ABSTRACT

Title of Document: THE LATTICE PROJECT: A MULTI-MODEL GRID COMPUTING SYSTEM

Adam Bazinet, Master of Science, 2009

Directed By: Professor Michael Cummings
Center for Bioinformatics and Computational Biology
Affiliate Professor, Department of Computer Science

This thesis presents The Lattice Project, a system that combines multiple models of Grid computing. Grid computing is a paradigm for leveraging multiple distributed computational resources to solve fundamental scientific problems that require large amounts of computation. The system combines the traditional Service model of Grid computing with the Desktop model of Grid computing, and is thus capable of utilizing diverse resources such as institutional desktop computers, dedicated computing clusters, and machines volunteered by the general public to advance science. The production Grid system includes a fully-featured user interface, support for a large number of popular scientific applications, a robust Grid-level scheduler, and novel enhancements such as a Grid-wide file caching scheme. A substantial amount of scientific research has already been completed using The Lattice Project.

THE LATTICE PROJECT: A MULTI-MODEL GRID COMPUTING SYSTEM

By

Adam Bazinet

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2009

Advisory Committee:
Prof. Michael Cummings, Chair
Prof. Alan Sussman
Prof. Chau-Wen Tseng

Foreword

Some of the material in this thesis has been previously published, and appropriate citations of that work have been made. Initial development of GSBL (section 5.1.3) and the Globus-BOINC adapter (section 6.2) was done jointly by Daniel Myers and me, and the original version of the GSG (section 5.1.8) was developed by John Fuetsch. I have been the primary developer of The Lattice Project since late 2004, and during this time I have been responsible for the continued development of GSBL, the GSG, and the Globus-BOINC adapter. In addition, I have led the development of the command line user interface and web monitoring tools (sections 4.1.1–4.1.3), have been responsible for the hardware and software configuration of the core Grid infrastructure (sections 4.2.1 and 4.2.2), have led the technical effort to integrate all the Grid resources, including setup and administration of the Lattice BOINC Project (section 4.2.3), have developed many Grid services (section 5.1.1), have designed and implemented a data management scheme (section 5.2) and Grid meta-scheduler (section 5.3), have developed new methods of porting BOINC applications (section 6.5.1), have worked on novel user interface prototypes (section 7.2.1), and have been solely responsible for supporting all the research projects that have used the production Grid system (Appendix B).

Acknowledgements

First and foremost, I would like to thank my advisor, Michael Cummings, for all of his support during my time here at the University of Maryland, and for his steadfast commitment to the development of The Lattice Project, without which none of this would have been possible.

Second, I would like to thank Daniel Myers and John Fuetsch for their contributions to the design and development of the project when it was still in its infancy. I would also like to thank Stephen McLellan and Christopher Milliron, who assisted with project development at an early and critical stage; Andrew Younge, who recently helped upgrade the Globus-BOINC adapter; and Jonathan Howard and Deji Akinyemi, who authored a few Grid services.

I thank Fritz McCall, director of computing facilities at UMIACS, for supporting our laboratory since the inception of the project. I have also received support from Mike Landavere and Meldavid Manela in the College of Chemical and Life Sciences, where many researchers that use the system are from. I also thank the Office of Information Technology (OIT) for their contribution of resources to the project, and particularly Kevin Hildebrand for his technical expertise.

Last, but certainly not least, I would like to thank my family for all of their love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Overview

Grid computing is a relatively recent formulation of distributed computing, and although there are more formal definitions [40], we use the following one: Grid computing is a model of distributed computing that uses geographically and administratively disparate resources. In Grid computing, individual users can access computers and data transparently, without having to consider location, operating system, account administration, and other details. In Grid computing, the details are abstracted, and the resources are virtualized [15].

The Lattice Project is a Grid computing research project and production system. Among its aims are to unite heterogeneous computing resources into a computational Grid system, so that resources are uniformly usable and addressable. Our Grid is composed of institutional resources, such as clusters and workstations, and resources that are volunteered by users running Berkeley Open Infrastructure for Network Computing (BOINC – http://boinc.berkeley.edu/) software, which is derived from the SETI@home project [4]. We have made a special effort to unite traditional Grid computing with what is known as desktop or volunteer computing, and our work has benefited greatly as a result. Since this research and development work is coming out of the Laboratory of Molecular Evolution, most of our Grid-enabled applications to date have been associated with the life sciences, although nothing about the system design precludes other scientific domain applications from running on the Grid.

There are some important characteristics that make The Lattice Project unique. Whereas most BOINC projects concern themselves with one particular problem, biological or otherwise, we set out to create a generalized Grid system using Globus [18], BOINC, and Condor (http://www.cs.wisc.edu/condor/) that would be capable of running many different applications simultaneously. Most of the applications we run were not originally written with the idea of Grid computing in mind, which presents a unique set of challenges. Since this is a fully-featured Grid system, we have also spent time developing user interfaces, integrating many different resource types (of which BOINC is one), and working diligently to improve other aspects of the system.

1.2 Motivation and Philosophy

As the size and complexity of scientific data has increased, so has the sophistication and computational complexity of data analysis. For example, within the life sciences entire data types that did not exist a relatively short time ago (e.g., complete genome sequences, large-scale microarray experiment results, large multi-locus genotypes) now constitute much of data that is generated. Correspondingly, estimation and inference lead to combinatorial optimization problems and other challenges that have been dealt with using computationally intensive methods (e.g., stochastic simulation, machine learning approaches, Bayesian analysis, Markov-chain Monte Carlo sampling). As a consequence, the computational demands of scientific research continue to increase. Therefore, some scientific researchers are turning to Grid computing to meet their computing resource needs, which is well suited to academic institutions in general [15]. However, there

are several barriers to widespread use of Grid computing in many areas of scientific research, including the lack of Grid-enabled applications and the difficulty of producing them, the deficit of Grid computing resources available for research, and the difficulty of using Grid computing effectively. Several of these barriers to the use of Grid computing are being addressed [9, 23, 32, 33, 36].

Our ongoing Grid computing research and development efforts have been motivated in large part by the computational demands of our own research in computational biology and bioinformatics. This research program focuses on problems in molecular evolution and genetics, which often require approaches that are computationally intensive. Our need for computer resources for our work led to the development of a simple Grid computing system using commodity tools [37], which was used for a large-scale simulation study [14]. Our subsequent work has made use of the Globus Toolkit and BOINC, and has focused on expanding the reach of Grid computing by creating a system that combines these two models [36, 7].

Some of the basic perspectives guiding The Lattice Project are described next.

1.2.1 Computational Resources

With regard to including resources, our approach is simple: we believe that there is a place for every computer to participate in a Grid. The approximate number of computers at the University of Maryland alone is estimated at 40,000, and most of these computers are idle the majority of the time. It is a challenge to convince individuals and organizations within the institution to join the Grid and to lower the barriers to doing so.

We feel that the large heterogeneity in types of research problems is best met

with heterogeneous computational resources. For example, some problems may require a closely coupled parallel computing environment (e.g., a cluster with low latency, high bandwidth interconnections between nodes). Other problems are easily atomized into wholly independent processes, the results of which can be united to form a composite result (e.g., a parameter sweep). These are often called "embarrassingly parallel" problems, and are appropriately handled by desktop computing resources. Hence, we designed the system to include a variety of these resource types.

1.2.2 Software Development

With regard to software development, our approach has been to use open source tools when possible, and to create software that is modular, flexible, and able to adroitly incorporate upgrades to the Globus and BOINC toolkits. Scalability and robustness are also important, especially in Grid computing. We have worked hard to make sure the Grid architecture scales to thousands of simultaneously running jobs, and have also made sure the system is robust enough to run somewhat autonomously and predictably. One can imagine that as a Grid system grows in complexity, there are many possible points of failure that need to be identified and safeguarded against.

1.2.3 User Interface

With regard to using the Grid, we approach it from the perspective of a user familiar with the applications, but not necessarily familiar with Grid computing. Therefore we have striven to make the system easy to use, almost to the point of making it seem like one is running applications as they would on their own system. Most of the analytical applications that scientific researchers are familiar with employ

a command line interface, and we have attempted to provide a similar interface to our Grid services. (Note: in the context of The Lattice Project, a scientific application enabled to run on the Grid is called a Grid service.) Thus, invoking a particular Grid service with a particular string of arguments might exactly mimic the standard use of the application, except that upon hitting return, the Grid takes the program executable, input files, and job description, and sends it off to a remote resource. The person submitting the job is not concerned about where the job is actually running.

1.3 Models of Grid Computing

At present, Grid computing systems can be broadly classified into two models. The first model is the Service Grid, which is considered the "classical" computational Grid system used by the scientific research community. Service Grids provide rich feature-sets (e.g., resource discovery services and multi-user authentication) and tend to concern themselves primarily with providing access to large-scale, intra- and inter-institutional level resources such as clusters or large multiprocessors.

The second model of Grid computing systems is the Desktop Grid, in which cycles are scavenged from idle desktop computers. The power of desktop systems has increased dramatically in recent years, and there has been a concomitant shift away from centralized client/server computing to a decentralized model. Although individual desktops remain inferior to "big iron" machines in many ways (e.g., typically in terms of available memory, amount of mass storage, and interprocessor latency and bandwidth), the combined power of hundreds to millions of desktop systems united in a Desktop Grid represents a substantial computing resource.

Desktop Grids excel at pleasingly parallel problems, and they have become particularly popular in the natural sciences where they have been used in research areas as diverse as radio astronomy [4], phylogenetics [37, 14], structural biochemistry (http://folding.stanford.edu/), and anti-HIV drug discovery (http://fightaidsathome.scripps.edu/).

In contrast to classical scientific research Grid systems, lightweight Desktop Grids provide only a thin layer of abstraction over the resources they manage. This is largely a function of their origins: systems such as SETI@home [4] (and its relatives and descendants) were initially conceived to solve immediate research problems, not as objects of study themselves. Note that we specifically exclude Condor (http://www.cs.wisc.edu/condor/) and similar systems from our definition of Desktop Grids. Although Condor is a distributed computing system that uses cycles from idle computers, the individual computers typically reside wholly within a single institution and administrative domain. (As we will describe later, Condor can play an important role in Grid computing systems, as it does in The Lattice Project.)

Many computational biology and other scientific problems are well suited to processing by Desktop Grids for two main reasons. First, many scientific research problems require considerable CPU time to solve (e.g., large parameter sweeps), and provisioning a cluster or symmetric multiprocessor to provide reasonable response times for a large number of such jobs can be prohibitively expensive and lead to massive over-provisioning during periods when demand for the resource is light. Second, many scientific computing algorithms exhibit extremely coarse-grained parallelism, and many existing applications do not take advantage of the special

features of parallel hardware (e.g., multithreading on symmetric multiprocessor systems). In these cases, the fast interconnect of a symmetric multiprocessor or cluster is simply wasted. Hence, many scientific computing problems would be well suited to Desktop Grid systems if they could be made available and easy to use.

Thus, we have two largely separate models of Grid computing. One provides a rich feature set for accessing large-scale resources; the other provides a minimal feature set but can utilize resources as informal as personal computers in private residences. Ideally, we would like the best of both worlds: we would like to apply the features of the first model over the scope of the latter.

Chapter 2: Related Work

2.1 Service Grids

Service Grids are what one normally thinks of when they think of Grid computing: heavyweight, feature-rich Grid systems that federate a large number of institutional computing resources. Service Grids may be international in scale, such as with the Enabling Grids for E-sciencE (EGEE – http://www.eu-egee.org/) project or the Open Science Grid (OSG – http://www.opensciencegrid.org/), or primarily national, as with the TeraGrid (http://www.teragrid.org/). Usually, the computing resources remain under the ownership and control of participating institutions, and the rights to use those resources are established by some kind of consortium. These Grids exist primarily to advance various domains of science, and since they are built with similar underlying middleware, they often interoperate, sharing compute cycles and data with one another. Typically, researchers must request an allocation to use such Grid systems, or be involved with a large project that already has an active resource allocation. The Grids mentioned here are much larger in scope than The Lattice Project, although are built with similar middleware technology.

2.2 Desktop Grids

Desktop Grids, as we define them, are composed primarily of personal computers volunteered by the general public. These machines run a client program that allows them to receive work from a centralized server they communicate with periodically.

BOINC is the most widely used client/server software for setting up a Desktop Grid system. Most BOINC projects are strongly associated with a particular scientific domain or problem, such as climate prediction (http://climateprediction.net/) or protein folding (http://boinc.bakerlab.org/). A distinguishing characteristic of many of these projects is that they have a vast supply of work that is homogeneous in nature, and thus can easily satisfy the expectations of those who choose to participate. We set out to see if we could include a Desktop Grid in a comprehensive Grid system for scientific analysis. Therefore, participants in our BOINC project may receive work from a wide variety of different scientific applications. This is a bit of a public relations challenge since people do not always know what kind of behavior to expect from the application they may be running, but it is well worth meeting that challenge to include the vast numbers of volunteers that are willing to contribute their computers to the advancement of science.

2.3 Combining Service and Desktop Grids

Aside from The Lattice Project, which was the first project to combine Service and Desktop Grids, there have been very few projects aiming to do something similar. The only one of any note is the Enabling Desktop Grids for e-Science (EDGeS – http://edges-grid.eu/) project, which has come along somewhat recently. Here is the abstract from a recent book chapter, EDGeS: The Common Boundary Between Service and Desktop Grids:

> Service grids and desktop grids are both promoted by their supportive communities as great solutions for solving the available compute power

> problem and helping to balance loads across network systems. Little work, however, has been undertaken to blend these two technologies together. In this paper we introduce a new EU project, that is building technological bridges to facilitate service and desktop grid interoperability. We provide a taxonomy and background into service grids, such as EGEE and desktop grids or volunteer computing platforms, such as BOINC and XtremWeb. We then describe our approach for identifying translation technologies between service and desktop grids. The individual themes discuss the actual bridging technologies employed and the distributed data issues surrounding deployment. [5]

It appears they have been relatively successful in their endeavor thus far. However, their middleware is interoperable with gLite-based [31] Service Grids, whereas our system is the only one known to successfully integrate Desktop Grids with Globus. Next we provide some additional detail about the middleware systems we use in The Lattice Project.

Chapter 3: Middleware Systems

3.1 Globus

The Globus Toolkit [18] represents the current state of the art in Grid middleware. It is the focus of much of the ongoing research in Grid computing, and we can expect to see continued support and development for it well into the future. Based on a web services architecture, Globus provides facilities for the execution and management of jobs on remote resources, resource monitoring and discovery, file transfer, authentication and authorization, and encryption of messages. Using the Globus Toolkit, it is possible to build large, highly distributed, and robust computational grids.

The Globus Toolkit is the paradigmatic example of a heavyweight Grid system. Its Grid Security Infrastructure (GSI) provides for strong, distributed authentication of mutually distrustful parties, and its Community Authorization Service (CAS) provides robust authorization capabilities. The Monitoring and Discovery System (MDS) allows for on-the-fly resource discovery. The Grid Resource Allocation and Management (GRAM) service provides an abstraction layer that allows jobs to be submitted to computational resources without prior knowledge of the underlying job submission and queuing systems used by those resources. The Grid File Transfer Protocol (GridFTP) and Reliable File Transfer (RFT) services enable efficient data transfer, and the Replica Location Service (RLS) enables efficient Grid-wide data management. Globus operates on a push model: work is sent from a submitting node to a computational resource, which then accepts and processes the job, returning the results to the submitter. Moreover, these jobs can be

arbitrary: Globus resources are capable of executing user-supplied code. Input and result files are typically transferred between a submitting node and a computing resource.

Newer versions of Globus (version 3 and onward) support the concept of Grid services, which are closely related to standard web services in both design and implementation. Globus Toolkit 4 is compliant with the Web Services Resource Framework (WSRF), so its Grid services are, in fact, WSRF-compliant web services. Grid services provide a clean way of representing operations that the Grid can perform on behalf of its users; they represent a higher level of abstraction than that of individual computational jobs, and they allow Globus-based Grids to serve as more than large queuing systems.

Over the past several years, our research has been aimed at using the Globus Toolkit, in combination with other Grid middleware, to create a computational Grid for scientific research. We began development with Globus Toolkit 3 (GT3), which formed the backbone of our Grid system. Development continued until we had a fully functional production-level Grid system built around GT3. After successful production use of this system, we focused our efforts on upgrading our infrastructure to use Globus Toolkit 4 (GT4), which was released in early 2005.

3.2 BOINC

The Berkeley Open Infrastructure for Network Computing (BOINC – http://boinc.berkeley.edu/) is the direct descendant of the SETI@home project [4]. Developed by the same group at the University of California, Berkeley that developed SETI@home, BOINC is a generalized implementation of the master/worker,

Internet-scale model that SETI@home popularized. BOINC implements a public-computing Desktop Grid: it harnesses resources outside the bounds of direct institutional control. As in SETI@home, BOINC clients (i.e., personal computers) retrieve jobs to execute from a server that acts as a central repository of work. In contrast to Globus, which uses a push model, BOINC clients *pull* work from a server. Moreover, although BOINC is generalized in the sense that it can manage any arbitrary project, it is limited in that it expects to manage a small number of very large, well-defined projects: its aim is to allow individual research groups to manage SETI@home-style projects without developing their own software [3]. As such, BOINC does not provide mechanisms for executing arbitrary jobs on the fly, for determining which users may modify which jobs, or for any of the other functions one would expect a normal queuing system to provide.

