# 1 Expanding the Reach of Grid Computing: Combining Globus- and BOINC-Based Systems

DANIEL S. MYERS[†], ADAM L. BAZINET, and
MICHAEL P. CUMMINGS

Center for Bioinformatics and Computational Biology

Institute for Advanced Computer Studies

University of Maryland

College Park, MD 20742

USA

## 1.1 INTRODUCTION

Grid computing is a relatively recent formulation of distributed computing, and although there are more formal definitions [25], we use the following description [9]: 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.

[†]Present address: Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139-4307, USA

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. Such heavy-weight 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 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, the combined power of hundreds to many thousands of desktop systems united in a Desktop Grid represents a substantial computing resource. Desktop Grids excel at embarrassingly-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 [2], phylogenetics [8, 24], structural biochemistry [21], and anti-HIV drug discovery [38].

In contrast to classical computer science research Grid systems, light-weight 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 [2] (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 [22] 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.

Many computational biology problems are well-suited to processing by Desktop Grids for two main reasons. First, many computational biology problems require considerable CPU time to solve, 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 computational biology algorithms exhibit extremely coarse-grained parallelism, and many existing applications do not take advantage of parallel hardware. In these cases, the fast interconnect of a cluster or symmetric multiprocessor is simply wasted. Hence many computational biology 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. Here, we describe middleware that allows us to realize this goal. We present an interface between the Globus Toolkit [11] and the Berkeley Open Infrastructure for Network Computing (BOINC) [37]. First, however, we provide an overview of the two software toolkits.

### 1.1.1 Globus

The Globus Toolkit [11] is the paradigmatic example of a heavy-weight 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. Additionally, 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.

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 between the submitting node and the computing resources.

Finally, 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 completely compliant with the Web Services Resource Framework (WSRF), so its Grid services are, in fact, sanctioned 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.

### 1.1.2   BOINC

The Berkeley Open Infrastructure for Network Computing (BOINC) [37] is the direct descendant of the SETI@home project. 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, here, 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 [1]. 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, given that BOINC compute clients

are expected to be unreliable, 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.

Finally, it is useful to define some BOINC-related terms that we use 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 five result units for a given work unit. These five (not yet processed) result units are sent to clients, which process and return them. Once a quorum is reached (e.g., three 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.

## 1.2  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.

### 1.2.1  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 multiuser functionality.

### 1.2.2    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 universe of BOINC.

### 1.2.3    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.

### 1.2.4    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 a general overview of our approach to integrating BOINC and Globus.

## 1.3 MEETING THE CHALLENGES — GENERAL IMPLEMENTATION

### 1.3.1 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 interface (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 other 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 multiuser 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 [12], or in the case of our current system [3], a meta-scheduler of our own design. In either case, the component is tightly integrated with the Globus security infrastructure. It is our belief 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, multiuser authentication and authorization not heretofore available to BOINC.

The other three challenges require somewhat more complicated solutions, and we discuss them below.

### 1.3.2 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 an job description document.

Globus job description documents contain a few fields of particular interest in this context. First, we have 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, we need to map the executable field 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, we establish a mapping between the JDD *executable* field and the BOINC *application_name* field. To do so, we remove any path information from the *executable* field and look for a BOINC application matching the remainder. If we find a match, we designate it as the application to use. If a matching application cannot be found, we reject the job submission and return an error to Globus. Note that this requires applications to be pre-registered with the BOINC server; we do not allow user-supplied code. 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 $60 \times 10^9$. (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 the section 1.3.3.

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 command, which completes the translation from a generic Globus job description to a resource native format.

### 1.3.3   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 which 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 1.3.4.

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. Assigning the client an executable appropriate for its architecture is also handled by BOINC.

### 1.3.4   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, note that by default, BOINC does not trap

the standard output of the processes it executes. Thus, applications executing under BOINC need to send their standard output to the file *boinc_stdout*, which we arrange to have copied back to the BOINC server (our application compatibility library, discussed in section 1.5, automatically redirects standard output). The task then becomes one of copying these files to the correct locations.

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. We provide cleanup code for the BOINC server that calls this Perl script when a work unit completes. Files are thus placed in the correct locations at the correct times.

## 1.4  EXAMPLES

Here, we present the flow of control for a job dispatched to a more typical Globus resource, such as a cluster managed by the Portable Batch System (PBS) [39], and for a job dispatched to a BOINC server as a Globus resource. As an example

application, we use SSEARCH from William Pearson's FASTA [27] suite of DNA and protein sequence analysis programs, which are important bioinformatics applications. SSEARCH uses the Smith-Waterman algorithm [32] 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).

