this case the current date, in the rectangular window in the middle of the page. This interface is convenient because it is easy to view the output of all finished jobs across all clusters from the Web Portal interface. In the case of a "multiple" job type submission, it shows the standard output of all job runs in the `Stdout` link. We did not make any changes to this interface in the course of our work.



Job Status

Job Status shows the status of jobs submitted through the Grid Portal Job Services. Jobs submitted from Advanced and jobs submitted from a cluster head node do not appear here.

Refresh

Program	Submitted	Started	Ended	Target Cluster	Output	Error	Status	Purge
testCondor	Thu Apr 05 09:31:59 PDT 2007	Thu Apr 05 09:32:15 PDT 2007	Thu Apr 05 09:32:15 PDT 2007	mayhem	Stdout	Stderr	Done	
testCondorMultiple	Thu Apr 05 09:32:27 PDT 2007		Thu Apr 05 09:33:32 PDT 2007	mayhem	Stdout	Stderr	Done	
DateTest	Sun Apr 15 19:36:37 PDT 2007	Sun Apr 15 19:36:45 PDT 2007	Sun Apr 15 19:36:46 PDT 2007	mayhem	Stdout	Stderr	Done	

Page 1 out of 1 | 1 | Show all

```
Sun Apr 15 19:37:07 PDT 2007
```

Pool Job Status

Pool Job Status shows the status of jobs submitted to pooled resources.
You MUST Download your output if you want to keep it. Pool job output is periodically DELETED from the cluster on which the job has run after a suitable interval.

Refresh

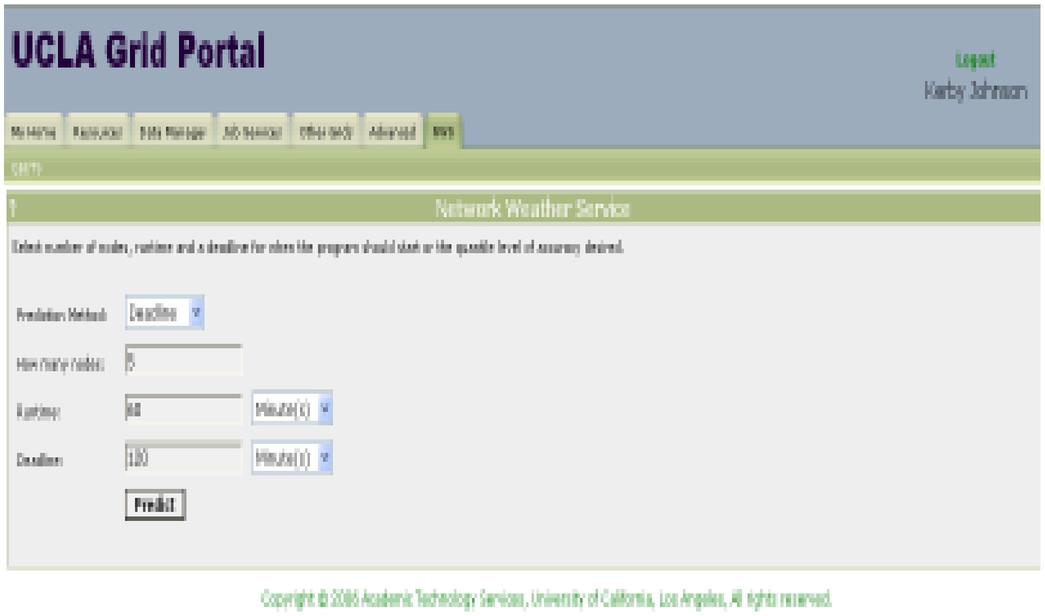
Program	Submitted	Target Cluster	Pool User	Started	Ended	Stdout	Stderr	Download	Status	Purge

Page 1 out of 1 | 1 | Show all

Figure 18: Job Status Page with Job Info

5.4. NWS Page

The NWS Page is a new top-level tab we created in order to integrate QBETS into the UCLA Grid Portal; Figure 19 shows the user interface for the QBETS portlet. The user selects the prediction method desired, deadline or waittime, the number of nodes, the runtime of the job and either the deadline for the job to start by (for deadline prediction), or the quantile level of accuracy desired (for waittime prediction) and then requests a prediction. In the following examples, we had the QBETS portlet think that *IUTeragrid* and *DataStarP690* clusters were alive and the *BigBen* cluster was not alive. We did this to show how the QBETS UI would work with some alive and not alive clusters



Select number of nodes, runtime and a deadline for when the program should start or the quantile level of accuracy desired.