Although BOINC does not support many of the features that Globus does, it does provide the more limited functionality required by its model. For example, BOINC can automatically match work to be processed with hosts suitable to execute it, taking into account estimated memory and disk requirements as well as architecture and operating system constraints. Moreover, BOINC compute clients are expected to be unreliable; both in terms of returning a result in a timely manner and in returning correct results. Therefore, BOINC includes support for redundant computing, in which multiple copies of the same computation are performed by different clients and then cross-checked for agreement.

3.3 Condor

The Condor project from the University of Wisconsin has been around for almost twenty years. Condor is not a Grid middleware toolkit *per se*, but rather a middleware toolkit for distributed computing by means of cycle scavenging. The software has proved to be extremely popular, robust, and useful. We normally use Condor as a queuing system or a job scheduler for resource subsets (Condor pools) comprised of computers in a single administrative domain. The Globus Toolkit includes a Condor scheduler adapter that enables a job submitted via GRAM to run on a Condor pool. This is the primary way that we make use of Condor; we encourage various groups and departments on campus to federate their machines into Condor pools, and then we submit jobs to these pools via the Grid.

As a side note, our GT3-based production Grid system used Condor-G [19] as the Grid meta-scheduler, or "master job queue", although we eliminated the need for this component in the GT4 upgrade. Since the queuing systems on remote resources are sufficient to buffer jobs, the simple scheduling functionality provided by the Condor matchmaking feature can be replaced by a more sophisticated scheduling algorithm, which is precisely what we have done by implementing our own Grid-level scheduler. However, Condor software continues to be an integral, reliable part of the Grid system.

3.4 PBS

The Portable Batch System (PBS – http://www.openpbs.org/) is software that performs job scheduling on compute clusters. PBS runs on several of our clusters and is the third resource manager type that we currently interface with; Condor and

BOINC are the other two. Other popular queuing systems that operate in a similar manner are Sun's Grid Engine (SGE – http://gridengine.sunsource.net/) and Platform's Load Sharing Facility (LSF – http://www.platform.com/). It would be possible to integrate resources running these other queuing systems, too.

Chapter 4: The Lattice Project

4.1 Features

We provide users with a command line interface for submitting and monitoring jobs on a machine we call the "Grid Brick". After logging in, a user may run commands that submit jobs to the Grid, monitor the status of jobs, or remove jobs from the system. By convention, the user's home directory is the staging area for Grid input and output data. A user typically uploads input files to the Grid Brick and organizes them in some manner before conducting their analyses. As their compute jobs complete, result files are automatically returned to the directory the job was submitted from. We also make available web pages on the Lattice intranet for monitoring job status (Figure 4.1).

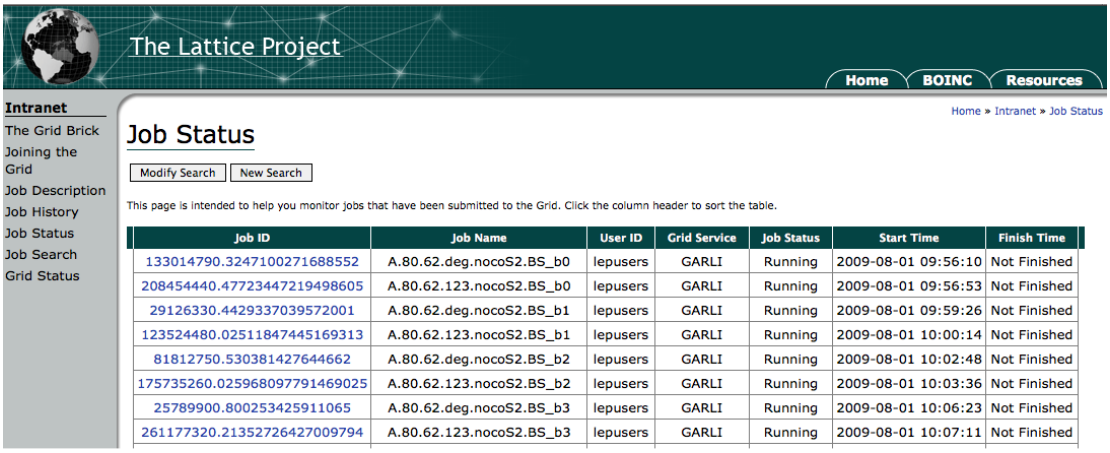


Figure 4.1. A screenshot from the Lattice intranet showing the current status of jobs.

4.1.1 Job Types

Running a Grid-enabled application using our command line tools is usually very similar to using the original program, since most of the same command line arguments will be present. Grid users are able to submit the job in an MPI mode if

that is supported by the application, and are able to submit entire batches of jobs with a single command if that is supported by the Grid service. We have found that a Grid user often needs to submit many replicates of a particular job type, so enabling support for batch submission has been a high priority. Since some algorithms are stochastic, some job batches will be completely *homogeneous* – i.e., the same combination of program executable, input files, and parameters will produce different results for each job replicate. Homogeneous job batches were relatively easy to implement and support, but it is easy to imagine any number of ways a *heterogeneous* job batch could be devised – using different input files with each job replicate, for example, or varying program arguments to conduct a parameter sweep. We currently support heterogeneous batches that use a different combination of input files for each job in the batch. Batch functionality is essential because it makes scheduling more efficient, eliminates redundant data transfers, simplifies record keeping, and generally speeds up the flow of work through the system. The majority of our researchers make use of batch submissions, so the utility of this feature has been proven.

4.1.2 Job Submission

Once a user has organized their input files and authenticated themselves using their X.509 GSI certificate, they may use the `lattice_submit` script to submit a job to the Grid. They may also require help to use various services, as in the following example, which shows the usage of the Structure [47] Grid service:

```
gridtest@valine:>lattice_submit Structure --help
Usage: Structure [OPTIONS]
-K n                    Change the number of populations
-L n                    Change the number of loci
-N n                    Change the number of individuals
-e filename             Read a different parameter input file
                        instead of 'extraparams'
```

```
-i filename              Read a different input file
--jobname jobname        The arbitrary name to assign this job or
                         batch of jobs.
-m filename              Read a different parameter input file
                         instead of 'mainparams'
-o filename              Print results to a different output file
--replicates replica     The number of times to execute this job.
```

All of the flags are optional. Default behavior is to search the current working directory for a file named `mainparams` and a file named `extraparams`, one of which must specify the input data file. The `--replicates` flag is used to submit batches of jobs, either homogeneous or heterogeneous. In the case of a heterogeneous batch, instead of specifying a single file, a directory of files may be specified for the following arguments: `-m`, `-e`, and `-i`. Here is an example of a heterogeneous batch submission:

```
gridtest@valine:>lattice_submit Structure --replicates 10
                 -m mainparam_dir/ -e extraparams -i myinputfile
```

In this example, `mainparam_dir` must have at least 10 differently named parameter files in it (whose contents are also hopefully different; perhaps the user is varying a parameter, renaming the output file, or varying the random seed). `extraparams` and `myinputfile` stay constant and are used unchanged by each job replicate in the batch.

The Grid user does not have to specify the resource on which the job is run; this is for the Grid meta-scheduler to determine. Next we describe tools for monitoring the status of submitted jobs.

4.1.3 Job Monitoring and Management

We have provided a utility called `job_status` that reports the status of jobs in the system. This script uses a combination of command line arguments to filter results and display them to the user. If no command line arguments are given to the program,

it will list the jobs of the current user from the past 30 days. This list will include jobs that are idle, running, completed, retrieved, and failed. Because this list has the potential to get very long, it is recommended that the user provide some filters. For example, issuing the command `job_status --user [username]` will show only the jobs submitted by a given user.

List View

The following is a sample run of `job_status` that prints out a list of jobs submitted by user `freed`. Several fields are displayed when jobs are listed in this manner. The JobID is a unique identifier for a specific job, and can be used in conjunction with the `kill_job` script to remove a job from the system for any reason.

```
gridtest@valine:>job_status --user freed

                        JobID      User      Started       App      Status     Finished    Job Name
-------------------------------------------------------------------------------------------------
   536740582.9558141272948417     freed  02/14 13:15      MDIV        Idle           --     khoisan

Total Jobs: 1  Running: 0  Idle: 1  Finished: 0  Retrieved: 0  Failed: 0
```

Information View

In addition to the listing view, `job_status` also contains an "info" view that lists more details about the job(s), which is activated with the `--info` flag. This information can be used to help resubmit jobs that fail, to discover where jobs have been scheduled, to inspect the command line that was used, and so forth.

```
gridtest@valine:>job_status --jobid 536740582.9558141272948417 --info
-------------------------------------------------------------------------------
JobName: khoisan
JobID: 536740582.9558141272948417
Submitted by: freed
Submitted on: 2005-02-14 13:15:47
Finished:
Status: Idle
Scheduler: https://128.8.120.35:8443/wsrf/services/ManagedJobFactoryService
Resource: BOINC
BOINC Credit: None yet
Application: MDIV
Command line: --steps 10000000 --maxT 5.0 --maxM 10.0 --model HKY --burnin 1000000 --random 2375625 MDIV-BG-MW.txt
```

Ordering Results

Jobs can be sorted by using the `-o` or `--order` flag. The output generated by `job_status` is sortable on eight values. A comma-separated list can be used to sort on more than one value. The sortable values are job ID (`id`), job start time (`start_time`), user who submitted the job (`userid`), current status of the job (`status`), job finish time (`finish_time`), name of the job (`job_name`), BOINC credit assigned to the job (`credit`), and the resource the job was assigned to (`resource`). Thus, to sort first by user name, then by job status, then by job start time, the following string would be used: `"userid,status,start_time"`. For example:

```
gridtest@valine:>job_status -u freed -o "status,start_time"
                        JobID      User      Started       App      Status     Finished    Job Name
-------------------------------------------------------------------------------------------------------
   536740582.9558141272948417     freed  02/04 11:52      MDIV   Retrieved  02/10 20:10     khoisan
   282587440.7911062768208433     freed  02/04 11:53      MDIV   Retrieved  02/07 09:38     khoisan
   36782170.34160974924516145     freed  02/04 11:54      MDIV   Retrieved  02/09 15:07     khoisan
   261021060.8170057331660435     freed  02/04 11:58      MDIV   Retrieved  02/07 09:38     khoisan
   265995320.5873708825896095     freed  02/04 11:59      MDIV   Retrieved  02/07 09:38     khoisan
   161120550.0364442403973666     freed  02/04 12:05      MDIV   Retrieved  02/07 09:38     khoisan
```

Viewing Old Jobs

When `job_status` is run without any arguments, some default values are used. As discussed earlier, `job_status` assumes the user is only interested in their own jobs. In addition to this assumption, "old" jobs are also filtered out automatically. By default, only jobs that were started within the past 30 days (or jobs that have not yet completed) are shown when `job_status` is run. Jobs that were started more than 30 days ago can still be accessed via the `-d` or `--days` flag, which can be combined with other command line arguments. To check jobs owned by `gridtest` that were started in the past 10 days, one would issue the following command:

```
gridtest@valine:>job_status -u gridtest -d 10
```

The functionality provided by `job_status` has also been made available through a web interface (Figures 4.1 and 4.2).

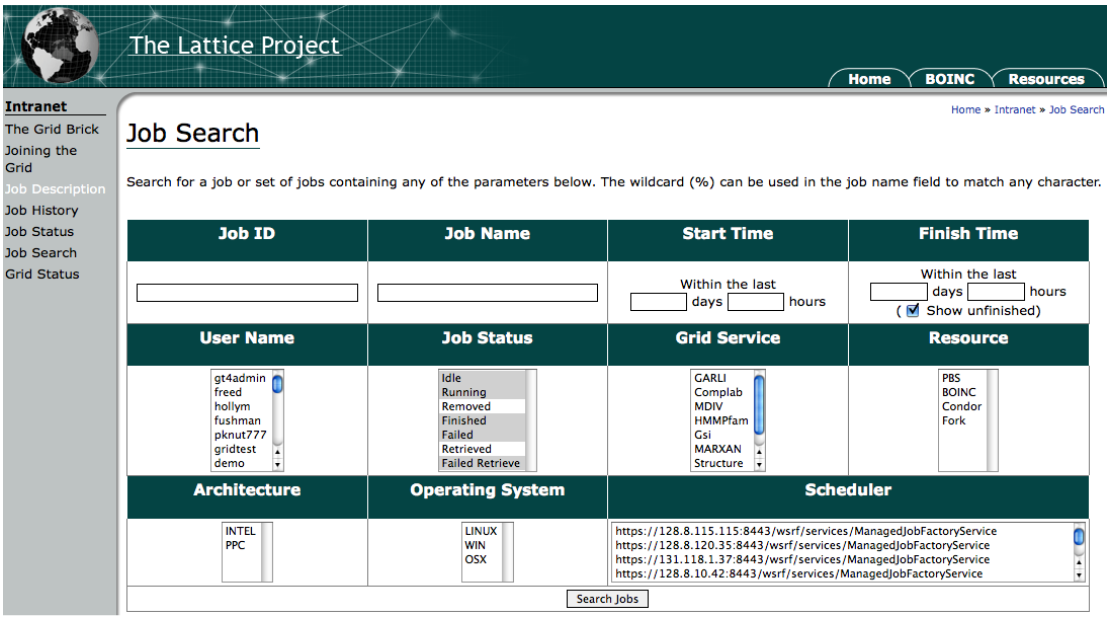


Figure 4.2. A screenshot from the Lattice intranet showing a number of job search filters.

4.2 Architecture and Infrastructure

This section discusses the general architecture of the system, and also provides a description of the infrastructure that currently makes up the core Grid system. The architecture of the system is a general client, server, resource model; that is, a Grid client invokes a Grid service in a particular way and transfers data to the server, which in turn schedules the job (or batch of jobs) to an appropriate resource. Figure 4.3 shows an architectural diagram of the system. The infrastructure required to support the general architecture is modest but can be scaled up as necessary. (Here we only describe the machines in our production system; we also have a development system intended for testing services and new functionality before it is deployed.)

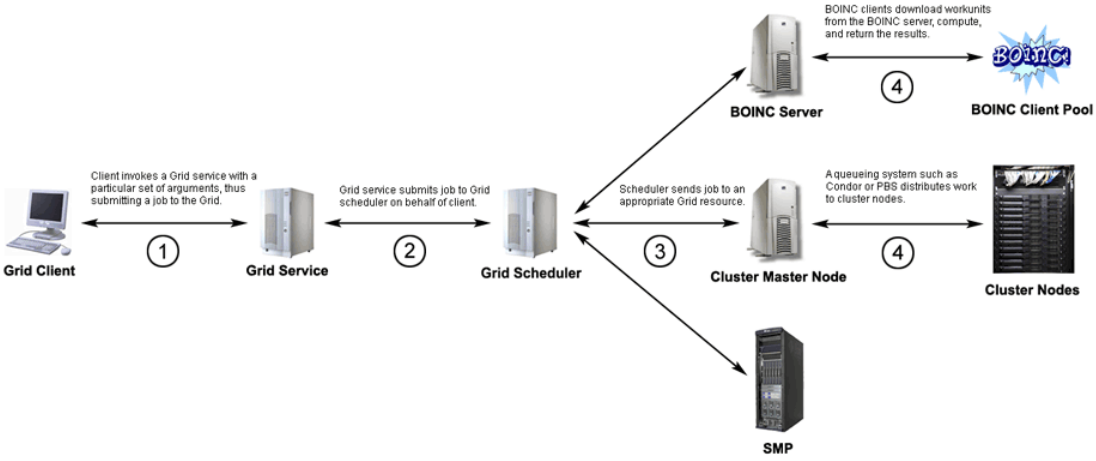
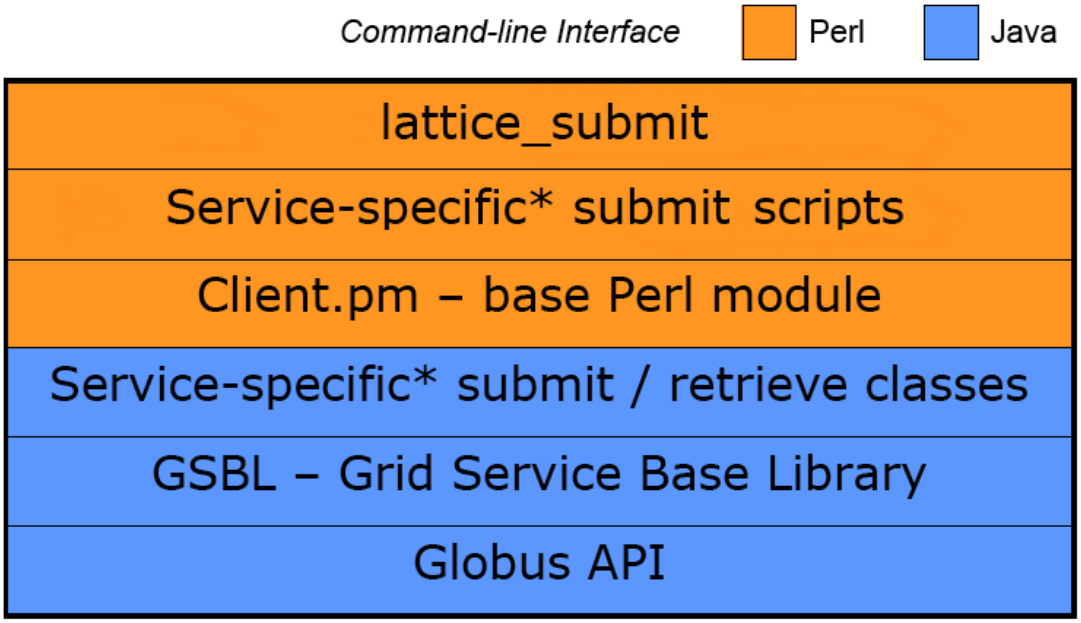


Figure 4.3. As shown in the diagram, data generally flows from left to right and back again through the system, i.e., from client to server to resource and back. Despite the fact that they are represented separately in the diagram, the Grid service and the Grid scheduler are both located on the Grid server in our current production system.

