The Lattice Project: A Grid Research and Production Environment Combining Multiple Grid Computing Models

Adam L. Bazinet and Michael P. Cummings

pknut777@umiacs.umd.edu and mike@umiacs.umd.edu

Center for Bioinformatics and Computational Biology, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA

Author for Correspondence:
Michael P. Cummings
Center for Bioinformatics and Computational Biology
Biomolecular Sciences Building
University of Maryland
College Park, MD 20742
USA

mike@umiacs.umd.edu

Overview

Grid computing is a relatively recent formulation of distributed computing, and although there are more formal definitions[1], 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.[2]

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. The Lattice Project is primarily composed of computing resources at the University of Maryland campus in College Park, though we are moving forward to include both other institutions as well as public computing resources. Thus, our Grid encompasses both dedicated resources, such as clusters, and non-dedicated resources that are volunteered, such as administrative desktops or possibly users "@home" (at home), as with Berkeley Open Infrastructure for Network Computing (BOINC; http://boinc.berkeley.edu/) projects, which are derived from the SETI@home project[3]. 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 applications to date have been concerned with life sciences, though nothing about our architecture precludes other kinds of applications from running on the Grid.

There are some important characteristics that make The Lattice Project unique. Whereas many other projects in this book concern themselves with one particular problem, biological or otherwise, and are principally executed using BOINC, we set out to create a generalized Grid system using Globus[4], BOINC, and Condor (http://www.cs.wisc.edu/condor/) that would be capable of running many different applications simultaneously. One might think of this as so many "projects" in the BOINC sense, but without the overhead associated with any one project. Indeed, most of the applications that we run were never written with the idea of Grid computing in mind. Since this is a fully-featured Grid system, we have also spent time building user interfaces, integrating many different resources types (of which BOINC is one), and diligently working with others on and off campus to grow and improve the system.

Motivation and Philosophy

As the size and complexity of scientific data has increased, so have the sophistication and computational complexity of data analysis increased. 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 multilocus 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 computer 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[2]. However, there are several barriers to widespread use of Grid computing in many areas of the 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. [5-9]

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 very computation-intensive. Our need for computer resources for our work led to the development of a simple Grid computing system using commodity tools[10], which was used for a large-scale simulation study[11]. Our subsequent work has made use of the Globus Toolkit[4] and the BOINC (http://boinc.berkeley.edu/), and has focused on expanding the reach of Grid computing by creating a system that combines these two models.[9]

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

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 University of Maryland alone is estimated at 40,000, and most of these computers are idle the majority of the time. Of course, convincing various individuals and organizations within the institution to join the Grid and lowering the barriers to doing so are challenges. We feel that the large heterogeneity in the types of research problems, and the associated computational needs, is best met through 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 almost wholly independent processes, the results of which can be united to form a composite result (e.g., a parameter sweep). These are conventionally referred to as "embarrassingly parallel" problems, and are appropriately handled by desktop computing resources. In contrast to convention we see no need for embarrassment in completing these problems using a Grid, and we prefer to view these problems as pleasingly parallel.

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 we have also worked to insure 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 treated.

User interface: With regard to using the Grid we approach it from the perspective of users 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 systems. Most of the analytical applications that scientific researchers are familiar with employ some sort of command line interface, and we have attempted to provide a similar interface to our Grid services. (Note: a scientific or other application enabled to run on the Grid is a Grid service.) Thus, invoking a particular Grid service with a particular string of arguments might mimic exactly the standard use of the application, except that upon hitting return, the Grid takes the program executable, input files, job description and so forth, and sends it off to some remote resource. The person submitting the job does not have to worry about where the job is actually running.

General Types of Grid Computing Systems

At present, Grid computing systems can be broadly classified into two types. The first type might be considered the "classical" computational Grid system used by the computer science research community. These heavyweight systems 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 general class 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[3], phylogenetics[10,11], structural biochemistry (http://folding.stanford.edu/), and anti-HIV drug discovery (http://fightaidsathome.scripps.edu/).

In contrast to classical computer science 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[3] (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 of low load. 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 multiprocessors systems). In these cases, the fast interconnect of a symmetric multiprocessor or cluster is simply wasted. Hence many scientific computing problems would be well served by Desktop Grid systems if such systems 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 suburban homes. Ideally, we would like the best of both worlds: we would want to apply the features of the first model over the scope of the latter.

Globus

The Globus Toolkit[4] 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 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.