Prediction Method: Deadline

How many nodes: 5

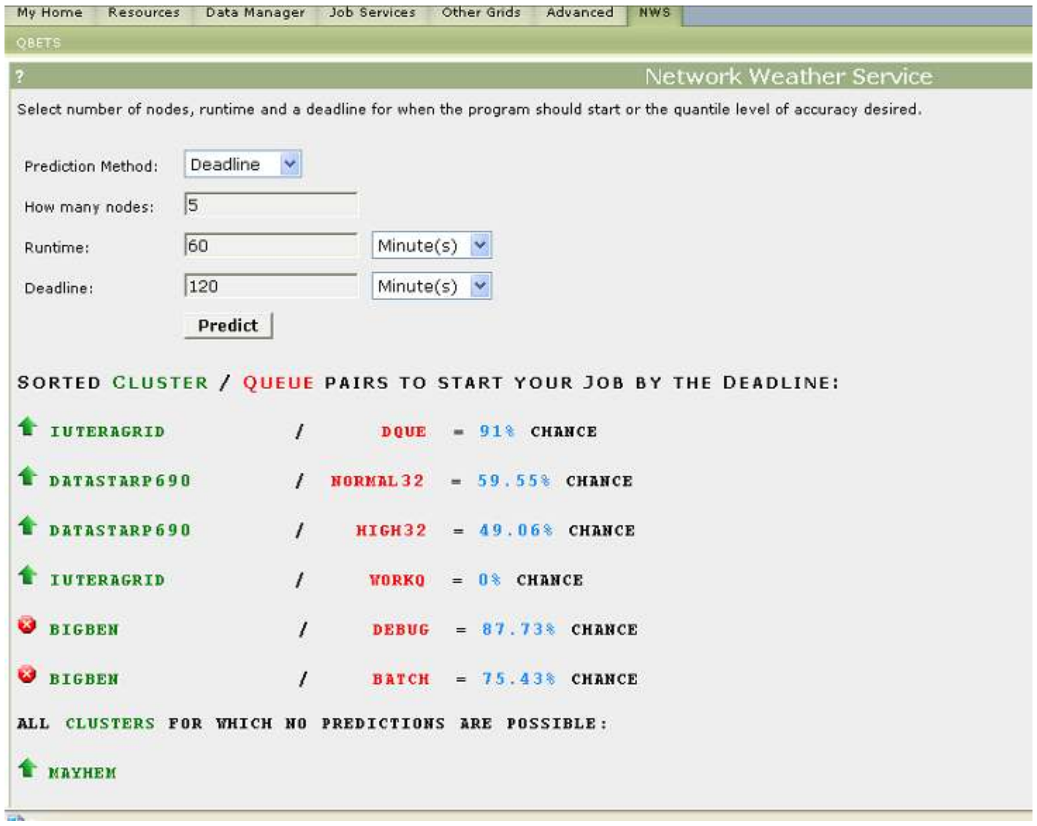
Runtime: 60 Minute(s)

Deadline: 120 Minute(s)

Copyright © 2006 Academic Technology Services, University of California, Los Angeles, All rights reserved.

Figure 19: QBETS Deadline Prediction

. In Figure 19, the Deadline prediction method is chosen for a job that requires 5 nodes, has a runtime of 60 minutes, and a deadline of 120 minutes. We then request the prediction, with the results shown in Figure 20.



Select number of nodes, runtime and a deadline for when the program should start or the quantile level of accuracy desired.