4.2.1 Grid Client

The client in the architectural diagram and the “Grid Brick" mentioned in section 4.1 are the same entity. The original such machine, `valine.umiacs.umd.edu`, is an HP dual core Intel Xeon workstation running RHEL5 at 3.1 GHz with 2 GB of RAM and over 100 GB of disk space allocated for user home directories. (There is also currently one other Grid Brick in the College of Chemical and Life Sciences.) These machines have sufficed to accommodate our small user base so far, but scaling up will require more storage for user data and additional points of submission. Other possibilities include a more ubiquitous command line interface, or a web interface for job submission and data management, which are discussed in Chapter 7. In terms of Globus software, the Grid Brick needs to have GSI libraries installed for authenticating Grid users, and it runs a GridFTP server in order to transfer files to and from the Grid server. It does not, however, need to run a Globus web services container. Our own software, to be described later, makes calls to Globus libraries to

facilitate job submission and lifecycle management, and thus is also present on the Grid Brick, along with various utility scripts (Figure 4.4).



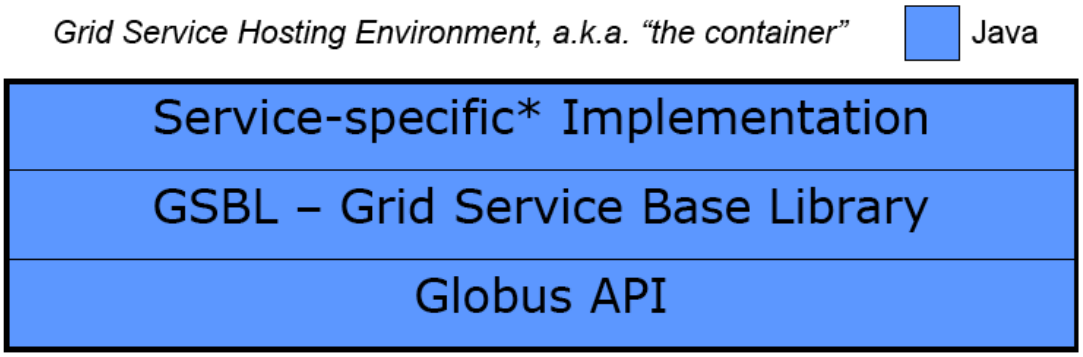
*** Service-specific templates and stubs are created by the Grid Service Generator**

Figure 4.4. Grid client software stack.

4.2.2 Grid Server

The main Grid server is `asparagine.umiacs.umd.edu`, and it is similar to the Grid Brick except that its processors are 2.5 GHz and it has 4 GB of RAM. (Aside: `aspartate.umiacs.umd.edu`, a sister server with 8 GB of RAM, powers the Lattice BOINC Project.) The Grid server has an 80 GB disk partition for transitory job storage. It runs the full Globus software stack, including a web services container into which Grid services are deployed (Figure 4.5). Since jobs are submitted directly from the Grid server to one of many computational resources, it is important that firewalls be configured to allow traffic on various ports (e.g., 8443–https, 2811–GridFTP) between the Grid server and remote resource. One can easily imagine a distributed,

decentralized model in which multiple Grid servers are actively functioning, but so far this has not been necessary.



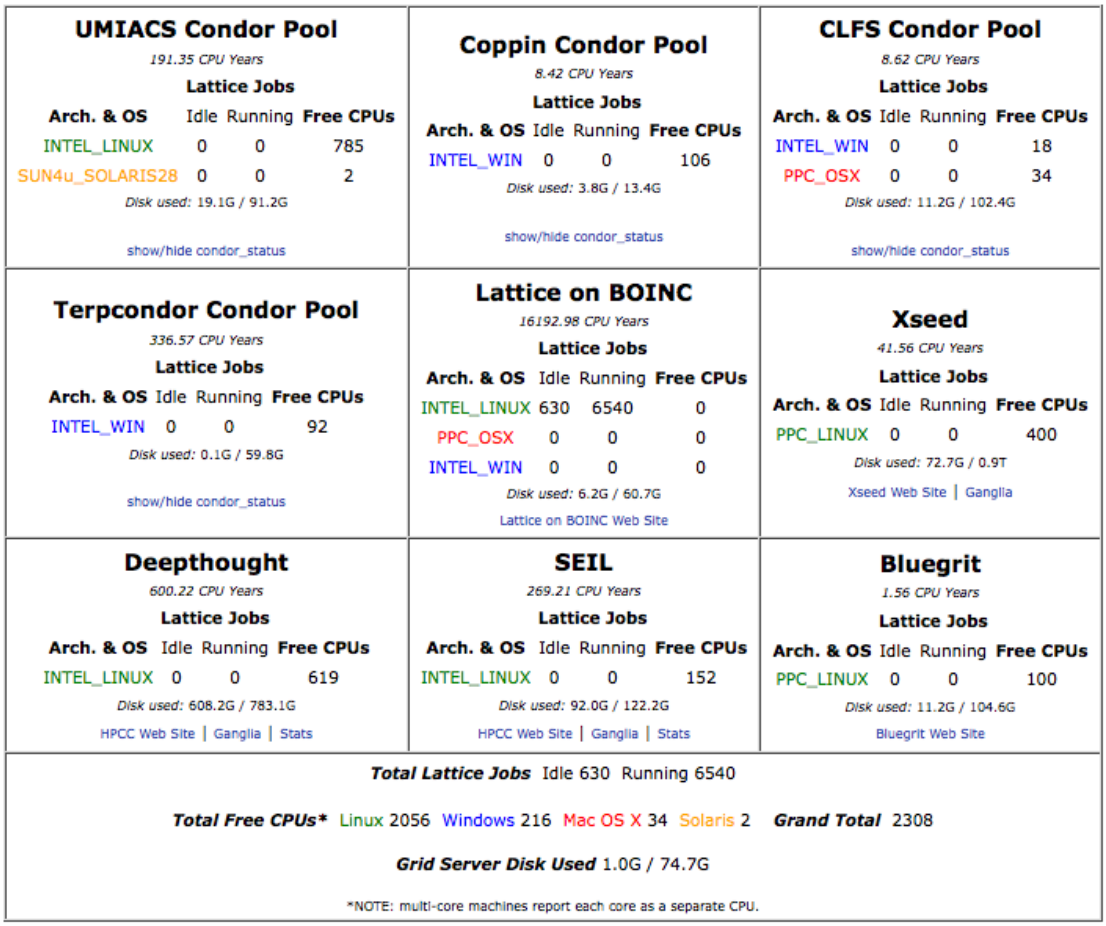
*** Service-specific templates and stubs are created by the Grid Service Generator**

Figure 4.5. Grid server software stack.

4.2.3 Grid Resources

On the Lattice web site we provide an up to the minute view of our Grid resources (Figure 4.6). There are currently eight distinct resources where jobs can run. We have compiled a table of current resources (Table 4.1), prospective resources (Table 4.2), and retired resources (Table 4.3). These tables include information about the size of the resources, the institution to which they belong, and individuals to contact for more information. Resource building is one of the principal activities associated with creating and expanding a Grid system, and as such it has been one of our highest priorities. Beyond simply aggregating CPU power, resource building usually involves collaboration between different people and organizations. Such people may not know anything about Grid computing, in which case we take the time to explain the goals of the project, the benefits of being involved with it, and the technical details that enable these groups to effectively contribute their local resources to the project. However, it is important to define exactly what constitutes a

local resource, how one can be created, and the kinds of policies and procedures that govern its use, which we do next.



UMIACS Condor Pool

191.35 CPU Years

Lattice Jobs

Arch. & OS	Idle	Running	Free CPUs
INTEL_LINUX	0	0	785
SUN4u_SOLARIS28	0	0	2

Disk used: 19.1G / 91.2G

show/hide condor_status

Coppin Condor Pool

8.42 CPU Years

Lattice Jobs

Arch. & OS	Idle	Running	Free CPUs
INTEL_WIN	0	0	106

Disk used: 3.8G / 13.4G

show/hide condor_status

CLFS Condor Pool

8.62 CPU Years

Lattice Jobs

Arch. & OS	Idle	Running	Free CPUs
INTEL_WIN	0	0	18
PPC_OSX	0	0	34

Disk used: 11.2G / 102.4G

show/hide condor_status

Terpcondor Condor Pool

336.57 CPU Years

Lattice Jobs

Arch. & OS	Idle	Running	Free CPUs
INTEL_WIN	0	0	92

Disk used: 0.1G / 59.8G

show/hide condor_status

Lattice on BOINC

16192.98 CPU Years

Lattice Jobs

Arch. & OS	Idle	Running	Free CPUs
INTEL_LINUX	630	6540	0
PPC_OSX	0	0	0
INTEL_WIN	0	0	0

Disk used: 6.2G / 60.7G

Lattice on BOINC Web Site

Xseed

41.56 CPU Years

Lattice Jobs

Arch. & OS	Idle	Running	Free CPUs
PPC_LINUX	0	0	400

Disk used: 72.7G / 0.9T

Xseed Web Site | Ganglia

Deepthought

600.22 CPU Years

Lattice Jobs

Arch. & OS	Idle	Running	Free CPUs
INTEL_LINUX	0	0	619

Disk used: 608.2G / 783.1G

HPCC Web Site | Ganglia | Stats

SEIL

269.21 CPU Years

Lattice Jobs

Arch. & OS	Idle	Running	Free CPUs
INTEL_LINUX	0	0	152

Disk used: 92.0G / 122.2G

HPCC Web Site | Ganglia | Stats

Bluegrit

1.56 CPU Years

Lattice Jobs

Arch. & OS	Idle	Running	Free CPUs
PPC_LINUX	0	0	100

Disk used: 11.2G / 104.6G

Bluegrit Web Site

Total Lattice Jobs Idle 630 Running 6540

Total Free CPUs* Linux 2056 Windows 216 Mac OS X 34 Solaris 2 ***Grand Total*** 2308

Grid Server Disk Used 1.0G / 74.7G

*NOTE: multi-core machines report each core as a separate CPU.

Figure 4.6. A live snapshot of our Grid resources, available at http://lattice.umiacs.umd.edu/resources/.

A *local resource* is defined as an established computing resource administered in one domain and capable of functioning independently from a Grid system. Users of a local resource submit and monitor compute jobs using a local resource manager (LRM), often simply called a "scheduler". Pools of computers running Condor software or dedicated clusters running PBS software are common local resources. A typical Grid system contains a meta-scheduler, which is discussed at length in section

5.3. Meta-scheduling is the process of assigning computational jobs submitted at the Grid level to an eligible local resource, where the job is then rescheduled locally. This kind of hierarchical functionality is what makes Grid computing appealing: it is the ability to use many different resources simultaneously and efficiently, wherein the Grid system handles user authentication and authorization, job scheduling and monitoring, and data placement. The Lattice Project provides these basic features, thus enabling the student, scientist, or researcher to perform a large amount of computation in a short amount of time without having to worry about low level details. In addition, the user gains access to computational resources outside of his or her administrative domain. In the case of The Lattice Project, one of these resources is a fully functional BOINC project capable of running Grid jobs (Figure 4.7).

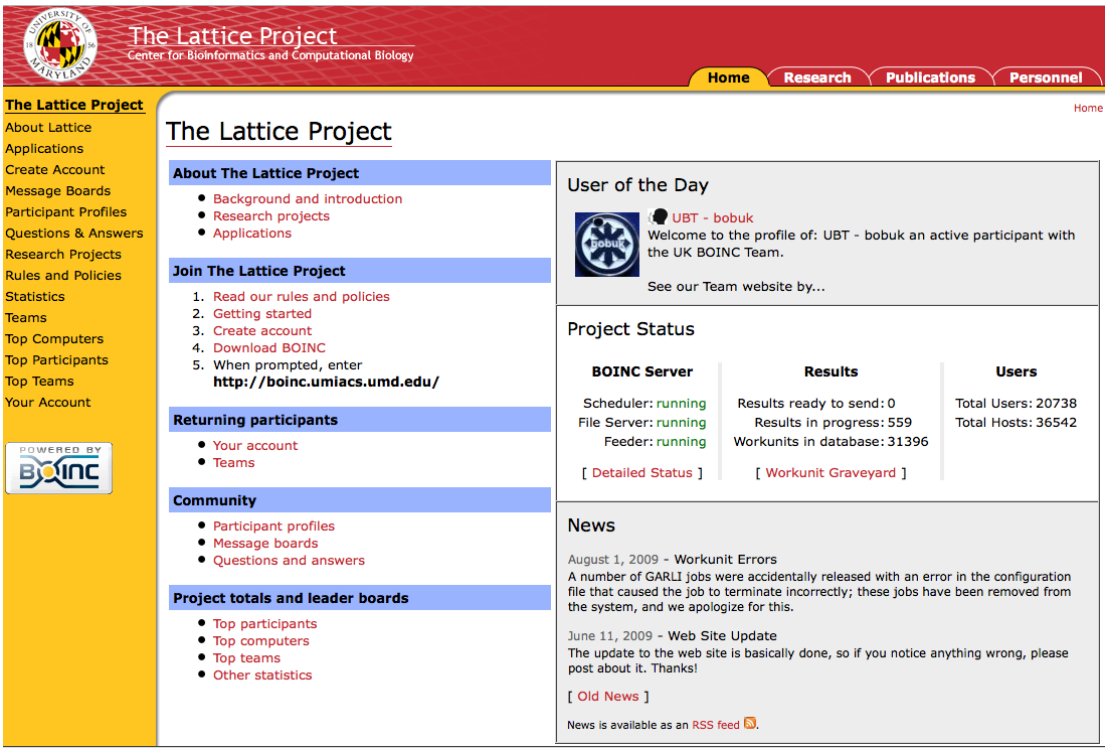


Figure 4.7. Any member of the general public with a computer may participate in The Lattice Project by signing up at our web site (http://boinc.umiacs.umd.edu) and downloading BOINC client software.

Current Grid Resources

RESOURCE	INSTITUTION	APPROX. SIZE	CONTACT
UMIACS (Condor Pool)	UMCP	600 CPUs	Fritz McCall fmccall@umiacs.umd.edu
Terpcondor (Condor Pool)	UMCP	230 CPUs	Josh Thornton joshthor@umd.edu
CLFS (Condor Pool)	UMCP	120 CPUs	Meldavid Manela mel@umd.edu
CSU Desktops (Condor Pool)	Coppin State	700 CPUs	Ahmed El-Haggan aelhaggan@coppin.edu
Xseed (PBS Cluster)	Bowie State	448 CPUs	Sadanand Srivastava ssrivastava@bowiestate.edu
Deepthought (PBS Cluster)	UMCP	1600 CPUs	Kevin Hildebrand kevin@umd.edu
SEIL (PBS Cluster)	UMCP	300 CPUs	Jeff McKinney kevin@umd.edu
Lattice on BOINC (BOINC Pool)	UMCP	3000+ CPUs	Adam Bazinet pknut777@umiacs.umd.edu

Table 4.1. Current computational resources – four Condor pools, three PBS clusters, one BOINC pool.

Prospective Grid Resources

RESOURCE	INSTITUTION	APPROX. SIZE	CONTACT
Bowie State (Condor Pool)	Bowie State	unknown	Marivic Weiss MWeiss@bowiestate.edu
CSEE Desktops (Condor Pool)	UMCP	unknown	Jeff McKinney mckinney@umd.edu
RIT Desktops (Condor Pool)	RIT	unknown	Gurcharan S Khanna gskpop@rit.edu
Kansas Cluster (Cluster)	University of Kansas	unknown	Dave Vieglais vieglais@ku.edu

Table 4.2. Prospective computational resources – three Condor pools, one cluster.

Retired Grid Resources

RESOURCE	INSTITUTION	APPROX. SIZE	CONTACT
Gridiron (Condor Pool)	UMCP	100+ CPUs	Kevin Hildebrand kevin@umd.edu
USM Desktops (Condor Pool)	USM	10 CPUs	Suresh Balakrishnan suresh@usmd.edu
Bluegrit (PBS Cluster)	UMBC	128 CPUs	John Dorband dorband@umbc.edu

Table 4.3. Retired computational resources – two Condor pools, one cluster.

We have attempted to be all-inclusive with regard to the types of resources that can be integrated into our system. Currently we support resources federated with Condor, PBS, or BOINC. Once a scheduler is installed on a local resource, that resource must be tied into the Grid system by installing Globus on a resource node that has the capability to submit jobs using the local scheduler. Once a Globus web services container is running on that node, jobs may be submitted to the resource by contacting the ManagedJobFactoryService at a particular URL (e.g., `https://[host-IP]:8443/wsrf/services/ManagedJobFactoryService`).