Globus operates on a push model: work is sent from some submitting node to some computational resource, which then accepts and processes the job, returning the results to the submitter. Moreover, these jobs can be arbitrary: Globus resources are able (although perhaps not always willing) to execute user-supplied code. Input and result files may be automatically transferred from the submitting node to the computing resource.

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 (http://gdp.globus.org/gt3-tutorial/), which formed the backbone of our Grid system. Development continued until we had a fully functional production-level Grid system built around Globus Toolkit 3.

After successful production use of this system, we focused our efforts on upgrading our infrastructure to use Globus Toolkit 4 (http://gdp.globus.org/gt4-tutorial/), which was released in early 2005.

BOINC

The Berkeley Open Infrastructure for Network Computing (BOINC; http://boinc.berkeley.edu/) is the direct descendant of the SETI@home project[2]. 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 made famous. 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., desktop personal computers) contact a server that acts as a central repository of work to retrieve jobs to execute. 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.[12] 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 which 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.

Condor

The Condor project from the University of Wisconsin (http://www.cs.wisc.edu/condor/) 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 itself 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 one administrative domain. The Globus Toolkit includes a Condor job manager thus allowing a job submitted via the Globus Resource Allocation Manager (GRAM; a suite of tools to submit, monitor, and cancel jobs on Grid computing resources) to run on a Condor pool somewhere. 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 tie the pools into the Grid using Globus mechanisms. As a side note, our Globus Toolkit 3-based production Grid system made use of Condor-G[13] as the Grid meta-scheduler or "master job queue", although we eliminated the need for this component in the Globus Toolkit 4 upgrade. In our experience the queuing systems on remote resources were sufficient to buffer jobs, and the scheduling functionality provided by the Condor matchmaking feature can be easily re-implemented and made arbitrarily complex, which is precisely what we have done with our own Grid-level scheduler. However, Condor software continues to be an integral part of our Grid system.

Architecture

Our Grid architecture has not changed much from our initial design (Figure 1). The Grid scheduler component used to be Condor-G[13]; now the scheduling logic is more tightly coupled with our Grid services layer. Of course, the figure is a deliberate simplification, as each numbered area represents multiple software components. Furthermore, most parts of the system could be replicated. For instance, there could be several interfaces, there could be distributed Grid servers, and multiple cluster head nodes or BOINC servers. Such is the distributed potential of Grid computing.

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. We wanted BOINC to function as just another Globus-addressable GRAM resource, and thus we created a Globus job manager for BOINC, henceforth known as the BOINC job manager, which is significantly more complex than some of the other standard job managers that come with Globus. However, it solves the same problems, namely, job specification (in this case mapping a Globus job description written in Resource Specification Language [RSL] into some kind of a BOINC workunit), data and executable staging (noting that each Grid service must be pre-registered as a BOINC project application), and reporting of results (results generated by BOINC must be returned to the Grid user). More details on how our BOINC job manager works is available elsewhere.[9]

Adapting Applications for Use With BOINC

Although we do not allow arbitrary code from Globus to run on the BOINC-managed Desktop Grid, it is desirable to minimize the effort required to port an application to BOINC in order to make BOINC a somewhat more general-purpose resource. BOINC has an application programming interface (API) that it expects applications to call; this API handles tasks such as mapping between application-expected filenames and BOINC-required unique filenames. 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.

In order to ease the task of porting a large number of existing bioinformatics applications, we have written 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 use the Microsoft Detours package[14], and existing binaries may be used unmodified. Under UNIX-like systems (such as Linux and Macintosh OS X), only re-linking is required. For more information on these procedures, please see our technical report.[15]

Homogeneous redundancy in BOINC presents an interesting problem for applications that were not written with BOINC in mind. Oftentimes random seeds, timestamps, or other program features cause raw program output to vary. Therefore, running the same program the same way multiple times may yield output files could be identical with respect to the analytical results of principal interest, but might differ in some uninteresting or insignificant way, which brings us to the problem: the standard BOINC validator simply checks for identical output. Thus, one could either have a custom validator for each application, or one could make modifications to the application source code to remove timestamps or fix the random seed to be the same for each result unit in a workunit. We have found that it is usually easier to modify the application source code when it is available.

The Lattice Project and BOINC

As mentioned previously, we first deployed BOINC clients on desktops at the University of Maryland. This enabled us to test our integration of Globus and BOINC in a somewhat controlled environment, because typically these BOINC clients ran as daemons and were always available to receive work. We did have a couple hundred clients deployed in this manner, and for a while they received work along with our public clients. Gradually, however, we phased them out in favor of Condor.