Prediction Method: Deadline
How many nodes: 5
Runtime: 60 Minute(s)
Deadline: 120 Minute(s)

SORTED CLUSTER / QUEUE PAIRS TO START YOUR JOB BY THE DEADLINE:

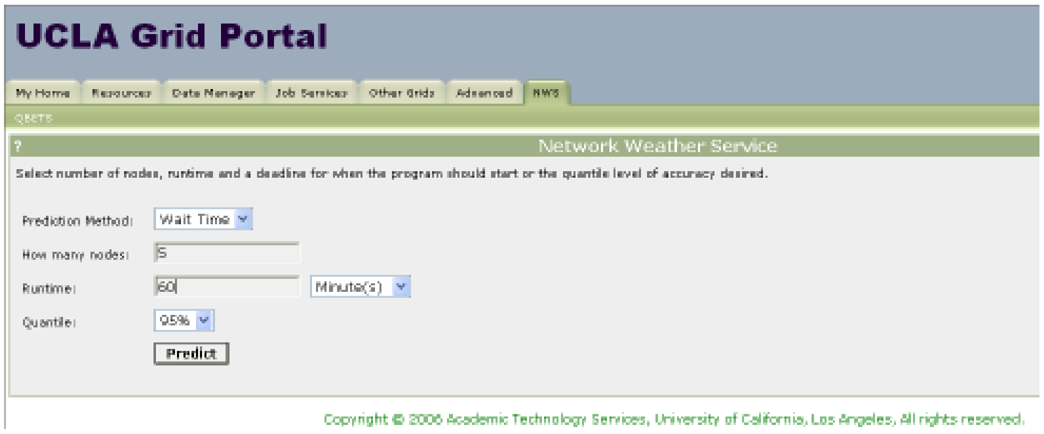
IUTERAGRID	/	DQUE	=	91% CHANCE
DATASTARP690	/	NORMAL32	=	59.55% CHANCE
DATASTARP690	/	HIGH32	=	49.06% CHANCE
IUTERAGRID	/	WORKQ	=	0% CHANCE
BIGBEN	/	DEBUG	=	87.73% CHANCE
BIGBEN	/	BATCH	=	75.43% CHANCE

ALL CLUSTERS FOR WHICH NO PREDICTIONS ARE POSSIBLE:

MAYHEM

Figure 20: QBETS Deadline Prediction Results

The *IUTeragrid Dque* queue has the highest chance to start the job by the 120-minute deadline, followed by the *DataStarP690 Normal32* queue and *DatastarP690 High32* queue. Finally the *IUTeragrid Workq* has a 0% chance of starting the job by the deadline. The two queues on the *BigBen* cluster have a higher probability than the *DataStarP690* queues, but because *BigBen* is not currently alive in the Portal it is reported after all alive queues. The green up arrow or red X next to clusters clarify whether the cluster is alive or not and all clusters in the Resources Page will show up in QBETS results. The last section reported by QBETS lists all clusters for which no predictions are possible, currently the *mayhem* cluster is alive and cannot be predicted for because it is using a Condor scheduler, and QBETS does not currently provide predictions for Condor Pools as a whole as discussed in section 2.7.



Select number of nodes, runtime and a deadline for when the program should start or the quantile level of accuracy desired.

Prediction Method: Wait Time

How many nodes: 5

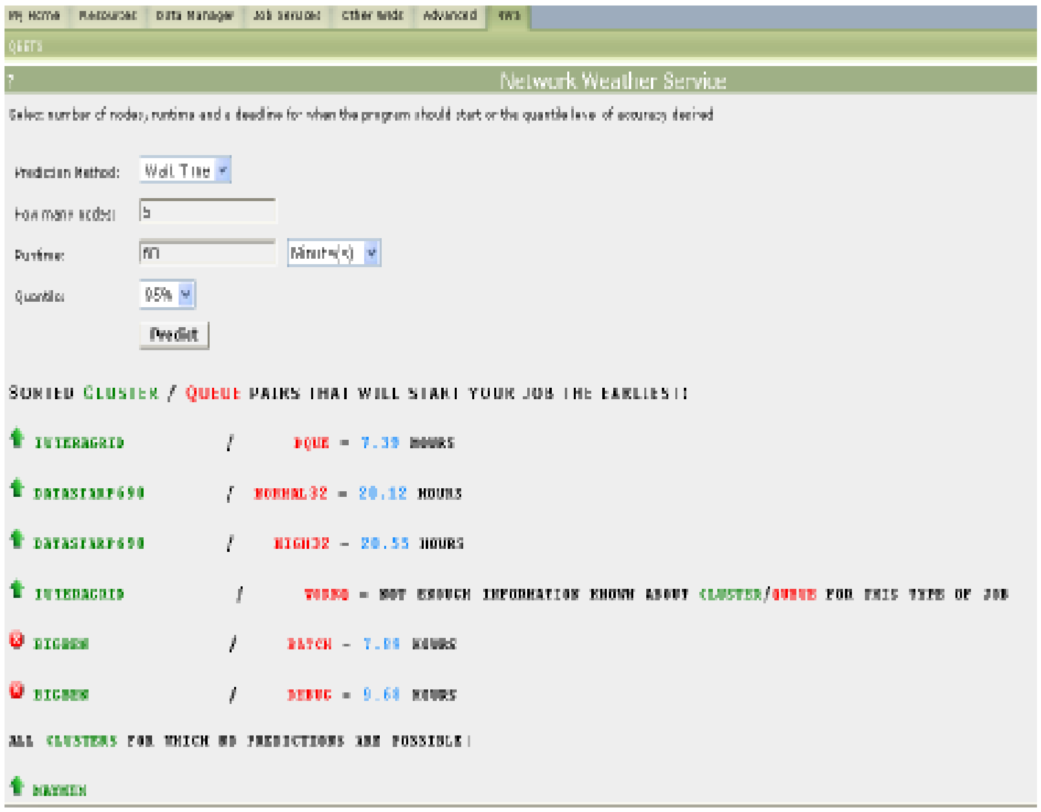
Runtime: 60 Minute(s)