Installing Globus and configuring the necessary components is not a trivial task, but we have created detailed instructions to ease the process. One of these components is called a *scheduler adapter* (formerly *job manager*). There exists a different scheduler adapter for each resource type. This is typically a collection of scripts responsible for translating a generic job description (e.g., Globus RSL or JSDL format) into a resource-specific job description (e.g., a Condor or PBS submit file). The Globus Toolkit comes with several of these by default, of which we have customized and extended the Condor and PBS adapters. Our Globus-BOINC interface includes, naturally, a BOINC scheduler adapter, which we wrote from

scratch. Another important resource-specific component is the *scheduler provider*, which collects information about the current state of a resource – e.g., number of free CPUs, total RAM, total disk space, and so on. This information is aggregated by the Globus MDS service to produce a high-level view of the current state of the computational resources on the Grid, which is summarily used for job scheduling and monitoring. Once Globus is installed and configured, the resource should be able to receive jobs from the Grid. We take measures to ensure that Grid jobs do not interfere with local use of the resource; some of this responsibility may fall to the scheduler itself, or things may be explicitly configured such that Grid jobs only backfill, or take the lowest priority. Once a resource is configured, it generally does not require much additional maintenance as a result of its integration with the Grid. We are currently installing the most recent version of the Globus Toolkit in the 4.2.x line. Of course, these Globus installations may need to be periodically upgraded, but usually only when a new major version of Globus is released.

Here are some facts about our resources:

- We support three major platforms: Linux (both PowerPC and Intel-based), Windows, and Mac OS (both PowerPC and Intel-based). There is also a smattering of Solaris machines.
- Three different institutions are currently tied in to the Grid: UMCP, Bowie State University, and Coppin State University.
- Within UMCP, several groups have contributed resources: UMIACS, OIT, CLFS, PSLA, and ECE/ISR.
- We currently have four Condor pools, three dedicated clusters, and a BOINC project with a steadily growing number of participants.
- We currently have a total of 4000-5000 CPUs.
- A sizeable subset of our resources has R [48] installed, and we have developed unique methods for running R code on Condor resources.

Given the growth and success of our BOINC project, it seems clear that our resource building efforts going forward should primarily focus on the integration of dedicated, MPI-capable resources or resources with other specialized capabilities to complement the high throughput capability of BOINC. As users of the Grid system increase, so will demand for resources. The BOINC pool can easily grow to meet this demand, but it will be necessary to continue to seek out other resources and new institutional partners. These resources are already paid for; they are simply underutilized. In light of this, most people are receptive to the idea of joining a Grid.

Participants sharing computing resources derive a number of benefits. First and foremost: if a group contributes computing resources to the Grid, they are eligible to use *all* Grid resources. Therefore, they gain access to many more computing resources than they previously had access to, including a very large pool of public computing clients through the Lattice BOINC Project. Joining the Grid might obviate the need for future hardware purchases (e.g., a new cluster). Some participants do not have a surplus of compute jobs, but instead have the opposite problem: they have purchased a cluster that is underutilized, and would like to increase its utilization rate. By contributing such resources to the Grid system, they are making them available to many more people. In addition, we believe that compute resources in a Grid system with an intelligent scheduler are used more efficiently. For example, jobs with large memory requirements can be sent to clusters with large memory nodes, and tightly coupled jobs (e.g., MPI jobs) can be sent to clusters with fast interconnects. Pleasingly parallel jobs can be sent to Condor pools or to the BOINC pool, and so on. In this manner, all computing resources are used more productively.

Chapter 5: Core Functionality

5.1 Grid Services

5.1.1 Overview

In the context of The Lattice Project, a *Grid service* usually refers to a single application that has been Grid-enabled, a relatively involved process that we have worked hard to streamline. However, it is worth stressing – in order for an application to run on the Grid, a Grid service for that application must be written and deployed – i.e., submission of arbitrary code is not currently supported at the Grid level. We have created 23 Grid services (see Table 5.1 for a brief description), though many early ones were for proof of concept, and so fewer than half have been run to any appreciable extent (as reflected by their CPU year credit – see Table 5.2). Grid users may have a research project that requires the creation of one or more Grid services, so frequently services are created on demand.

In keeping with standard web services procedures, we have designed our Grid system with a generalized client-service architecture in mind. A remote Grid client invokes a set of operations that cause a particular application to be run on the Grid. These operations are performed during job setup, submission, monitoring, and cleanup, and they fall into the following areas: initial configuration of Grid client-service interaction, argument processing, transfer of files between client and service, and submission and monitoring of Grid Resource Allocation and Management (GRAM) jobs by the service.

BLAST (Basic Local Alignment Search Tool): a sequence database search program. [1, 2]
Clustal W: a multiple sequence alignment program. [57]
CNS (Crystallography & NMR System): a program for molecular structure determination. [13]
GARLI (Genetic Algorithm for Rapid Likelihood Inference): a phylogenetic analysis program. [60]
gsi (Genealogical Sorting Index): a program for a statistical analysis of evolutionary trees. [16, 8]
IM (Isolation with Migration): a population genetics estimation program. [24]
LAMARC (Likelihood Analysis with Metropolis Algorithm using Random Coalescence): a population genetics estimation program. [28-30]
MARXAN: a program used to design reserves for biodiversity conservation. [6, 46]
MDIV (Migration and Divergence), a population genetics estimation program. [41]
Migrate: a population genetics estimation program. [11, 12]
Modeltest: a program for evaluating the fit of evolutionary models. [45]
MrBayes: a phylogenetic analysis program. [53]
ms: a population genetics simulation program. [26]
MUSCLE: a multiple sequence alignment program. [17]
PAUP*: Phylogenetic Analysis Using Parsimony (*and other methods), a phylogenetic analysis program. [56]
PHYML: a phylogenetic analysis program. [22]
Pknots: an RNA structure prediction program. [52]
Seq-Gen: a sequence simulation program. [49]
Snn: a population genetics estimation program. [25]
SSEARCH: a pairwise sequence alignment program. [43, 55]
Structure: a population genetics inference program. [47]

Table 5.1. A short description of our Grid services.

Application	Ported To BOINC	Ported To			CPU Years[0]
		Linux X86	Windows	Mac OS X	
BLAST[1]	No	Yes	No	No	N/A
Clustal W	Yes	Yes	Yes	Yes	N/A
CNS	Yes	Yes	Yes	No	**25.43**
Complab[2]	No	Yes	Yes	Yes	**6.66**
GARLI	Yes	Yes	Yes	Yes	**4912.76**
gsi[3]	No	Yes	Yes	Yes	**142.45**
HMMPfam	Yes	Yes	Yes	Yes	**8193.54**
IM	Yes	Yes	Yes	Yes	**0.18**
LAMARC	Yes	Yes	Yes	Yes	N/A
MARXAN	Yes	No	Yes	No	**5248.17**
MDIV	Yes	Yes	Yes	Yes	**13.25**
Migrate-N	Yes	Yes	Yes	Yes	**0.00**
Modeltest	Yes	Yes	Yes	Yes	N/A
MrBayes	Yes	Yes	Yes	Yes	N/A
ms	Yes	Yes	Yes	Yes	N/A
Muscle	Yes	Yes	Yes	Yes	N/A
PAUP*[4]	No	Yes	No	No	N/A
Phyml	Yes	Yes	Yes	Yes	N/A
Pknots	Yes	Yes	Yes	Yes	N/A
Seq-gen	Yes	Yes	Yes	Yes	N/A
S_{nn}	Yes	Yes	Yes	Yes	N/A
ssearch	Yes	Yes	Yes	Yes	N/A
Structure	Yes	Yes	Yes	Yes	N/A

0 The Lattice Project has performed **18542.44** CPU Years of computation.
1 BLAST has not been ported to BOINC because it requires pre-staged databases.
2 Complab has not been ported to BOINC because it is implemented in Java.
3 *gsi* has not been ported to BOINC because it is implemented in R.
4 PAUP* has not been ported to BOINC because of licensing restrictions.

Table 5.2. A list of our Grid services, the platforms to which they have been ported, and a measure (in CPU Years) of how much they have been used in production.

5.1.2 The Challenges of Working with Globus

As might be expected in research-grade software, there are problems with the Globus Toolkit. First, the application programming interface (API) that Globus provides for writing Grid services is a relatively low-level one, and accomplishing common tasks (such as transferring a file between two systems) can often require a lot of code. Writing a fully-featured application-based Grid service is not as easy as we would like it to be.

Second, Globus uses an asynchronous, event-based model for programming Grid services. Although such a model is well suited to Grid computing, where one may have to wait unknown lengths of time for operations to complete (e.g., between submitting a job and receiving the results), it is not necessarily the most intuitive programming model. In many cases the task of writing Grid services will be facilitated if it can be done using a procedural model with blocking function calls, even if the underlying infrastructure is event-based.

Third, because the Globus Toolkit software is under continual development, there is always the possibility that the API presented to Grid services will change between versions. This is precisely what happened between GT3 and GT4. A high perceived probability of API change can make programmers hesitant about writing Grid services using the API. Finally, creating a new Grid service requires creating a number of new files in a very specific directory structure and with very specific names, namespaces, and classes. This is a tedious and error-prone process at best, but one we have to repeat each time we write a Grid service. Moreover, because we are interested in having our applications run in a general framework, we designed our

Grid system around the idea that every Grid-enabled application would be presented as a Grid service. Thus, since we knew we would be building a significant number of services, it was desirable to reduce the overhead associated with this process as much as possible.

5.1.3 Grid Services Base Library

To address the above problems, we have written the Grid Services Base Library (GSBL) [9], which provides a high-level, procedural API for writing Grid services. In our Grid system, GSBL is the API called by our body of Grid services; at this level, no Globus code is invoked directly. Thus, in the event that the Globus API changes, only GSBL will require updating. It should also be noted that the Globus team tries to preserve concepts from version to version of the toolkit, which means that high-level GSBL-supported operations should also migrate easily. This solves the problem of a changing API.

Admittedly, we have not attempted to provide a friendly interface to the entire Globus API or to support all possible operations. As a guiding principle of our API design we have focused on making simple and common tasks easy to implement, while leaving the programmer to the Globus API for more difficult and uncommon tasks. We note, however, that after having built more than twenty production Grid services for life science applications, we have yet to encounter the need to circumvent GSBL to write custom Globus code. In the rest of this section, we discuss the GSBL API and how it solves the problems associated with the low-level, event-based programming model of Globus.

5.1.4 Initial configuration of client-service interaction

There are several steps that a Globus Grid client needs to take in order to establish communication with a Grid service. Because our Grid services are implemented with the WS-Resource Framework (Web Services Resource Framework, WSRF), these services provide users with the ability to access and manipulate state (i.e., data values that persist across service interactions). Following standard GT4 conventions, each of our Grid services is composed of a Factory service and an Instance service. When a client requests resource creation, it contacts the Factory service. When a client requests that an operation be performed on a specific resource, it contacts the corresponding Instance service.

Thus, assuming the WS-Resource Factory pattern is in use, the client first contacts a Factory service that in turn creates and initializes a new resource. The Factory service returns an endpoint reference to a WS-Resource composed of an Instance service and the recently created resource. The interface of the Instance service object has been defined in Web Services Description Language (WSDL), and the associated resource provides state for this particular Grid service Instance. This process requires a significant amount of relatively dense code that is nearly identical between Grid services. Unfortunately, although the overall logic remains constant, the classes involved do change, because each Grid service is uniquely typed. Moreover, there is no supertype for the classes, and the names of the functions to be called depend on the name of the service (e.g., one has to call `get[SvcName]FactoryPortTypePort()` and `get[SvcName]PortTypePort()`), so placing this logic into a library is not straightforward: neither subclassing nor

templating is effective.

In order to place this code in a library, we made use of the Java Reflection APIs. The constructor for the Grid client base class takes as parameters a Class object representing the type of the class used to contact the Factory service, and a Class object representing the type of the class used to contact the Instance service; using these objects, it can create new instances of these classes without prior knowledge of their type. To call the creation method (whose name varies based on the name of the Grid service), we use the reflection API to search the methods of the locator object for a method whose name and signature match that which is needed; then we obtain a reference to this method and call it on the object. To reiterate, when this initial setup is complete, a new Grid resource will be created for this particular job request and a handle to an Instance service will be returned to the client. This handle is used to contact the Grid service when performing the operations discussed in the next few sections.

5.1.5 Argument Processing

The applications most often used in our computational biology research can frequently accept a large number of command line arguments (e.g., SSEARCH, part of the FASTA package [42], has 24 arguments). The straightforward Globus solution to representing these parameters in a Grid services context would be to create a complex type to hold them, and pass an instance of this complex type from the client to the Grid service. Although this approach is adequate in many cases, it does not fully meet our needs. Defining a type to handle configuration parameters is helpful, but when such a type has dozens of fields, some sort of additional support is needed:

manually copying user input into and out of such a type becomes tedious and error-prone.

In order to provide more robust support for configuration parameters, we chose to create a separate XML file describing the parameters. Each parameter has a corresponding record in the file giving the name of the parameter, its description, and to facilitate understanding, the name of the flag that the parameter corresponds to in the original program. A sample record appears as follows:

```
<argument key="dbSize">
<flag>Z</flag>
<type>java.lang.Integer</type>
<description>Set the database size to
use when computing E-values.</description>
<takes>size</takes>
<optionalFlag>true</optionalFlag>
<optionalValue>false</optionalValue>
</argument>
```

The WSDL required to describe the complex type corresponding to the arguments is automatically generated from this XML file using our Grid Services Generator (see section 5.1.8). Perl scripts are also automatically generated that accept the configuration parameters as command line arguments, write them out to a specifically-formatted file, and then execute the Grid service client. GSBL provides a class for the Grid service client that will read in this file and initialize an Instance of the custom type.

Finally, once the argument type has been sent to the Grid service, it will need to be converted back into an argument string to be passed to a GRAM job (and ultimately to the original command line program). GSBL provides a class that accepts the argument type and, using the XML file described above, generates the corresponding argument string.

One might ask, why not simply convert the client command line arguments to a string, send that to the Grid service, and be done with it? By parsing the arguments, we allow clients and services to make choices based on the values of the arguments, which is required for properly configuring GRAM jobs and helpful for Grid-level parallelism.

5.1.6 File Transfers

Effective Grid computing requires easy, reliable, bidirectional transfer of files between Grid clients and Grid services. There are, however, two key problems that need to be solved. First, there is the question of how the files are to be transferred: Globus provides a number of different mechanisms for transferring files. Second, file transfer is one of the areas in Grid computing in which the Globus asynchronous model is particularly important: subject to file sizes and network speeds, transferring a file could easily take more time than the timeout of the underlying remote procedure call libraries. Thus, we need to provide some mechanism by which this event-based process can be made to look procedural.

Our original GT3-based Grid system used the Global Access to Secondary Storage (GASS) protocol to send files between the client and the server, but we are now using GridFTP in conjunction with the Reliable File Transfer (RFT) service in our GT4-based system. The GridFTP protocol provides for secure, robust, fast and efficient transfer of data. The Globus Toolkit also provides the most commonly used implementation of that protocol, composed of a server implementation and a scriptable command line client. In our system, a GridFTP server runs on both the client and the service, thus enabling file transfer between them. RFT is a

WSRF-compliant web service that provides scheduler-like functionality for data movement. Provided with a list of source and destination URLs (e.g., `gsiftp://localhost/foo`), the service writes the file transfer description into a database and then moves the files on behalf of the user using GridFTP. Thus, in summary, GSBL negotiates with the RFT service to initiate file transfers, which in turn makes recourse to GridFTP for the actual data transfers.

In GSBL, the `ReliableFileTransferManager` class is used to initiate and monitor file transfers. It accepts a list of files, an upload or download operation, and a local and remote endpoint. Once the transfer is initialized, one calls `beginTransfer()` to start the transfer in a separate thread. This call should be immediately followed by a call to `waitComplete()`, which will block until the file transfer job object has issued its "transfer complete" notification. Using these two simple function calls, file transfer can be made to look procedural; at no point do developers have to concern themselves with event-handling. As a side note, the ability to transfer a batch of files in one method call marks an improvement over the GT3-based system.

This file transfer code is used in two phases of a job life cycle. The first phase is uploading job input files from the Grid client to the Grid server, and the second phase is uploading job output files from the server to the client.

5.1.7 Creating and Monitoring GRAM Jobs

Our Grid services need to submit GRAM jobs to remote computational resources on behalf of the client. These jobs may have to wait in a remote queuing system for some period of time, and even once execution begins, processing can take

a long time. As such, the Globus API for submitting GRAM jobs is an asynchronous, event-based construction.

The `GSBLJobManager` class for Grid services works much like the `ReliableFileTransferManager` class does for file transfer: it provides methods for starting a GRAM job and testing whether or not it completed successfully.

When a client calls `runService()`, passing along the complex argument type discussed in section 5.1.5, this Grid service method prepares to create the GRAM job and returns immediately to the client, which may then terminate. From this point on, the service is in charge of submitting and monitoring the job, and is also responsible for transferring output files back to the client host when the job is finished.