The Lattice Project BOINC alpha test was our foray into bona fide public computing. This prompted us to create and manage a public portal (the typical modification of the Hypertext Preprocessor [PHP] pages that come with BOINC), maintain message boards, provide feedback about the work we were doing and the state of the project at any moment, and so on. Moreover, we were able to test our applications on a wider range of platforms and under more realistic conditions using less reliable clients. Of course, there were problems that arose, and the community was usually very helpful in figuring them out. Then we would release a new application version of this program or that, and try again. The process was fun and sometimes nerve-wracking.

People donating their computer to a BOINC project expect to receive some work to do. With alpha or beta projects, there is a general understanding that the project is unstable and that work may not be available all of the time. Nonetheless, this problem turned into a particular source of frustration in our case because unlike some other projects that have seemingly unlimited quantity of workunits to send out, our activity was intermittent and the quantity of work we had often was not enough to keep all of our public clients busy. (The client pool swelled to well over 1000 users across 21 countries despite the fact that we did not advertise the project.) The explanation for the unpredictable levels of work is sort of implicit in the kind of system we have built: our researchers submit job batches of varying sizes at different times, and these jobs may be assigned to run on BOINC, or they may be assigned to other resources. In the future, we hope to have some larger projects that will be able to satisfy larger numbers of BOINC contributors, but look at the upside: a researcher who is accustomed to waiting in line for a cluster can have their work finish much faster using this Grid system which not only includes clusters but also a horde of BOINC clients at the ready. Thus, it really becomes an issue of interest and expectations. We should take appropriate measures to advertise to the BOINC community the various projects being run on The Lattice Project, be they large or small, and our client base should not expect the same endless stream of work from this project as they might from SETI@home. However, this could change depending on our project mix, the popularity of our Grid system in the future, and the number of clients we may have at any given time.

To date, we have not implemented BOINC checkpointing for any of our applications because this would require extensive knowledge of the application source code, which we typically do not have. For shorter running jobs, this is not usually a problem, but some of our jobs can run for several days depending on program parameters, hardware specifications, and so on. Thus we received several complaints from participants who were unable to complete workunits because they kept restarting from the beginning every time they were interrupted. An interim recommendation we made was to keep the application in memory when suspended, although this would sometimes require more RAM than a particular machine had. Thus, we had to increase memory requirements for certain applications. This remains an outstanding problem that we will need to address in the future.

Our experience with the BOINC alpha test was extremely positive and it is always encouraging to see the high level of interest and dedication from BOINC users.

The Challenges of Working with Globus

As might be expected in research-grade software, there are problems with the Globus Toolkit.

1. 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.

2. Globus uses an asynchronous, event-based model for programming Grid services.

Although a 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.

3. The Globus Toolkit is research software under continual development, and therefore it is always the possibility that the API presented to Grid services will change between versions. This is precisely what happened between Globus Toolkit 3 and Globus Toolkit 4. A perceived high probability of API change can make programmers hesitant about writing Grid services using the API.

4. 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 were 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, we knew we would be building a significant number of services, and so we needed to reduce the overhead associated with this process as much as possible.

Our Solution

To resolve the above problems, we have written the Grid Services Base Library (GSBL)[5], which provides a high-level, procedural API for writing Grid services. In The Lattice Project 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 over twenty production Grid services for life science applications, we have yet to encounter the need to circumvent GSBL to write custom Globus code.

In keeping with standard web services procedures, we have designed our Grid system with a generalized client-service architecture in mind. As mentioned previously, each Grid service represents a Grid-enabled application. A Grid client then 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 the client and the service, and submission and monitoring of GRAM jobs by the service. It should be noted that GSBL is a library for writing both Grid services and Grid clients that are inter-operable in the framework outlined.

Grid Services Generator

In order to further streamline the creation of Grid services using this library, we have written a program, the Grid Services Generator (GSG)[5], 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 Grid 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. Additional information about GSBL and GSG has been presented elsewhere.[5]

Software Availability: Recent versions of GSBL and the GSG are available for download from The Lattice Project web site, http://lattice.umiacs.umd.edu/. This software is free; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. Please credit the original authors and cite the paper by Bazinet et al.[4] where appropriate.

Grid-enabled Applications and Projects

We have built twenty-one Grid services implementing scientific computing applications using the GSBL library as part of a Grid system for comparative genomics research. Each of these programs is available to be run on our Grid system, which provides researchers access to more resources than they would otherwise have. Thus, large amounts of work can be done in a relatively short time. Our list of applications includes the following.