Quantile: 95%

Copyright © 2006 Academic Technology Services, University of California, Los Angeles, All rights reserved.

Figure 21: BQP Wait Time Prediction

Next, we look at the QBETS interface when the Wait Time prediction method is chosen, as shown in Figure 21. The job still needs 5 nodes and runs for 60 minutes, but instead of a deadline there is a quantile field, which is specifies the level of accuracy desired from QBETS. QBETS supports 95%, 75% and 50% as options for quantile.



Select number of nodes, runtime and a deadline for when the program should start or the quantile level of accuracy desired

Prediction Method: Wait Time
How many nodes: 5
Runtime: 60 Minute(s)
Quantile: 95%

SORTED CLUSTER / QUEUE PAIRS THAT WILL START YOUR JOB THE EARLIEST!

```
IUTERAGRID      /      DQUE = 7.39 HOURS
DATASTARP690    /  NORMAL32 = 20.12 HOURS
DATASTARP690    /    HIGH32 = 20.55 HOURS
IUTERAGRID      /     WORKQ = NOT ENOUGH INFORMATION KNOWN ABOUT CLUSTER/QUEUE FOR THIS TYPE OF JOB
BIGBEN          /     BATCH = 7.89 HOURS
BIGBEN          /     DEBUG = 9.68 HOURS
```

ALL CLUSTERS FOR WHICH NO PREDICTIONS ARE POSSIBLE!

```
MAYHEM
```

Figure 22: BQP Wait Time Prediction Results

We request a prediction from QBETS and see the results in Figure 22. The *IUTeragrid Dque* has a 95% percent chance to start before 7.39 hours, then the *DataStarP690 Normal32* and *High32* queues have a 95% chance to start in 20.12 and 20.55 hours, respectively, and QBETS does not have enough information about *IUTeragrid WorkQ* to make a prediction for when it would start with 95% probability. The *BigBen* queues *Batch* and *Debug* would start within 7.89 and 9.68 hours respectively, but the *BigBen* cluster is currently not alive in the Portal. As before, the *mayhem* cluster is shown in the no predictions possible section.

Using both the Deadline prediction results and the Wait Time prediction results, we can see that our job has a 91% chance to start within 2 hours (120 minutes) and there is a 95% chance to start within 7.39 hours if we submit it to the *IUTeragrid Dque* queue. If the user prefers not to submit their job to the *IUTeragrid dque*, they can also submit to other available clusters and queues and know the probability that the job will start by a deadline or the upper bound on when the job will start.

6. Lessons Learned and Future Work

In Chapter 6 we present the lessons we have learned, unexpected difficulties encountered and future work desired. In section 6.1 we discuss the lessons learned over the course of this work. In section 6.2 we mention some of the unexpected difficulties that came up and in 6.3 we discuss adding Condor glide-in as future work.

6.1. Lessons Learned

We have learned many lessons over the course of this work. First and foremost, we learned that doing Web Portal development is hard primarily because none of the code in Java Server Pages (JSP) is checked at compile time and there is a conflict between what the client-side code knows and what the server-side code knows. A minor typo in a JSP causes a stack trace exception in the Portal, which can be difficult to track down when the information from stack traces is sometimes cumbersome and unhelpful.