Because of this design, it is necessary for job monitoring to resume in the event that the Globus web services container is shut down and restarted, or otherwise interrupted. We have provided mechanisms that Grid services can use to recreate GRAM job objects and check the status of jobs that were previously submitted. These mechanisms make use of persistent state information about jobs that Globus keeps on disk, as well as a database that helps to determine which jobs have not yet finished. This monitoring process resumes automatically as each Grid service is initialized when the web services container is restarted.

5.1.8 Grid Services Generator

In order to further streamline the creation of Grid services using GSBL, we have written a program, the Grid Services Generator (GSG), that generates skeleton implementations and build environments for Grid services based on an extremely limited set of inputs (name of the service, package in which implementation classes

should reside, Extensible Markup Language [XML] description of the program arguments, and location in the web services container at which the service will be deployed). After running the program, the user will have client and service Java class templates that work with GSBL, a Web Services Description Language (WSDL) file for both the Factory service and the Instance service (both of which are basic Globus services), other required Globus configuration files, and build files so that the code can be easily compiled and deployed within a working directory. Because setting up this development environment for each new Grid service is otherwise an extraordinarily tedious and error-prone task, we have found that the GSG dramatically increases programmer productivity.

The Grid Services Generator was designed to ease the overall process of developing Grid services. In particular, it attempts to minimize the amount of code a programmer has to write by stamping out generic GSBL-based Java classes for a Grid client and service. Afterward, a programmer simply completes the non-templated portions of these classes to customize the behavior of their Grid service. In this way, it is possible to quickly develop a suite of application-based Grid services.

5.2 Data Management

We recently implemented a relatively sophisticated Grid-wide data caching scheme, which saves on disk space and bandwidth throughout the system. The basic idea is that an input data file cache is maintained on the Grid server and on each Grid resource. Before any new file transfers are initiated, (either from client to server or from server to resource), the system checks with a central directory to see if the file(s) to be transferred already exist at the destination. This may be the case if a particular

input file has been used for a job previously, which we find happens quite often with certain services. Furthermore, program executables are also cached as part of this scheme, which otherwise would be transferred repeatedly with each job submission.

We use the Globus Replica Location Service (RLS) to keep track of the locations of files. RLS maintains and provides access to mapping information from logical names for data items to target names. These target names may represent physical locations of data items, or an entry in the RLS may map to another level of logical naming for the data item. RLS is intended to be one of a set of services for providing data replication management in a Grid. By itself, it does not guarantee consistency among replicated data or guarantee the uniqueness of filenames registered in the directory, but is intended to be used by higher-level Grid services that provide these functionalities.

While we are not maintaining replicas *per se*, the basic functionality RLS provides is appropriate for our needs. A simple MD5 hash uniquely identifies a particular file, which RLS calls a logical file name (LFN). There is a one-to-many mapping from an LFN to a physical file name (PFN), since the same file may exist on multiple resources. In our system, PFNs are GridFTP URLs. Here is an example of an LFN $\rightarrow$ PFN mapping:

```
2349ab6c2d32527b0c9dbcfa26e8690c ->
gsiftp://128.8.141.68:2811/${GLOBUS_SCRATCH_DIR}/cache/234/2349ab6c2d
32527b0c9dbcfa26e8690c/garli.conf
```

The PFN specifies the remote resource and the path to the file on the remote file system. To handle the case of two files having identical contents but different

filenames, we keep only one physical copy and use symlinks to enumerate the different filenames. For example, here is a subdirectory of the file cache on the Grid server:

```
gt4admin@asparagine:/export/grid_files/cache/559/559b4f8378046a6f9ed2
04fc5260160d> ls -l

-rw-rw-r-- 1 gt4admin gt4admin 1147 Aug  1 19:22
559b4f8378046a6f9ed204fc5260160d

lrwxrwxrwx 1 gt4admin gt4admin   94 Aug  1 19:22 garli167.conf ->
/export/grid_files/cache/559/559b4f8378046a6f9ed204fc5260160d/559b4f8
378046a6f9ed204fc5260160d
```

Since these caches may grow to contain many thousands of files, we make them hierarchical (as can be seen from the URLs and paths in the previous examples), since UNIX file systems have a limit on how many inodes a directory may contain.

Of course, it is necessary to avoid completely filling up the physical volume the cache resides on. Our strategy is to periodically remove files from a cache that is becoming dangerously large. In order to achieve this, three things are needed. First, resources must report disk usage statistics about the physical volume where the cache resides. Second, the RLS database must be augmented with metadata for each file – namely, a timestamp marking when the file was last "requested" for use in a job, and the file size in bytes. Third, the cleanup process must combine this information to determine which files should be removed, *actually remove them somehow*, and also delete the corresponding RLS entries.

To report disk usage, each Globus scheduler provider is modified to query for two quantities: "disk used", and "disk available". This information propagates back to the central MDS database on the Grid server every n minutes (in our system, $n = 3$).

Once the information is aggregated, it can be displayed on the resources page (Figure 4.6) and used in the cache cleanup algorithm.

Augmenting the RLS database with metadata about the files it contains involves a one-time addition of two string "attributes" (in RLS terminology) associated with each PFN: `size` (in bytes), and `requested` (a UNIX timestamp). Code was added to GSBL to find the sizes of files and also to store dates associated with files as UNIX timestamps for the sake of easy comparison. However, there are at least two conditions that need to be met before files can be safely removed: 1) no file transfers (job submissions or result retrievals) should be in progress and 2) only files whose requested timestamp is *older* than the oldest currently running job are eligible for deletion. Thus, Grid activity is paused once nightly for a short while to perform cache cleanup and other maintenance.

To actually carry out the deletion, the following formula is currently used: if a cache is > 80% full AND < 20G remain, remove files until the cache is < 75% full OR > 25G remain. Note that it may not be possible to do this for the following reasons: 1) the cache on a remote resource is sharing the volume with files outside of our control or 2) some files may be in use by jobs and not eligible for deletion. The algorithm chooses to delete the oldest files first and then *only enough* files to bring disk usage under desired thresholds. However, it is still possible for caches to grow very full of files that have not been requested recently, thus putting a strain on backup systems because of the sheer number of files, even if they do not take up much space. Thus, it has been necessary to layer on an additional policy of deleting files that were last requested more than d days ago (where for example, $d = 90$). Eventually, it would

be nice to replace the current hand-picked, global values (i.e., 75%, 20G, and 90 days) with auto-tuned, resource-specific settings based on recent usage patterns.

For each resource cache in need of cleanup, a GRAM job is submitted to the proper scheduler on the remote resource (e.g., Condor, PBS, or BOINC). Submitting a GRAM job is necessary in order for the `GLOBUS_SCRATCH_DIR` variable to be properly interpreted – otherwise, we could simply use the RFT client program. These are very simple jobs (`/bin/true`) with file cleanup directives that delete the appropriate files, which correspond to LFN → PFN mappings in the RLS database. The RLS mappings are then deleted using the `globus-rls-cli` command line utility.

Since disk space on some resources is quite limited, in some cases we have been able to greatly increase the number of concurrently running jobs by eliminating redundant copies of files. Another benefit is that because there are no unnecessary file transfers, the job submission process is sped up and no bandwidth is wasted. Naturally, these gains are only made when input files are reused, but we find this happens rather frequently. Furthermore, we can use the knowledge of where the data in the Grid currently resides to make more intelligent scheduling decisions, since moving the computation to the data is generally more efficient than the converse (and a popular paradigm nowadays, as in the Google File System [20]).

5.3 Meta-scheduler

The scheduling component of any Grid system is likely to be one of the most important and logically complex, since to a large extent it determines the overall efficiency of the system. This component is called a meta-scheduler because it

decides on which local resource a job should run; when a job reaches the remote resource, it is usually scheduled again in that local environment. A scheduler must be informed about the present state of remote resources, and this is what the Monitoring and Discovery Service (MDS) does. MDS is a default Globus component that requires minimal configuration. For example, consider a Globus installation for which MDS has been configured to report about the status of a Condor pool. In that case the Condor scheduler provider will periodically parse the output of the Condor command `condor_status` to discover the total number of nodes in the pool, the number of nodes that are actually free (not bound to a machine owner or another computational process), and other information about the pool. This information is stored in XML format in the Globus container memory space and is valid only for a specified lifetime (in our system, 3 minutes).

The MDS database can be queried to retrieve various kinds of information, such as the status of the Condor pool in the example above. Also, the information in an MDS database can be periodically propagated to another MDS database running in a different Globus container process. Using this mechanism, it is possible to centrally aggregate all of the data about remote Grid resources, which is precisely what we do: we collect all the information about remote resources in the central Grid server MDS database, and query it to make scheduling decisions. Next we describe the scheduling algorithm in detail.

5.3.1 Scheduling Algorithm

First of all, the scheduler needs to know which resources are reporting. If a remote installation goes offline, any jobs sent there will fail, so we cannot safely

assume that our resources are always up and running. Instead, if we cease to receive MDS information from a certain resource, we mark the resource as "offline" and make sure no new jobs are scheduled there. Then the question becomes: of the resources that *are* reporting, which one do we send a particular job to? Well, the simple fact is that not all jobs will run on all resources, so the scheduler must match on various attributes to narrow down the possibilities. For example, the system keeps track of which CPU architecture and operating system combinations each application is compiled for (e.g., Intel/Mac OS X), and compares this list against the platforms each resource is advertising. Then, if the job has a minimum memory requirement, we filter out resources that do not meet the minimum memory criterion. Other resource requirements are also considered if necessary, such as whether or not the resource is MPI-capable, and whether or not it has additional software installed (e.g., R). One can imagine any number of additional filtering and ranking criteria, especially around complex issues like *policy* – determining which users may access a particular resource, which users have priority over other users, when a particular resource may be used and for how long, and so on. We have not yet placed any such policy restrictions on resource use at the Grid level, though it may be necessary to do so in the future. However, it is important to stress that when Grid jobs run at the local resource level, they are always subject to whatever local policies govern use of that resource. From the final set of eligible resources, the scheduler chooses the one with the lightest load and submits the job there.

5.3.2 Scheduler Implementation

As previously mentioned, the GT3-based Grid system made use of Condor-G, and specifically of its "matchmaking" feature. This simple load balancing scheduler was fed information by an older version of MDS. We abandoned Condor-G in favor of a custom scheduling framework, but retained some basic ideas from that system. As things stand, our meta-scheduler is basically comprised of a couple of Perl scripts and a GSBL class. The `get_resource_info` script periodically reaps the central MDS database and stores a list of available resources and their attributes in a simplified plaintext format. The `get_resource` script implements the scheduling algorithm and is called by the GSBL class to pick a resource when a job is being scheduled. Planned improvements to the scheduler are discussed in section 7.2.2.

Chapter 6: Combining Globus- and BOINC-based Systems

It is useful to define some BOINC-related terms that will be used throughout this chapter. In BOINC, a *work unit* defines a unit of computation to be executed. A *result unit* is an instance of a work unit: i.e., due to redundant computing, a BOINC server might create three result units for a given work unit. These three (not yet processed) result units are sent to clients, which process and return them. Once a quorum is reached (e.g., two matching result units have been received from clients), one result unit becomes the canonical result for the work unit. For simplicity, we may sometimes refer to "the result" of a work unit, in which the quorum/canonical designation process is subsumed.

6.1 Challenges in Combining Globus and BOINC

As described previously, Globus and BOINC differ significantly in their assumptions regarding the need they seek to fill and in the features that they provide. Any attempt to join these two systems must thus reconcile these differences. Here, we discuss some of the concrete challenges that must be overcome.

6.1.1 Job Submission

BOINC was designed to allow a single coordinated group to manage large-scale distributed computing projects. As such, BOINC has a number of assumptions about the way in which it will be used. In particular, BOINC has no concept of users, and thus no concept of remote users: there is simply a single local entity that provides work for the system. Globus, on the other hand, expressly allows

multiple distributed users to submit jobs. Thus, BOINC must somehow gain multi-user functionality.

6.1.2 Job Specification

GRAM, the protocol Globus uses to manage jobs, was designed assuming jobs would execute on conventional UNIX systems (i.e., systems with UNIX-like file systems where programs are executed by specifying a path, a command, and some arguments). BOINC, on the other hand, has no concept of paths and only a loose conception of a file system. Thus, a Globus job description document (JDD) will specify something like "`<executable>/usr/bin/foo</executable>`". In a Grid system where this request could be tasked to a desktop computer using the Windows operating system without `foo` installed, what is the meaning of "`/usr/bin/foo`"? This request needs to be mapped into the file-system-less BOINC universe.

6.1.3 Data and Executable Staging

Globus is able to stage both data and executable files from submitting systems to the host on which the job executes. In particular, this means that Globus compute resources are able to execute arbitrary, user-supplied codes. Thus, there needs to be a mechanism to handle the staging of data all the way down to the BOINC clients, and the issue of arbitrary code execution on a Desktop Grid needs to be addressed.

6.1.4 Reporting of Results

Globus can also stage result data and program output back to the submitting node from the compute node(s). Therefore, there needs to be some way to take files generated by BOINC clients and return them to the Globus submitting node. In the

next section, we provide details of our solution that integrates BOINC and Globus.

6.2 Globus-BOINC Adapter

6.2.1 Job Submission

By design, Globus provides mechanisms and procedures for integrating new types of resources: by placing an abstraction layer (GRAM) over its resources, it reduces the task of integrating a new resource type to that of writing a GRAM-compliant interface for that resource. Therefore, we have written a GRAM scheduler adapter (commonly known as a job manager) for BOINC. The job manager in this case is more complicated than in others, however, because the BOINC model is significantly different from more traditional queuing systems.

Globus provides a Perl base class from which job managers may derive, and by extending this base class, BOINC gains the ability to accept jobs from the outside world, thus acquiring multi-user functionality. Although this achieves many of the capabilities of a true multi-user system, it does not provide robust, production-grade authentication and authorization capabilities. Rather than graft authentication and authorization onto BOINC, we choose to leave these tasks to a Grid meta-scheduler such as Condor-G, or in the case of our current system, a meta-scheduler of our own design. In either case, the component is tightly integrated with the Globus Security Infrastructure. We believe that this represents a much preferred solution than forcing the concept of “BOINC local users” onto BOINC or making BOINC aware of Grid credentials. Note, however, that our design does provide, through Globus, multi-user authentication and authorization not heretofore available to BOINC.

The other three challenges require somewhat more complicated solutions, and

we discuss them next.

6.2.2 Job Specification

One of the primary tasks of a Globus job manager is to translate the job description documents (JDD) used by GRAM into a native format that the managed resource can understand. In many cases, this can be a straightforward mapping between corresponding fields. In our case, however, more work is required to generate a BOINC work unit from a JDD.

Globus job description documents contain a few fields of particular interest in this context. First, there is the executable field, which specifies the program to execute. This could be either a fully-qualified pathname or a simple executable name. As discussed earlier, however, BOINC does not have a UNIX-like execution environment, and it certainly does not have a shell capable of resolving a non-path-qualified name to a specific executable. Thus, the executable field needs to be mapped manually.

The closest BOINC concept to an executable file is an application. Essentially, each BOINC project is composed of one or more applications, which represent computations that clients may perform. Each application in turn is composed of one or more application versions, which are executables implementing the computation for specific client architectures. Thus, to establish a mapping between the JDD *executable* field and the BOINC *application name* field, we remove any path information from the executable field and look for a BOINC application matching the remainder. If a match is found, it is designated as the application to use. If a matching application cannot be found, the job submission is rejected and an error

is returned to Globus. Note that this requires applications to be pre-registered with the BOINC server; user-supplied code is not allowed. Although user-supplied code could be supported, our design specifically excludes this capability due to security concerns, as BOINC lacks mechanisms to protect clients from malicious programs.

Resource limits constitute another set of difficult mappings from Globus to BOINC. There are trivial mappings between certain resource limits, such as maximum memory required. However, BOINC and Globus measure computing requirements in fundamentally different ways. Globus measures them in minutes of CPU time, whereas BOINC measures them in number of floating-point operations required. Moreover, for Globus, CPU time limits are entirely optional, whereas in BOINC, operation counts rest at the core of the scheduling process. BOINC work units have an "estimated number of floating point operations" field, which is used to estimate how long the job will take to run on any given BOINC client. This allows BOINC to only send work to those clients able to complete it before the *delay bound*, or maximum permissible elapsed wall-clock time, expires. So, if estimated CPU time is not correctly set, BOINC scheduling will work sub-optimally. Further complicating the matter, the WS-GRAM job description schema has a field to set maximum permissible CPU time, but it does not have one for expected CPU time.

Our solution is two-fold. First, using standard Globus extension mechanisms, we introduce a new JDD parameter, `estCpuTime`, which is defined to be the estimated CPU time (in minutes) required by the job on a computer capable of one gigaflop. (Such a computer is identical to the reference computer used by BOINC when calculating expected real execution times from the estimated number of

required floating-point operations.) If this parameter is supplied, it is used to compute the number of floating point operations required by multiplying it by 60×10^9. (We chose to express `estCpuTime` in minutes instead of in operations so as to maintain consistency with the other Globus CPU time parameters). If a value for `estCpuTime` is not given, it defaults to one-half the maximum permissible CPU time.

The other JDD fields of particular interest are those relating to file staging, or the copying of files to and from the submitting node. Those fields need to be added as `<file info>` and `<file ref>` sections to BOINC work units so that file staging can be extended all the way through to the BOINC clients. We discuss file staging in more detail in section 6.2.3.