BLAST (Basic Local Alignment Search Tool): a sequence database search program.[16,17]

ClustalW: a multiple sequence alignment program.[18]

CNS (Crystallogrphy & NMR System): a program for molecular structure determination.[19]

GARLI (Genetic Algorithm for Rapid Likelihood Inference): a phylogenetic analysis program.[20]

gsi (Genealogical Sorting Index): a program for a statistical analysis evolutionary trees.[21,22]

IM (Isolation with Migration): a population genetics estimation program.[23]

LAMARC (Likelihood Analysis with Metropolis Algorithm using Random Coalescence): a population genetics estimation program.[24-26]

MARXAN: a program used in the design of reserves for biodiversity conservation.[27,28]

MDIV (Migration and Divergence), a population genetics estimation program.[29]

Migrate: a population genetics estimation program.[30,31]

Modeltest: a program for evaluating the fit of evolutionary models.[32]

MrBayes: a phylogenetic analysis program.[33]

ms: a population genetics simulation program.[34]

MUSCLE: a multiple sequence alignment program.[35]

PAUP*: Phylogenetic Analysis Using Parsimony (*and other methods), a phylogenetic analysis program.[36]

PHYML: a phylogenetic analysis program.[37]

Pknots: an RNA structure prediction program.[38]

Seq-Gen: a sequence simulation program.[39]

Snn: a population genetics estimation program.[40]

SSEARCH: a pairwise sequence alignment program.[41,42]

Structure: a population genetics inference program.[43]

The analyses represented by these applications and their associated Grid services are those in demand for computational biology projects in the Laboratory of Molecular Evolution, Center for Bioinformatics and Computational Biology, and for several other researchers at the University of Maryland. Many of these bioinformatics applications have not yet been addressed by other Grid systems. However, GSBL is generally applicable to the development of any computational Grid service under Globus.

Research Projects Using the Grid

A significant amount of scientific research computing has been completed using The Lattice Project Grid system by several groups at the University of Maryland, including the laboratories of David Fushman, Sarah Tishkoff, Maile Neel, and Michael Cummings.

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.[44] 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.[45] The computations were done using our globally-distributed BOINC resources[9], 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[34] 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.[21,22] The many millions of individual analyses required consumed over 94 CPU years.

Across these three studies our Globus Toolkit 3-based production Grid system performed approximately 120 CPU years of work during intermittent use over about a 7 month period. Such studies are only possible using large collections of resources such as a Grid computing system.

Our Globus Toolkit 4-based production Grid system has seen limited use thus far. Two researchers are studying problems involved in the design of reserve networks for biological conservation using MARXAN, and collectively have consumed almost a quarter CPU-century during this period. 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.[46,47] 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[48].

User Interfaces

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 submitting 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.

A nice enhancement to our interface would be to implement a web portal for job submission and file management, which would minimize the amount of Linux our researchers have to learn in order to use the Grid system. It might also ease the process of submitting and describing batches of jobs.

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[7,8]. In Bio-STEER, bioinformatics Grid services are mapped to Semantic Web[49] 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 (GUI), which allows them to easily construct workflows needed.

Developing The Lattice Project

Work on The Lattice Project began in 2003, where a large portion of time was dedicated to outlining the architecture of the system and deciding which existing software toolkits to use. Of course, we settled on using many features of the Globus Toolkit, but we also decided that Condor and BOINC would be particularly useful.

Much research and programming with the Globus Toolkit had to be done in order to determine how to build Grid services. In early phases of the project we were working with Globus Toolkit version 3. We developed GSBL, GSG, wrote Grid services for a variety of applications, developed a simple command-line user interface, and also deployed BOINC clients on machines at the University of Maryland alongside an existing Condor pool. As we scaled up, we refined our techniques for porting applications to BOINC on the various platforms we were targeting[15], made modifications to our BOINC job manager, and we became more familiar with BOINC server administration.

Eventually, we upgraded our BOINC server to major version 4, and with that we opened up the project to select members of the BOINC community for alpha testing. During this time we had a handful of researchers running projects on our Globus Toolkit 3-based production Grid system. Eventually the work ran out and the alpha test was temporarily retired. Our Globus Toolkit 3-based Grid system logged approximately 120 CPU years of computation over a period of several months.

As early versions of the Globus Toolkit version 4 were being released, we decided to put a great deal of time into upgrading the Grid system, which was more like a complete overhaul. This development work required most of a year to complete as we took the opportunity to change and improve things. For example, instead of using Condor-G as our Grid meta-scheduler, we decided to write our own simple scheduler using information from MDS4, the new XML-based Monitoring and Discovery Service from the Globus Toolkit.