Debugging of JSP itself is cumbersome because each minor change requires the Portal to be redeployed, which takes 10-30 seconds. Furthermore, when the Portal is redeployed and Apache is restarted, we sometimes needed to log out and log back in again instead of just refreshing the web browser because the changed code would use data that gets set when the user initially logs in. If we tried to just refresh the Portal, then we would be trying to use uninitialized data and cause an exception and throw a stack trace. This caused difficulties because when we would forget to perform a full restart, code that was well tested would mysteriously stop working and need to be debugged.

A final development challenge with Web Portals was discovering that things we thought were easy could turn out to be really hard, and some things that we thought would be hard turned out to be easy. An example of something that we thought was easy turning out to be hard is dynamic modification of the user interface presented from JSP pages, particularly mutually dependent dynamic dropdown boxes. It seemed like a simple if-statement would provide a solution to this problem where if a specific value is in the first drop box, the JSP looks up what should be in the second dropdown box and displays it. The difficulty in this "easy" fix is that the overlap in state shared between the client and server is problematic in development and the server cannot look up the values in the first dropdown box because that is known by the client code and not the server code. Thus, in order to accomplish dynamic modification of the user interface, the developer typically either needs submit the page to the backend logic and refresh the page, or for some limited problems, the use of Javascript in the JSP is enough to solve the problem.

Throughout this thesis, we have learned how to do Web programming, and programming in JSP and Java classes using Java Beans. We have alos learned about Web and Grid Portals, Web Services, batch schedulers, and the details of how

Condor works. We feel that this thesis has improved our general programming and problem solving skills and taught us specific techniques about programming in a Web environment. We also feel that writing this thesis has greatly improved our ability to communicate the details of complex technical projects in an understandable manner.   Writing this thesis has specifically taught us how to organize and present information in a written medium, and the importance of precision and clarity in writing. We feel that the skills we have learned while writing this thesis will prove to be very valuable in the future.

6.2. Unexpected Difficulties

In this section we discuss some of the unexpected difficulties that we encountered in our work. The first and foremost difficulty we encountered was that setting up the ULCA Grid Portal was very time consuming. It required a fresh install of two separate machines with Fedora Core 4, and getting the communication set up correctly between the portal machine and appliance machine. The installation instructions were eight pages long and some installation issues we encountered were hardware dependant. The UCLA Grid Portal is still under development, and as a result, installing the UCLA Grid Portal was error-prone and required reinstalling the portal and the appliance several times. An additional delay arose when we made our initial installation unusable during development and needed to reinstall. After reinstalling, we spent a lot of time figuring out why the appliance was not visible in the Portal, like it had been before. It turned out that when you add a cluster to the Portal it asks for ClusterName, True Name, Head Node and scheduler and we entered the ClusterName (mayhem) instead of what the Portal actually wanted, the ClusterNode or ApplianceNode (bottlenose.cs.ucsb.edu). The UCLA Grid Portal has since developed an installation package that will help with setting up the UCLA Grid Portal.

Another difficulty came up while using Web Services to communicate between the QBETS portlet and the QBETS prediction system in order to get predictions. There are two primary methods of web service communication, using a WSDL (Web Service Definition Language) or using SOAP (Service Oriented Architecture Protocol) for communication. The QBETS prediction system provides a WSDL schema that can be used to automatically generate the necessary WSDL files which will provide Web Service communication between the Portal and the QBETS database. The generated WSDL files worked outside of the Portal in testing, but when we tried to use them inside the Portal we got a "MalformedURIException: Cannot call function with empty parameters" error. According to the WSDL standards, the function calls in question were allowed to be called with empty parameters, but, it turned out that our implementation disallowed that functionality. The QBETS Web Service is also used in deployment at the University of Texas User Portal. After consulting with the University of Texas, we learned that they did not use the WSDL schema because they found the generated files to be cumbersome and hard to use. Just as they did, we built a standalone SOAP client of about 70 lines of code to solve this problem and the SOAP client successfully communicated with the QBETS Web Service.