Once the various required parameters have been determined, a BOINC work unit based on those data may be written and submitted to the BOINC work database using the BOINC `create_work` utility, which completes the translation from a generic Globus job description to a resource native format.

6.2.3 Data and Executable Staging

File staging between the BOINC server and the submitting node is handled by standard Globus file transfer components. However, there is a need to extend file staging all the way down to the BOINC clients that actually execute the computations.

As expected, BOINC provides support for clients to exchange files with the server, so we simply need to ensure that the right files are sent to the right places at the right times. This is a two part problem: files need to be copied to the correct locations on the BOINC server, and BOINC clients need to be instructed to conduct

the correct sequence of uploads and downloads.

Globus jobs have a private working directory into which files are staged in from remote systems and out of which files are staged to remote systems. When a Globus job is sent to the BOINC server, files specified in the JDD as to-be-staged-in are automatically downloaded using Globus file transfer mechanisms. BOINC, on the other hand, has two file staging directories shared by all jobs and by all clients (one for staging files to clients – referred to as the "download" directory – and one for staging files from clients – referred to as the "upload" directory). Files staged to the BOINC server by Globus thus need to be copied from the Globus staging directory to the BOINC download directory, and they need to be renamed so as to ensure uniqueness, as BOINC requires all files to have unique names. Similarly, when BOINC clients upload their results to the upload directory on the BOINC server, they need to be uniquely named, but they need to be copied back to the Globus staging directory with the filenames that Globus expects them to have.

Our job description documents include a unique ID field that may be trivially used to generate unique filenames for job files. This is sufficient to handle the original name to unique name mapping required at job-submit time. The reverse mapping, required at job-completion time, is somewhat more difficult to handle, however; it requires additional techniques discussed more fully in section 6.2.4.

Once BOINC has been provided the job files, clients are instructed to transfer them by `<file info>` and `<file ref>` blocks in the work unit created for the job. Finally, BOINC assigns the client an executable appropriate for its architecture.

6.2.4 Reporting of Results

Without the ability to return results from the BOINC server to the Globus submitting node, our combined-model Grid system would be of little use. Returning results comprises two distinct tasks: returning any required output files to the submitting Globus node, and returning any standard output and standard error associated with the job to the submitting node.

First, Globus looks for the standard output of a job in a specific file, so by simply copying the standard output file returned from the BOINC client to that location, we can utilize the normal mechanisms provided by Globus to return standard output to the submitting node. Note that this design does not support real time streaming of standard output to the submitting node: standard output is buffered until the job terminates. Similarly, by copying output files from the BOINC upload directory to the Globus file staging directories, we can utilize the default Globus file staging mechanism. However, a problem now occurs: how do we know the location to which we need to copy our files? The file copying must be implemented by BOINC, not by the Globus job manager, as the Globus job manager should not (as a design decision) have to access BOINC internal data structures to locate these files. Moreover, BOINC will delete the work unit output files after it detects that the work unit has finished and that the associated cleanup code has executed. BOINC has no knowledge of Globus and thus no way of knowing where to copy the data.

Our solution is as follows. When a job is first submitted, the BOINC job manager writes out a Perl script containing the correct commands to copy files from

the BOINC upload directory to the Globus locations (even though these files do not yet exist); as part of Globus, the job manager has access to these locations. Cleanup code on the BOINC server calls this Perl script when a work unit completes. Files are thus placed in the correct locations at the correct times.

6.3 Other Custom Components

There are several other components that complete our Globus-BOINC interface. First among these is the BOINC Scheduler Event Generator (SEG). In the WS-GRAM framework, a SEG process runs for each local scheduler and propagates changes in job state to the Globus Job State Monitor (JSM). How it detects changes in job state is left up to the implementation. For example, the PBS SEG reaps log files periodically, whereas our custom BOINC SEG periodically queries the BOINC MySQL database. Previous scheduler adapter implementations included a `poll()` method to achieve the same thing by querying the scheduler directly, but for efficiency and other reasons, this method has been deprecated in favor of the SEG mechanism. As with our other supported resources (Condor and PBS), we also developed a BOINC scheduler provider that gives a rough idea of how many processors are currently available for each supported platform (Linux, Windows, and Mac OS).

BOINC has a component called the *validator*, which is responsible for comparing results, determining which to grant credit for, and determining how much credit to grant. We extend and customize this component, which is written in C++. BOINC has another component called the *assimilator*, which handles output from completed jobs. Our custom assimilator works closely with the BOINC scheduler

adapter to ensure that output is properly returned through the Grid system. This component is a mix of C++ and Perl.

6.4 Examples

Here, we present the flow of control for a job dispatched to a more typical Globus resource, such as a cluster managed by PBS, and for a job dispatched to a BOINC server as a Globus resource. As an example application, we use SSEARCH from the FASTA [42] suite of DNA and protein sequence analysis programs, which are important bioinformatics applications. SSEARCH uses the Smith-Waterman algorithm [55] to search a library of DNA or amino acid sequences (`lib.fa` in our examples) for sequences similar to a query sequence (`seq.fa` in our examples).

6.4.1 Portable Batch System

1. Globus user executes: `globusrun-ws -submit -Ft PBS -c /usr/bin/ssearch -O results.txt seq.fa lib.fa`
2. A Globus job description file is generated and passed to the Globus installation running on a PBS cluster node.
3. Globus copies `seq.fa` and `lib.fa` from the submitting host to a job-specific staging directory on the PBS cluster.
4. Submit method of the job manager executes: it writes a PBS job description file from the supplied JDD and submits it using `qsub`.
5. PBS eventually executes the job, and the job completes.
6. The PBS SEG recognizes that the job has completed and returns `results.txt` and any associated standard output to the submitting node. The job scratch directory is removed from the PBS cluster.

6.4.2 BOINC-based Desktop Grid

1. Globus user executes: `globusrun-ws -submit -Ft BOINC -c /usr/bin/ssearch -O results.txt seq.fa lib.fa`

2. A Globus job description file is generated and passed to the Globus installation running on the BOINC server.

3. Globus copies `seq.fa` and `lib.fa` from the submitting host to a job-specific staging directory on the BOINC server.

4. Submit method of the job manager executes:

    a. Strips "`/usr/bin/`" from "`/usr/bin/ssearch`" and checks to see if an "`ssearch`" application exists. Exits with an error condition if not.

    b. Determines `lib.fa` and `seq.fa` need to be staged to the BOINC client.

    c. Determines `results.txt` needs to be staged back from the BOINC client.

    d. Copies `lib.fa` and `seq.fa` to the BOINC download directory, giving them new names based on the unique ID present in the job description.

    e. Writes a work unit containing the arguments to `ssearch` and the file handling blocks for `lib.fa`, `seq.fa`, and `results.txt`; submits the work unit to BOINC, which generates result units for redundant computation.

    f. Writes a Perl script to be called on work unit completion that will copy the BOINC output files back to Globus-accessible directories.

5. Once per result unit: a BOINC client downloads the work unit, `lib.fa`, `seq.fa`, and an `ssearch` binary, caching the executable for future use.

6. Once per result unit: the BOINC client executes `ssearch` and returns `results.txt` to the server.

7. BOINC detects enough result units returned and designates one as canonical. It locates the callback script written out by the job manager and executes it.

8. Files corresponding to `results.txt` and `stdout` in the BOINC server upload directory are copied back to the locations and names expected by Globus.

9. BOINC deletes its copies of the result files associated with the work unit.

10. The BOINC SEG recognizes that the job has completed and returns `results.txt` and `stdout` to the submitting node. The job scratch directory is removed from the BOINC server.

6.5.1 BOINC Applications

Porting applications to run on BOINC can be non-trivial because BOINC expects applications that run in its framework to call its own API (Figure 6.1). The BOINC API handles tasks such as notifying BOINC when an application starts and exits, mapping between application-expected filenames and BOINC-required unique filenames, and checkpointing program state. The programs run on the Grid were not originally written with BOINC in mind; most are legacy applications written by a third party. Thus, porting an application to BOINC could require making extensive changes to its source code, which can present a significant hindrance to deploying applications on the BOINC-based Desktop Grid. Therefore, over the years we have employed different techniques for porting legacy applications to BOINC.

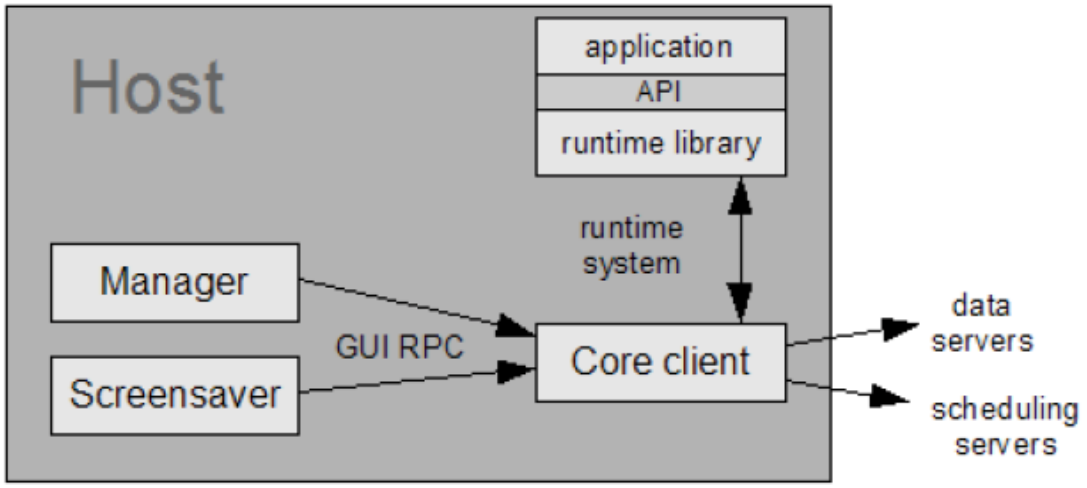


Figure 6.1. The BOINC client software includes a “core client” that executes applications and interacts with them through a runtime system.

First, we wrote compatibility libraries that allow programs written in C or C++ to run under BOINC; these libraries wrap C library functions so that the requisite calls to the BOINC API are made automatically. Under Windows, we used the Microsoft Detours package [27], and existing binaries could be used unmodified. Under UNIX-like systems (such as Linux and Mac OS X), only re-linking was required. For more information on these procedures, see our technical report [35].

The compatibility library is no longer used because BOINC eventually developed something called the wrapper application (to which we contributed code). The wrapper application can run unmodified legacy code as a child process, and it handles all communication with the BOINC client. It also supports checkpointing, graphics, and multiple child tasks. Porting applications with the wrapper is relatively straightforward.

However, it can be advantageous to rewrite source code to produce a native BOINC application, which is something Nathan Edwards has done with HMMPfam and Derrick Zwickl has done with GARLI, two of our primary Grid services. The source code is modified to make the required standard calls to the BOINC API, but also to write checkpoints and update the progress bar periodically. Checkpointing is nearly a requirement for jobs that run for any appreciable length of time, since interruptions happen frequently when running on a PC. Without checkpointing, much computation would be wasted. Updating the BOINC client manager progress bar (Figure 6.2) is also important because it is the only feedback mechanism a BOINC user has about how far along their jobs are. Our project volunteers prefer to run native BOINC applications that have these features.

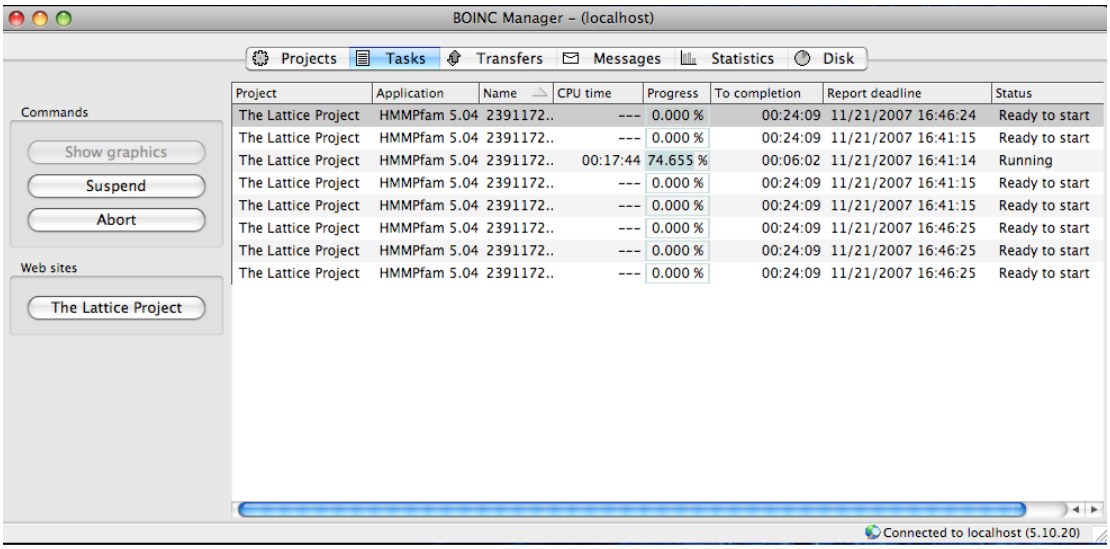


Figure 6.2. The BOINC client manager, showing the progress of one running task.

6.5.2 Homogeneous Redundancy

Most numerical applications produce different outcomes for a given work unit depending on the CPU architecture, operating system, compiler, and compiler flags. For some applications these discrepancies produce only small differences in the final output, and results can be validated using a “fuzzy comparison” function that allows for deviations of a few percent. Other applications are “divergent” in the sense that small numerical differences lead to unpredictably large differences in the final output. For such applications it may be difficult to distinguish between results that are correct but differ because of numerical discrepancies, and results that are erroneous. The “fuzzy comparison” approach does not work for such applications.

BOINC provides a feature called *homogeneous redundancy* (HR) to handle divergent applications. HR divides hosts into “numerical equivalence classes”: two hosts are in the same class if they return identical results for a certain application. If HR is activated, the BOINC scheduler will only send results for a given work unit to

hosts in the same equivalence class; this lets the BOINC validator use strict equality to compare redundant results.

Homogeneous redundancy in BOINC presents an interesting problem for applications that were not originally written with BOINC in mind. Oftentimes, random seeds, timestamps, or other program features cause normal program output to vary. Therefore, running a program the same way multiple times may yield output files that could be identical with respect to the analytical results of principal interest, but might differ in some uninteresting or insignificant way, which is problematic because the standard BOINC validator only checks for identical output. Thus, one could either write a custom validator for each application, or one could modify the application source code to remove timestamps or make the random seed the same for each result unit in a work unit. We have found that it is usually easiest to modify the application source code when it is available. However, validating results from applications programmed to run on a GPU (graphics processing unit) is more difficult, and is discussed in the following section.

6.5.3 GPU-enabled Applications

In the BOINC model, projects implement a validation procedure that ensures some level of agreement between sets of results returned from public computers. This is a way of dealing with "untrusted resources" – if results from disparate hosts agree, we assume there is only a minute probability they have been independently falsified, so we mark them valid. The current version of GARLI uses double-precision floating point values, and largely because of this, the log-likelihood values in the output are identical, even when the same job is run on very different operating systems and CPU

architectures. Thus, we have been able to use the default BOINC validation procedure that simply tests if the output files are identical. However, we are migrating to a version of GARLI that uses single-precision floating point values because it will be faster, both on modern GPUs and modern CPUs, and because it will use less memory. This makes validation more difficult, however, because it is more likely that there will be numerical discrepancies between hosts, especially when running on different GPUs. Since there are currently no numerical equivalence classes for GPUs, we will have to use a different technique to validate results in this case.

Chapter 7: Conclusion

7.1 Summary of Results

We have presented The Lattice Project, a comprehensive Grid system for scientific analysis that integrates a BOINC-based Desktop Grid with a feature-rich, Globus-based Service Grid. Our system makes a number of scientific applications available as Grid services through a UNIX-based command line interface, and provides tools for submitting and monitoring compute jobs. We have described, in detail, the features available to Grid users, the architecture and infrastructure of the system, the composition and makeup of our computational resources, the library we created for building Grid services, the functionality of our meta-scheduler and data caching scheme, and many components of the Globus-BOINC interface. The appendices provide a brief history of The Lattice Project, a description of some research projects that have used the Grid system, and some additional arguments for adopting Grid computing at the University of Maryland.

7.2 Future Work

Future development of The Lattice Project could take many directions, much like our development to date, which has been simultaneously focused on many different aspects of the project. Despite the fact that writing new Grid services, porting applications, supporting users of the Grid and maintaining the system takes time away from new development, there are major areas of the system that we would like to develop, given the opportunity to do so. First and foremost among these is the

user interface, which is discussed in section 7.2.1. Improving the meta-scheduler is also a high priority, which is discussed in section 7.2.2. We would also like to add other features, assimilate new resources, and continue to improve overall system performance.

7.2.1 User Interface Development

The current Grid interface is a mix of web tools and a command line interface. Researchers are given an account on a Linux machine supplied with programs for invoking our various Grid services. It is also on this machine that they are given a workspace in which to store results of computation. This is the primary interface for job submission. The current web tools allow one to more easily view the status of particular jobs and resources. These tools are also available in the command line interface.