Current Efforts

We have completed an upgrade of our production Grid system to use Globus Toolkit 4 and we are in the process of upgrading our BOINC server to the latest version. We are currently testing GSBL in this new Globus Toolkit 4 environment with a number of analyses, and we can expect increased usage as time goes on. The use of Globus Toolkit 4, which presents Grid services as web services, will facilitate our work in creating Semantic Web/Grid services and use of the Bio-STEER workflow composition system[7,8] and its logical extensions[6]. Finally, and as always, we are striving to continue to add computational resources in the University System of Maryland and elsewhere.

Our Grid system was designed for highly parallelized execution, like many other BOINC projects, so very often Grid users will submit large batches of jobs for processing. However, we do have plans to support running MPI[50] jobs on the Grid in the future.

Conclusion

The original aim of The Lattice Project was to develop a comprehensive Grid system to fulfill the ever-increasing computational needs of life scientists and other researchers. Our production Grid system, which has performed over 120 CPU years of computation since it came online, is based on a novel Grid architecture that encompasses almost any computational resource available, be it an institutional machine or a computer in the public domain. Our ability to be so inclusive is thanks in part to a novel combination of the Globus and BOINC toolkits. The Lattice project continues to develop as the primary Grid computing solution for the University System of Maryland and elsewhere.

Additional information, including documentation, is available on The Lattice Project web site, http://lattice.umiacs.umd.edu.

Acknowledgment

The following people have also made significant contributions to The Lattice Project: Daniel Myers, John Fuetsch, Stephen McLellan, Meldavid Manela, Jonathan Howard, Christopher Milliron, and Deji Akinyemi. We would also like to thank the University of Maryland Institute for Advanced Computer Studies (UMIACS) staff, who provided excellent system administration and support.

References

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

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

3. 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.

4. 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.

5. Bazinet, A. L., Myers, D. S., Fuetsch, J. & Cummings, M. P. Grid services base library: a high-level, procedural application programming interface for writing Globus-based Grid services. *Future Gener. Comp. Sy.* In press.

6. Hashmi, N., S. Lee, and 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.

7. 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.

8. 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.* In press.

9. Myers, D. S., Bazinet, A. L. & Cummings, M. P. Expanding the reach of Grid computing: combining Globus- and BOINC-based systems. In *Grids for Bioinformatics and Computational Biology* (Talbi, E.-G. & Zomaya, A., eds). Wiley & Sons. To appear.

10. 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.

11. 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.

12. 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.

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

14. 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.

15. 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.

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

17. 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.

18. 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.

19. 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.

20. 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.

21. Cummings, M. P., Neel, M. C. & Shaw, K. A genealogical approach to quantifying lineage divergence. In preparation.

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

23. 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.

24. 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.

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

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

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

28. 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.

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

30. 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.

31. 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.

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

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

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

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

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

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

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

39. 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.

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

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

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

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

44. 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.

45. 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. History of click-speaking populations of Africa inferred from mtDNA and Y chromosome genetic variation. Submitted.

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

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

48. Grand, J., Cummings, M. P., Rebelo, T., Ricketts, T. H. & Neel, M. C. Common data biases reduce efficiency and effectiveness of conservation reserve networks. In preparation.

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

50. Gropp, W., Lusk, E. & Skjellum, A. (1994). Using MPI, portable parallel programming with the Message-Passing Interface. MIT Press, Cambridge, MA.

Vitae

ADAM L. BAZINET received a B.S. degree in computer science from the Rochester Institute of Technology, Rochester, New York, and is a Research Assistant in the Center for Bioinformatics and Computational Biology, University of Maryland. He continues to do research in Grid computing, bioinformatics, and computational biology.

Michael P. Cummings received a B.Sc. degree in Botany from the University of California, Davis, and a Ph.D. degree in Biology from Harvard University. He received an Alfred P. Sloan Foundation Postdoctoral Fellowship in Molecular Studies of Evolution and has done postdoctoral research at the University of California, Berkeley, and the University of California, Riverside. He is an Associate Professor in the Center for Bioinformatics and Computational Biology, University of Maryland, with appointments in the Department of Biology and the Institute for Advanced Computer Studies. He is also Director of the Workshop on Molecular Evolution, Marine Biological Laboratory, Woods Hole, Massachusetts. His research interests are in the areas of molecular evolutionary genetics, and computer science related to bioinformatics, computational biology, and Grid computing.

Figure 1: Generalized architectural diagram of The Lattice Project from user interface to remote resource and everything in-between showing typical components and principal functional steps.