6.3. Future Work

Our work adds functionality and enhances the user's experience with the UCLA Grid Portal but there is still improvements that could be made to our work, which we discuss in this section. One of the most helpful additions to the UCLA Grid Portal would be the addition of Condor "glide-in", which is a method to temporarily install Condor onto machines in a cluster this running another batch scheduler. Figure 23 demonstrates how Condor glide-in would work. The user submits a Condor startup job in the Portal that is just like any other job and the job gets executed on the cluster and installs Condor on the number of nodes requested by the batch job, creating a temporary Condor Pool. Now that Condor is installed on the cluster, Condor jobs can be submitted to the Condor Pool or the Pool can be added to a pre-existing Condor Pool, adding more resources to the Pool. When the Condor startup job is terminated, Condor "glides" out of the cluster and is no longer running on any machines.

Condor glide-in provides a way to create a Condor Pool and execute Condor jobs on a cluster that does not have Condor as a cluster scheduler. Condor glide-in is used to add more resources to a Condor Pool or to provide a way to submit executables that are optimized for Condor in a cluster that is not running Condor. The main technical challenges with implementing Condor glide-in are in submitting jobs to the created Condor Pool from the Web Portal interface, creating a Condor glide-in job type that will also check whether a cluster has the required inter-cluster communication necessary for Condor glide-in to function. The current UCLA Grid Portal interface only provides a mechanism to submit jobs to a cluster and does not support a way to specify what job manager on the cluster a job should use. In order to add Condor glide-in to the UCLA Grid Portal, it would be necessary to both add the Condor glide-in job type to the Portal as a predefined application, setup Condor on the machines in the cluster, and provide ways to submit to a newly created Pool in the Portal and merge the newly created Pool with other Condor Pools.

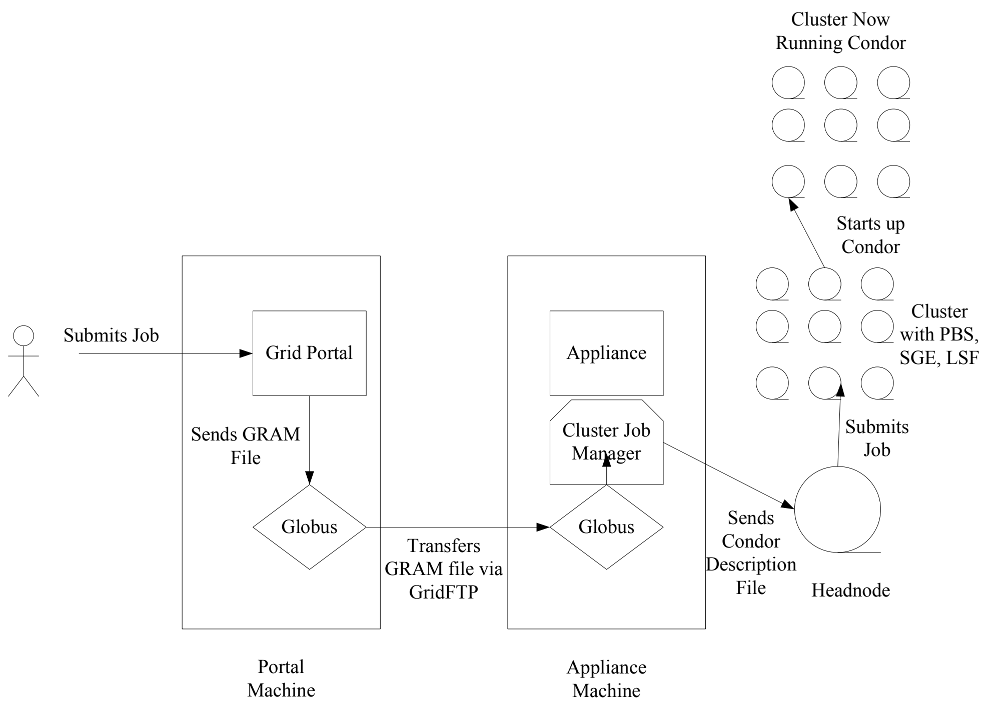


Figure 23: Condor Glide-In

7. Conclusions

In this work we have enriched the functionality of the UCLA Grid Portal by increasing the resources available in the UCLA Grid Portal and providing better support for computationally intensive jobs and by providing additional feedback concerning queue delay to the user. We have added this functionality by supporting Condor as a cluster scheduler in the UCLA Grid Portal and integrating QBETS, a prediction system that provides statistical bounds on queue delay based on *current* cluster conditions, into the Portal.