The command line interface is perfectly usable but may sometimes take getting used to, especially if the Grid user does not have a strong UNIX background. Experienced UNIX users, on the other hand, will probably appreciate the power of the command line interface and may actually prefer it to a GUI-based interface. The problem we currently face with our command line interface is one of scalability – there are just a couple of machines that our Grid users share. It would be relatively simple to add disk space to the existing machines, or add more Grid Bricks. Along these lines, one idea that has been proposed is to extend the command line interface to WAM and Glue machines throughout campus, and allow users to authenticate themselves using their existing WAM/Glue accounts. This work was under development for a while, but has been put on hold due to other OIT priorities.

Another user interface option is a web portal for accessing the Grid. There are already some web tools in the Lattice intranet, but a portal would be more fully-featured: it would contain all the functionality necessary for a Grid user to manipulate their file space, organize analyses, and submit and monitor jobs without leaving their web browser. Our recently funded grant proposal will require the creation of a web-based interface for submitting certain phylogenetic applications, such as GARLI. To that end, an application-specific, user-friendly portal for job submission will be developed in the near future.

Semantic Workflow System

Among the barriers to the widespread use of Grid computing in life sciences is the difficulty of integrating Grid computing into everyday laboratory procedures. Scientific research often involves connecting multiple applications together to form a workflow. This process of constructing a workflow is complex. When combined with the difficulty of using Grid services, composing a meaningful workflow using Grid services can present a challenge to life scientists. The solution proposed by collaborators at Fujitsu Labs of America is a Semantic Web-enabled computing environment, called Bio-STEER [32, 33]. In Bio-STEER, bioinformatics Grid services are mapped to Semantic Web [10] services, described in OWL-S (Web Ontology Language-Service). An ontology in OWL (Web Ontology Language) to model bioinformatics applications is also defined. A graphical user interface helps to construct a scientific workflow by showing a list of services that are semantically sound; that is, the output of one service is semantically compatible with the input of the connecting service. Bio-STEER can help users take full advantage of Grid

services through a user-friendly graphical user interface, which allows them to easily construct needed workflows. After a workflow has been composed, the user simply presses play and watches the workflow execute. Our working prototype actually submitted jobs to the Grid as the various steps in a non-trivial workflow required. Bio-STEER was implemented as a Windows desktop application (Figure 7.1), but a similar workflow manager could be integrated into a web portal. With a powerful Grid system on the back-end, such a tool would be extremely valuable.

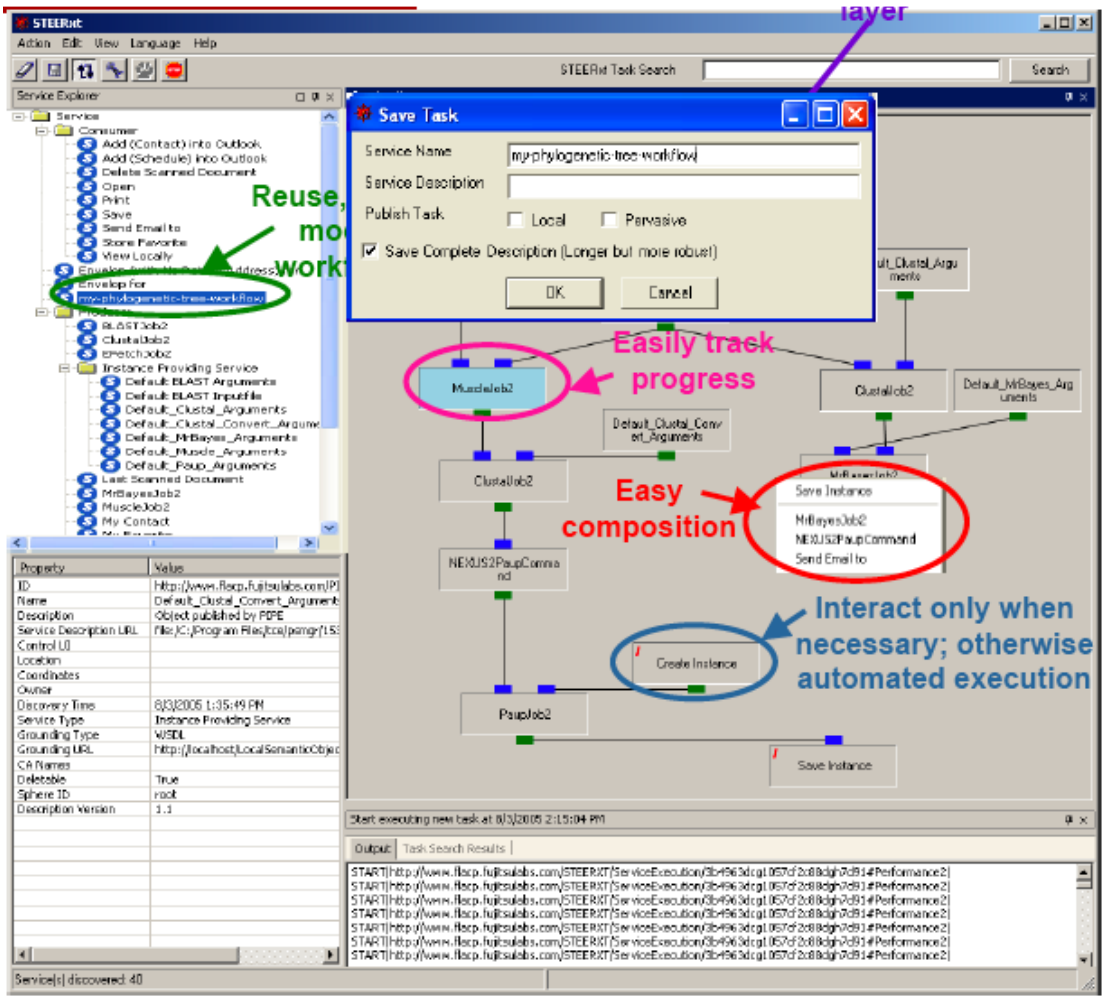


Figure 7.1. The Bio-STEER workflow composition tool.

7.2.2 Meta-Scheduler Development

We have recently added the capability to adaptively schedule jobs to our BOINC framework, a procedure which takes into consideration the composition of our BOINC client base, thus enabling each platform represented in the BOINC pool to be treated as a separate resource. There are several other features that would improve the meta-scheduler:

1. It would be helpful to rank resources based on overall performance (perhaps a combination of CPU speed and other recent performance metrics) so that jobs are sent to the fastest resources first.

2. The scheduler should divide work into batches on behalf of the user.

3. The scheduler should be able to reschedule jobs automatically in case of failures, or be able to reassign jobs if a faster resource becomes available.

4. Having the ability to break up long-running jobs into shorter, fixed-length pieces would benefit the scheduling of jobs to BOINC, where it is important to provide accurate runtime estimates in advance.

Appendices

Appendix A: A Brief History of The Lattice Project

In 2003, Michael Cummings and Daniel Myers built a Grid system "using commodity tools" [37] to complete a large-scale analysis, so our laboratory already had some experience in this area. We conferred with some members of UMIACS at this point about building a new Grid system, both in terms of the technology that should be employed and the infrastructure it would utilize. Some hardware purchases were made (and are still in use today) and research commenced, primarily into the workings of the Globus Toolkit. The Globus Toolkit is premier software employed by many major Grids in the world today, so our choice to use it several years ago turned out to be a good one. From the outset, we also knew that we wanted to include public computing in our project, so we became familiar with BOINC. We developed GSBL, the GSG, and prototyped a basic system that included the ability to submit jobs to the UMIACS Condor pool and a pool of BOINC clients. We also built up a corpus of Grid services that were popular in bioinformatics and molecular evolution, and for which we anticipated some demand.

Eventually we had built a production Grid system with GT3, which used Condor-G as the meta-scheduler. We also opened up the Lattice BOINC Project to the public during this “alpha” phase. The three Grid services that were being used most heavily at the time were CNS, IM, and MDIV, and all three of these applications were ported to BOINC. This early system completed more than 120 CPU years of computation over a period of several months. However, we knew that GT4 was slated

for release, so we began to rebuild the system around GT4. We replaced Condor-G with a custom scheduler and made other significant changes to the system, but the general architecture and many existing features remained the same.

Our transition to GT4 was mostly complete by the start of 2006. Since then we have been working to add some of the features already mentioned, but also to add new resources at UMD and elsewhere. We currently have four Condor pools and three dedicated clusters integrated as Lattice-addressable resources. Beyond simply trying to increase capacity, we have also focused on community-building and heightening awareness of The Lattice Project in an effort to make the Grid system multi-institutional. We have been assisted in this effort by Suresh Balakrishnan, Deputy CIO and Assistant Vice Chancellor of the University System of Maryland (USM). This led to fruitful interactions, in particular, with Coppin State University (CSU) and Bowie State University (BSU), who are now partners with us on a three-year grant proposal that leverages the Grid system to accelerate phylogenetic research. As a result of this collaboration, we have integrated computational resources at both institutions into The Lattice Project.

On our own campus, we organized a Grid Steering Committee that met regularly for a few years. This committee served two primary functions: 1) to foster discussion about Grid computing among interested parties on campus, and 2) to promote Grid computing as a priority within the Office of Information Technology (OIT), hoping that OIT would be able to spearhead the Grid computing effort on campus. Tangibly speaking, our collaboration with OIT led to the addition of two Grid resources: the Gridiron Condor pool (now retired), and the Deepthought cluster

(which continues to grow). In addition, by collaborating with OIT and ECE/ISR, we were able bring the SEIL cluster into the fold, a resource which was previously severely underutilized. We have been happy to find a strong interest in Grid computing on campus, as well as willing volunteers and participants.

Several people have worked on developing The Lattice Project over the years, in one way or another, and they are acknowledged at the beginning of this thesis.

Appendix B: Research Projects Using the Grid

Over the past several years, we have invited faculty, postdocs, graduate students, and others at the University of Maryland to use the Grid system for their research projects. The Lattice Project has now performed in excess of 18,000 CPU years of computation! That is equivalent to keeping 18,000 processors continuously busy for an entire year, which we consider to be a tremendous achievement. This estimate is a rough one given that processor speeds vary and our accounting methods are not precise to the second, but it nevertheless represents a considerable amount of computation.

Each research project that we have chosen to support has helped test and expand the capabilities of the system, as whatever Grid services need to be created for that project may present unique challenges. For example, a particular application may not be easily ported to each of our supported platforms (Linux, Windows, Mac OS), and porting legacy applications to run on BOINC presents additional challenges (as discussed in section 6.5.1). Related to code portability, some applications may be scripts meant to run in an interpreted environment instead of compiled code. A good example is the *gsi* service, which runs inside the R environment. We thus made

special efforts to ensure R was pre-installed on a significant portion of our resources, since there is no easy way to push out the necessary R environment at runtime.

Working together with many different researchers, helping them organize and submit their jobs, and listening to their feedback about the system has continually helped us improve it and has shown us where more work is needed. Taken together as a whole, the body of projects we have supported is extremely diverse. What follows is a description of the projects associated with three of our most heavily used Grid services: GARLI, HMMPfam, and MARXAN.

Phylogenetic Analysis – GARLI

The Cummings Laboratory and others are using GARLI [60] to infer phylogenetic trees from nucleotide or amino acid data. Various nucleotide, codon and amino acid models are implemented for maximum likelihood (ML) estimates. Multiple searches for the ML tree as well as the calculation of bootstrap support values are parallelized at the level of individual heuristic searches – i.e., every computing node has to carry out at least one complete heuristic search. This parallelization is particularly useful for large quantities of relatively short calculations, as is typical for nucleotide model bootstrap analyses with large numbers of repetitions.

The LepTree project (http://www.leptree.net/) investigates evolutionary relationships within the insect order Lepidoptera (moths and butterflies), in particular of higher taxa, such as families, superfamilies and infra-orders. This molecular "backbone phylogeny" is based on the analysis of up to 26 protein coding nuclear genes (~19kb) for 123 taxa, but work on a matrix for 550 to 600 taxa is well

underway. The chief method of analysis used in this study is a nucleotide model ML search in GARLI. The most commonly applied model is the general time reversible model with a gamma distribution of rates and a proportion of invariant sites (GTR+G+I). The LepTree project relies heavily on the computational resources provided by The Lattice Project, as the sheer number of heuristic searches is not feasible to run on an individual desktop machine. The bulk of these heuristic searches consist of bootstrap replicates (up to 2,000 per analysis), but in addition, due to the heuristic nature of the search, multiple searches (up to 500) are required for confidence in having found the ML tree. For the LepTree project, many analyses of these types are carried out, e.g., for individual and combined genes, synonymous and non-synonymous data partitions, and with and without topological constraints for subsequent hypothesis testing [50].

Miriam Reyna-Fabián aims to solve the intra- and inter-genus relationships of more than 15 species of rotifers, currently assigned to the family *Brachionidae*. Species of this family are free-living organisms and they compose part of the zooplankton in freshwater and marine systems. Variation in morphological characters has traditionally been used to differentiate species. However, the taxonomic positions of 3 species – *Brachionus patulus*, *B. macracanthus* and *B. polyacanthus* – have been controversial. A study based on scanning electron microscopy of the trophy [54] proposed erecting these 3 species to a new genus: *Plationus*. The phylogenetic analyses of the family *Brachionidae* were carried out with GARLI using genes encoding cytochrome oxidase subunit 1 (Cox 1) and domains (D2-D3) of 28S rRNA. A total of 23 sequences, including 8 outgroups, were aligned. The phylogenies

derived from this study were used to evaluate the validity of the new genus *Plationus*. The analyses support the hypothesis that *Plationus patulus* and *P. macracanthus* compose a distinct clade from *Brachionus* and *Platyias* with high bootstrap values [51].

The Neel lab is studying phylogenetic relationships among North American members of the genus *Agalinis* Raf. These species represent a taxonomically challenging group and there have been extensive historical revisions at the species, section, and subsection levels of classification. The genus contains many rare species, including the federally listed endangered species *Agalinis acuta*. In addition to evaluating the degree to which historical classifications at the section and subsection levels are supported by molecular data sampled from 79 individuals representing 29 *Agalinis* species, the monophyly of 27 species was assessed by sampling multiple individuals representing different populations of those species. Twenty-one of these species are of conservation concern in at least some part of their range [44].

Silvana Marten-Rodriguez aims to understand the role of pollinators in the evolution of floral traits and breeding systems in the Antillean tribe *Gesnerieae* by combining phylogenetic approaches with ecological studies. The tribe *Gesnerieae* includes species specialized for hummingbird or bat pollination as well as some generalized species pollinated by bats, hummingbirds and insects. Preliminary phylogenies suggest various independent pollination system transitions in addition to the evolution of reproductive assurance mechanisms (e.g., autonomous self-pollination). Low frequencies of hummingbird visitation and high pollen

limitation in specialized hummingbird pollinated species might be responsible for these transitions [34].

Protein Sequence Comparison – HMMPfam

`hmmpfam` is a program in the HMMER (http://hmmer.janelia.org/) package. HMMER uses profile hidden Markov models (HMMs) to characterize regions of similar amino acid sequence in protein families, groups of proteins with similar function found in related organisms. The `hmmpfam` program searches the protein sequences of proteins with unknown function against a carefully curated set of HMM models, called Pfam, from well-understood protein families. Protein sequences are assigned to one or more protein families on the basis of a statistically significant match to a Pfam HMM.

HMMPfam and RMIDb

The Edwards lab provides the Rapid Microorganism Identification Database (RMIDb – http://www.RMIDb.org), a freely available web-resource and database for the identification of bacteria and viruses using mass spectrometry. The RMIDb searches protein sequences from all of the major protein sequence repositories, plus computational protein sequence predictions from sequenced bacterial genomes, for mass matches with experimental masses from mass spectra. Protein sequences are carefully categorized according to strain, species, and other taxonomic groupings, and according to protein function, cellular location, and biological process using the Pfam assignments computed by `hmmpfam` and their associated gene ontology (GO) classifications. The functional classification of protein sequences must be recomputed using `hmmpfam` because each of the sources of protein sequence uses different,

sometimes conflicting, criteria for Pfam assignment, or provides no assignment at all. Functional classification of protein sequences makes it possible to analyze only the most likely to be observed proteins for mass matches, which decreases search time and increases the statistical significance of species identifications.

HMMPfam for RMIDb on BOINC

The Edwards laboratory is using the HMMPfam service to compute Pfam assignments for all bacterial, plasmid, and virus protein sequences from Swiss-Prot, TrEMBL, GenBank, RefSeq, and TIGR's CMR, plus an inclusive set of all plausible Glimmer predictions from RefSeq bacterial genomes. These protein sequences, and their Pfam assignments, are used in RMIDb. The HMMPfam service is also being used as a model for data-intensive bioinformatics applications on the Grid, as the amount of input and output data associated with the program is considerable. Supporting this work was a significant part of the impetus for designing the data caching scheme we now use routinely for all Grid services.

Conservation Reserve Network Design – MARXAN

MARXAN [6] is a decision support system for the design of conservation reserve networks. It is useful for selecting a reserve system from a large number of potential sites that satisfies various ecological, social and economic criteria. For example, it may be required that certain species or conservation features must be well protected within the reserve system, or the reserve system must not include more than a specified number of sites. The user translates their criteria into representation targets for the conservation features to be protected (e.g., number of populations of each species or percentage of each habitat type to be included in the reserve system),

and optionally a cost threshold or desired level of site compactness. MARXAN will produce reserve network solutions that meet these design constraints while simultaneously minimizing the cost of the design (e.g., number of sites required to meet all representation targets).

Two researchers are studying problems involved in the design of reserve networks for biological conservation using MARXAN, and collectively have consumed over 5000 CPU years using this Grid service. Maile Neel examines conservation decisions based on one target type (e.g., rare species) and the consequences at another level (e.g., genetic diversity), and this current work builds upon the theme of earlier work in this general area [38, 39]. Joanna Grand, a National Science Foundation Post Doctoral Fellow in Biological Informatics, studies the consequences of biased and incomplete data in the design of conservation reserve networks [21].

Biased Data and the Selection of Conservation Reserve Networks

Joanna Grand, Maile Neel, Michael Cummings (University of Maryland), Taylor Ricketts (World Wildlife Fund), and Tony Rebelo (South African National Biodiversity Institute) are collaborating on a project that uses MARXAN to quantify the impacts of basing the selection of conservation reserve networks on incomplete and biased species distribution data. Most species distribution data are biased in some way (e.g., higher sampling intensity closer to roads or within current reserves); however, they are commonly used to select sites for inclusion in reserve networks because they are considered to be the best data available. The ability of reserve

networks to adequately protect biodiversity when sites are selected based on incomplete and biased data is poorly understood.

The first set of analyses compared the quality of MARXAN reserve network solutions generated from both biased and complete species data. The data from a virtually exhaustive survey of the Proteaceae family of flowering plants in the Cape Floristic Region of South Africa was used as a baseline for "complete" data. To produce a sufficient range of solutions for comparison with the complete data solution, 1000 biased and random incomplete datasets were sampled from the full Proteaceae dataset. MARXAN was run 1000 times for each dataset. This study design required 1.2002×10^7 separate MARXAN runs which was possible to complete in only a few weeks by using the Grid system.

Current investigations are focused on how well reserve networks protect species when their design is based on species distribution data which is incomplete and biased, versus coarser environmental data which is easier to acquire and unaffected by the issue of sampling bias. MARXAN solutions generated with complete, biased, and random species data will be compared to those generated with environmental data (vegetation classes), and combinations of both data types. This analysis will require over 7.6×10^7 separate MARXAN runs and will again rely on the Grid system to make this enormous amount of processing feasible.

Older Research Projects

There were several research projects that ran on our GT3-based Grid system.

The Fushman laboratory ran thousands of protein-protein docking simulations using the CNS Grid service. When driven by experimentally derived constraints,

these simulations help in modeling the structures of large multi-subunit proteins, and the interactions of such proteins with various ligands. An example is analysis of the structural determinants for recognition of a polyubiquitin chain [59]. The computation for this problem was primarily done using BOINC, and the accumulated processing time was approximately 12.4 CPU years.

Floyd Reed and Holly Mortensen from the Laboratory of Sarah Tishkoff have run many analyses using the MDIV and IM Grid services. These analyses are for studies of human population genetics that use DNA sequence polymorphisms to estimate the times of divergence and migration rates among ethnically diverse populations in Africa [58]. The computations were done using our globally-distributed BOINC resources, and the accumulated processing time was approximately 13.1 CPU years.

Our own lab has made extensive use of the *gsi* Grid service to complete a study demonstrating the application of the genealogical sorting index (*gsi*) statistic for distinguishing species. Using coalescent theory-based simulations [35] to model genetic samples drawn from diverging species, the Grid system was used to calculate the statistic and assess its behavior. In addition, the probabilities of observed values were estimated using permutation [16, 8]. The many millions of individual analyses required consumed over 94 CPU years.

Appendix C: A Pitch for Grid Computing at the University of Maryland

The Grid system is of immediate utility to a number of groups at the University of Maryland, the primary ones being OIT, UMIACS, and CLFS. We have had extensive interaction with these groups about Grid computing via The Lattice

Project. The majority of on-campus Grid resources reside within these organizations, as do most of the researchers who have used the Grid system. UMIACS has a long history of supporting research using HPC/HTC; OIT, a somewhat shorter one; and CLFS has shown both a need and an interest in this area, having recently purchased a new college computing cluster. As things stand, each of these groups has their own user base, their own local computing resources, their own policies, and their own infrastructure for support. We suggest that existing computational resources on campus could be used more efficiently as part of the Grid system.

It is probably the case that the majority of existing HPC/HTC users in UMIACS either run on private clusters or vie for use of the UMIACS Condor pool; people registering with OIT get funneled to Deepthought, a monolithic cluster that continues to increase in size; in CLFS, researchers may make use of their own resources or may utilize the new CLFS computing cluster, which is actually part of Deepthought due to OIT's attractive resource integration model (not explained here). Plainly stated, the amount of competition for these large shared resources leaves some people waiting to use them, and this trend will likely continue even as more hardware purchases are made. The Lattice Project can provide users with access to other resources outside their domain, thus helping to balance and distribute the load more efficiently. One easy way to make a difference would be to enable HTC users of shared resources to use the Grid system for their work. Their many smaller jobs could be distributed more evenly across the existing resource base and out onto the BOINC pool, thus keeping clusters free for more traditional HPC users and preventing any one resource from becoming overwhelmed.

In addition to more intelligently distributing the workload, using the Grid system fundamentally changes the way research is conducted in two ways: it increases the amount of resources available to any one user of the system, and it makes managing large amounts of work easier by performing many otherwise tedious functions on behalf of the user. Once an application is deployed on the Grid, a user simply uploads their data and submits jobs without worrying about the resource on which the job is actually running. Furthermore, having a large number of resources available causes the researcher to reconsider the scope and extent of their analyses and may enable entirely new kinds of analyses to be conceived of and executed.

As it stands right now, The Lattice Project is in a stable production state and we are comfortable with all of the technologies employed. As with any system, there remain improvements to be made, and we discuss some of these in Chapter 7. However, we are confident that The Lattice Project can be of immediate utility to a number of groups on campus.

Bibliography

1. Altschul, S., Gish, W., Miller, W., Myers, E. & Lipman, D. J. (1990). Basic local alignment search tool. *J. Mol. Biol.* 215:403-410.

2. Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* 25:3389-3402.

3. Anderson, D. P. (2003). Public Computing: Reconnecting People to Science. Conference on Shared Knowledge and the Web. Residencia de Estudiantes, Madrid, Spain. Nov. 17-19.

4. Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M. & Werthimer, D. (2002). SETI@home: An experiment in public-resource computing. *Commun. ACM* 45(11):56-61.

5. Balaton, Z. et al. (2008). EDGeS: The common boundary between Service and Desktop Grids. pp. 37-48. In *Grid Computing: Achievements and Prospects* (Gorlatch, S., Fragopoulou, P., & Priol, T., eds). Springer US.

6. Ball, I. R. & Possingham, H. P. (2000). Marine Reserve Design Using Spatially Explicit Annealing, a Manual. MARXAN (V1.8.2).

7. Bazinet, A. L., & Cummings, M. P. (2009). The Lattice Project: a Grid research and production environment combining multiple Grid computing models. pp. 2-13. In *Distributed & Grid Computing - Science Made Transparent for Everyone. Principles, Applications and Supporting Communities* (Weber, M. H. W., ed). Rechenkraft.net, Marburg. In press.

8. Bazinet, A. L. & Cummings, M. P. Genealogical sorting index: software and web site for quantifying the exclusivity of lineages. In preparation.

9. Bazinet, A. L., Myers, D. S., Fuetsch, J. & Cummings, M. P. (2007). Grid services base library: a high-level, procedural application programming interface for writing Globus-based Grid services. *Future Gener. Comp. Sy.* 23:517-522.

10. Berners-Lee, T., Hendler, J. & Lassila, O. (2001). The Semantic Web. *Sci. Am.* 279**:**34-43.

11. Beerli, P. & Felsenstein, J. (1999). Maximum likelihood estimation of migration rates and effective population numbers in two populations using a coalescent approach. *Genetics* 152:763-773.

12. Beerli, P. & Felsenstein, J. (2001). Maximum likelihood estimation of a migration matrix and effective populations sizes in *n* subpopulations by using a coalescent approach. *Proc. Natl. Acad. Sci. USA* 98:4563-4568.

13. Brunger, A. T., Adams, P. D., Clore, G. M., DeLano, W. L., Gros, P., Grosse-Kunstleve, R. W., Jiang, J.-S., Kuszewski, J., Nilges, M., Pannu, N. S., Read, R. J., Rice, L. M., Simonson, T. & Warren, G. L. (1998). Crystallography & NMR system: a new software suite for macromolecular structure determination. *Acta Cryst.* D54:905-921.

14. Cummings, M. P., Handley, S. A., Myers, D. S., Reed, D. L., Rokas, A. & Winka, K. (2003). Comparing bootstrap and posterior probability values in the four taxon case. *Syst. Biol.* 52:477-487.

15. Cummings, M. P. & Huskamp, J. C. (2005). Grid computing. *EDUCAUSE Review* 40:116-117.

16. Cummings, M. P., M. C. Neel & K. L. Shaw. (2008). A genealogical approach to quantifying lineage divergence. *Evolution* 62:2411-2422.

17. Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.* 32:1792-1797.

18. Foster, I. & Kesselman, C. (1999). Globus: a toolkit-based Grid architecture. In *The Grid: Blueprint for a New Computing Infrastructure* (Foster, I. & Kesselman, C., eds). pp. 259-278. Morgan-Kaufmann, Los Altos, CA.

19. Frey, J., Tannenbaum, T., Foster, I., Livny, M. & Tuecke, S. (2002). Condor-G: a computation management agent for multi-institutional Grids. *J. Cluster Comput.* 5:237-246.

20. Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File System. *19th ACM Symposium on Operating Systems Principles*, Lake George, NY.

21. Grand, J., Cummings, M. P., Rebelo, T., Ricketts, T. H. & Neel, M. C. (2007). Biased data reduce efficiency and effectiveness of conservation reserve networks. *Ecol. Lett.* 10(5):364-374.

22. Guindon, S. & Gascuel, O. (2003). A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst. Biol.* 52:696-704.

23. Hashmi, N., S. Lee, & M. P. Cummings. (2004). Abstracting workflows: unifying bioinformatics task conceptualization and specification through Semantic Web services. W3C Workshop on Semantic Web for Life Sciences. Cambridge, Massachusetts USA.

24. Hey, J. & Nielsen, R. (2004). Multilocus methods for estimating population sizes, migration rates and divergence time, with applications to the divergence of *Drosophila pseudoobscura* and *D. persimilis*. *Genetics* 167:747-760.

25. Hudson, R. R. (2000). A new statistic for detecting genetic differentiation. *Genetics* 155:2011-2014.

26. Hudson, R. R. (2002). Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics* 18:337-338.

27. Hunt, G. & Brubacher, D. (1999). Detours: binary interception of Win32 functions. In *Proceedings of the 3rd USENIX Windows NT Symposium*. pp. 135-143. Seattle, WA. USENIX.

28. Kuhner, M. K., Yamato, J. & Felsenstein J. (1995). Estimating effective population size and mutation rate from sequence data using Metropolis-Hastings sampling. *Genetics* 140:1421-1430.

29. Kuhner, M. K., Yamato, J. & Felsenstein, J. (1998). Maximum likelihood estimates of population growth rates based on the coalescent. *Genetics* 149:429-434.

30. Kuhner, M. K., Yamato, J. & Felsenstein, J. (2000). Maximum likelihood estimation of recombination rates from population data. *Genetics* 156:1393-1401.

31. Laure, E. et al. (2006). Programming the Grid with gLite. *Comput. Methods Sci. Tech.* 12(1).

32. Lee, S., Hashmi, N., Hendler, J. & Parsia, B. (2004). Bio-STEER: an application of Task Computing – the Semantic Web Meets Grid Computing. Technical Report FLA-PCR-TM-3, Pervasive Computing Research, Fujitsu Laboratories of America, Inc.

33. Lee, S., Wang, D., Hashmi, N. & Cummings, M. P. Bio-STEER: a Semantic Web workflow tool for Grid computing in the life sciences. *Future Gener. Comp. Sy.* 23:497-509.

34. Marten-Rodriguez, S., Fenster, C. B., & Zimmer, L. A. Evolution of pollination and breeding systems in Antillean *Gesneriaceae*. To appear.

35. Myers, D. S. & Bazinet, A. L. (2004). Intercepting arbitrary functions on Windows, UNIX, and Macintosh OS X platforms. Technical Report CS-TR-4585, UMIACS-TR-2004-28, Center for Bioinformatics and Computational Biology, Institute for Advanced Computer Studies, University of Maryland.

36. Myers, D. S., Bazinet, A. L. & Cummings, M. P. (2008). Expanding the reach of Grid computing: combining Globus- and BOINC-based systems. pp. 71-85. In *Grids for Bioinformatics and Computational Biology, Wiley Book Series on Parallel and Distributed Computing* (Talbi, E.-G. & Zomaya, A., eds). John Wiley & Sons, New York.

37. Myers, D. S. & Cummings, M. P. (2003). Necessity is the mother of invention: a simple Grid computing system using commodity tools. *J. Parallel Distrib. Comput.* 63:578-589.

38. Neel, M. C. & Cummings, M. P. (2003). Effectiveness of conservation targets in capturing genetic diversity. *Conserv. Biol.* 17:219-229.

39. Neel, M. C. & Cummings, M. P. (2003). Genetic consequences of ecological reserve design guidelines: an empirical investigation. *Conserv. Genet.* 4:427-439.

40. Németh, Z. & Sunderam, V. (2003). Characterizing Grids: attributes, definitions, and formalisms. *J. Grid Comput.* 1:9-25.

41. Nielsen, R. & Wakeley, J. (2001). Distinguishing migration from isolation: a Markov chain Monte Carlo approach. *Genetics* 158:885-896.

42. Pearson, W. R. (2000). Flexible sequence similarity searching with the FASTA3 program package. *Methods Mol. Biol.* 132:185–219.

43. Pearson, W. R. & Lipman, D. J. (1988). Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA* 85:2444-2448.

44. Pettengill, J. and Neel, M. (2008). Phylogenetic patterns and conservation among North American members of the genus Agalinis (Orobanchaceae). *BMC Evol. Biol.* 8:264.

45. Posada, D. & Crandall, K. A. (1998). Modeltest: testing the model of DNA substitution. *Bioinformatics* 14:817-818.

46. Possingham, H. P., Ball, I. R. & Andelman, S. (2000). Mathematical methods for identifying representative reserve networks. In *Quantitative Methods for Conservation Biology* (Ferson, S. & Burgman, M., eds). Pp. 291-305. Springer-Verlag, New York.

47. Pritchard, J. D., Stephens, M. & Donnelly, P. (2000). Inference of population structure using multilocus genotype data. *Genetics* 155:945-959.

48. R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

49. Rambaut, A. & Grassly, N. C. (1997). Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Comput. Appl. Biosci.* 13:235-238.

50. Regier, J. C., Zwick, A., Cummings, M. P., Kawahara, A. Y., Cho, S., Weller, S. J., Roe, A. D., Baixeras-Almela, J., Brown, J. W., Parr, C. S., Davis, D. R., Epstein, M. E., Hallwachs, W., Hausmann, A., Janzen, D. H., Kitching, I. J., Solis, M. A., Yen, S.-H., Bazinet, A., Mitter, C. Toward reconstructing the evolution of advanced moths and butterflies (Lepidoptera: Ditrysia): an initial molecular study. *BMC Evol. Biol.* In press.

51. Reyna-Fabián, M., Laclette, J. P., Cummings, M. P., Sarma, S.S.S., & García-Varela, M. Molecular phylogeny of some species of the genus *Brachionus* and the systematic position of *Plationus* based on nuclear and mitochondrial gene sequences. To appear.

52. Rivas, E. & Eddy, S. R. (1999). A dynamic programming algorithm for RNA structure prediction including pseudoknots. *J. Mol. Biol.* 285:2053-2068.

53. Ronquist, F. & Huelsenbeck, J. P. (2003). MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics* 19:1572-1574.

54. Segers, H., Murugan, G., & Dumont, H. J. (1993). On the taxonomy of the Brachionidae: description of *Plationus* n. gen. (Rotifera, Monogononta). *Hydrobiologia* 268:1-8.

55. Smith, T. F. & Waterman, M. S. (1981). Identification of common molecular subsequences. *J. Mol. Biol.* 147:195-197.

56. Swofford, D. L. PAUP*: Phylogenetic analysis using parsimony (*and other methods), version 4. Sinauer Associates. Sunderland, Massachusetts, USA.

57. Thompson, J. D., Higgins, D. G., & Gibson, T. J. (1994). Clustal W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.* 22:4673-4680.

58. Tishkoff, S. A., Gonder, M. K., Brenna M. Henn, B. M., Mortensen, H., Fernandopulle, N., Gignoux, C., Lema, G., Nyambo, T. B., Underhill, P. A., Ramakrishnan, U., Reed, F. A. & Mountain, J. L. (2007). History of click-speaking populations of Africa inferred from mtDNA and Y chromosome genetic variation. *Mol. Biol. Evol.* 24(10):2180-2195.

59. Varadan, R., Assfalg, M., Raasi, S., Pickart, C. & Fushman, D. (2005). Structural determinants for selective recognition of a Lys48-linked polyubiquitin chain by a UBA domain. *Mol. Cell* 18:687-698.

60. Zwickl, D. (2006). Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion. Ph.D. thesis, University of Texas at Austin.