Furthermore, we believe that the UCLA Grid Portal will be enhance the research efforts of students and faculty at the UC Campuses by providing access to resources and convenient resource and job management through a Web Portal interface and the QBETS prediction system. We look forward to seeing the effect the UCLA Grid Portal will have on research at the various UC Campuses.

Works Cited

[1] I. Foster and C. Kesselman. Computational Grids. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1998, pp. 3-37.

[2] G. von. Laszewski. The Grid-Idea and Its Evolution. *Journal of Information Technology*, Volume 47, Issue 6, pp. 319-329, 2005. doi: 10.1524/itit.2005.47.6.319.

[3] F. Berman, G. C. Fox, and A. Hey. The Grid: past, present, future. Grid Computing: Making the Global Infrastructure a Reality. Chichester : John Wiley & Sons Ltd, 2003, pp. 9-50.

[4] I. Foster and C. Kesselman (eds). The Grid: Blueprint for a New Computing Infrastructure. San Francisco, CA : Morgan Kaufmann, 1998.

[5] R. Allan, C. Awre, M. Baker, A. Fish. Portals and Portlet*s* 2003. *In Proceedings of the NeSC Workshop*. Edinburgh, 2004.

[6] C. Combar and C. Doremus. Portals 101. *Mainejug.org.* 2003. http://www.mainejug.org/jug/meetings/2004/may/Portals101_FINAL.pdf.

[7] D. Gannon, et al. Grid Portals: A Scientist's Access Point for Grid Services. *Draft from: GCE-WG*. 2003. Global Grid Forum.

[8] University of California Grid. UC Grid Portals. http://www.ucgrid.org.

[9] Portable Batch System. http://www.openpbs.org/.

[10] Platform LSF Family of Products. http:/www.platform.com/Products/Platform.LSF.Family.

[11] Sun Grid Engine. http://www.sun.com/software/gridware/.

[12] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. *In Proceedings of the 8th International Conference of Distributed Computing Systems*, San Jose, CA, 1988.

[13] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *In Proceedings of the Workshop on Environments and Tools for Parallel Scientific Computing*, Lyon, France, 1996.

[14] The Globus Project. http://www.globus.org.

[15] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *In IFIP: International Conference on Network and Parallel Computing*, 2006.

[16] GT 4.0 Security: Key Concepts. Globus. http://www.unix.globus.org/toolkit/docs/4.0/security/key-index.html.

[17] The TeraGrid Project. Teragrid project. http://www.teragrid.org.

[18] P. H. Beckman. Building the TeraGrid. *In Philosophical Transactions of Royal Society*, 2005.

[19] About Page. Teragrid. http://www.teragrid.org/about.

[20] G. von Laszewski, et al. CoG kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. *In ACM Java Grande 2000 Conference*, San Francisco, CA, 2000.

[21] Legion. http://legion.virginia.edu.

[22] A. Grimshaw, W. Wulf, et al. The Legion Vision of a Worldwide Virtual Computer. *In Communications of the ACM*, January 1997, Vol. 40(1) pp. 39-45.

[23] A. Natrajan, et al. The Legion Grid Portal. *In Grid Computing Environments 2001, Concurrency and Computation: Practice and Experience*, 2001.

[24] R. Ciapala. Develop Turbocharged Apps for Windows Compute Cluster Server. MSDN Magazine, April 2006.

[25] M. Corbatto. An Introduction to PORTABLE BATCH SYSTEM (PBS). 2000. http://hpc.sissa.it/pbs/pbs.html.

[26] What is Condor? Condor: High Throughput Computing. http://www.cs.wisc.edu/condor/description.html.

[27] Classified Advertisements. Condor: High Throughput Computing. http://www.cs.wisc.edu/condor/classad.

[28] J. Brevik, D. Nurmi, and R. Wolski. Predicting Bounds on Queuing Delay for Batch-scheduled Parallel Machines. *In Proceedings of ACM Principles and Practices of Parallel Programming (PPoPP)*, New York, NY, 2006.

[29] J. Novotny, M. Russel, and O. Wehrens. Gridsphere: A Portal Framework for Building Collaborations. *In Proceedings of the 1st International Workshop on Middleware in Grid Computing*, Rio de Janeiro, Brazil, 2003.