### 1.4.1    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 job manager 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.

### 1.4.2    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 that lib.fa and seq.fa need to be staged to the BOINC client.

   (c) Determines that 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. BOINC produces multiple result units for redundant computation.

   (f) Writes a Perl script to be called on work unit completion that will copy the BOINC files corresponding to results.txt and boinc_stdout back to Globus-accessible directories.

5. Once per result unit: a BOINC client downloads the 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 by the job manager and executes it.

8. Files corresponding to results.txt and boinc_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 job manager 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 BOINC server.

## 1.5    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 [17], and existing binaries may be used unmodified. Under UNIX-like systems (such as Linux and Macintosh OS X), only relinking is required. For more information on these procedures, please see our technical report [23].

## 1.6    GRID-ENABLED APPLICATIONS RUNNING ON BOINC

We have built more than 20 Grid services implementing bioinformatics and computational applications using the Grid Services Base Library (GSBL) library [4] as part of The Lattice Project Grid system [3], and of these 15 can run on BOINC. 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. The following is a list of applications that run on BOINC and some information about each one.

*ClustalW*   A program for multiple DNA or protein sequence alignment based on a progressive alignment strategy where more similar sequences are aligned first to produce groups of aligned sequences and then these groups are aligned together [34].

*CNS*   Crystallography and NMR System (CNS) is a program has been designed to provide a flexible multi-level hierarchical approach for the most commonly used algorithms in macromolecular structure determination [7].

*IM*   Isolation with Migration (IM) is a program for the fitting of an isolation model with migration to haplotype data drawn from two closely related species or populations [14].

*LAMARC*   Likelihood Analysis with Metropolis Algorithm using Random Coalescence (LAMARC) is a package of programs for computing population parameters, such as population size, population growth rate and migration rates by using likelihoods for samples of data (sequences, microsatellites, and electrophoretic polymorphisms) from populations [18, 19, 20]. LAMARC provides both maximum likelihood and Bayesian estimates, and uses a coalescent theory approach taking into account history of mutations and uncertainty of the genealogy.

*MDIV*   A program that simultaneously estimates divergence times and migration rates between two populations under the infinite sites model or under a finite sites model [26].

*MIGRATE-N*   MIGRATE estimates population parameters, effective population sizes and migration rates of $n$ populations, using genetic data [5, 6]. MIGRATE provides both maximum likelihood and Bayesian estimates, and uses a coalescent theory approach taking into account history of mutations and uncertainty of the genealogy.

*Modeltest*   A program that assists in evaluating the fit of a range of nucleotide substitution models to DNA sequence data through a hierarchical series of hypothesis tests [28]. Two test statistics, likelihood ratio and Akaike information criterion (AIC), are provided to compare model pairs that differ in complexity. The program is used together with PAUP* [33], typically as part of data exploration prior to more extensive phylogenetic analysis.

*MrBayes*    A program for phylogenetic analysis of nucleotide or amino acid sequence data using a Bayesian approach [31]. A Metropolis-coupled Markov Chain Monte Carlo (MCMCMC) algorithm is used with multiple chains, all but one of which is heated. The chains are used to sample model space through a process of parameter modification proposal and acceptance/rejection steps (also called cycles or generations). A choice of several commonly used likelihood models is available as are choices for starting tree (user-defined and random), data partitions (e.g., by codon position), and Markov Chain Monte Carlo parameters.

*ms*    A program that generates random independent sequence samples according to a simple Wright-Fisher neutral model [16]. If invoked with a minimum of options, it produces samples under a panmictic, equilibrium model without recombination. By specifying various options on the command line the model can include recombination, island-model type structure, gene conversion and simple population size changes in the past.

*Muscle*    A program for multiple DNA or protein sequence alignment [10].

*PHYML*    A program program for phylogenetic analysis of sequence data using maximum likelihood methods [13].

*Pknots*    PKNOTS implements a dynamic programming algorithm for predicting optimal RNA secondary structure, including pseudoknots [30]. The implementation generates the optimal minimum energy structure for a single RNA sequence, using standard RNA folding thermodynamic parameters augmented by a few parameters describing the thermodynamic stability of pseudoknots.

*Seq-Gen*    Seq-Gen is a program that will simulate the evolution of nucleotide or amino acid sequences along a phylogeny, using common models of the substitution process [29].

*Snn*   A program that performs the "nearest neighbor test" to detect genetic differentiation [15]. Given a matrix of pairwise differences between sampled sequences, Snn can determine genetic differentiation among local sample groups.

*SSEARCH*   A program for doing Smith-Waterman search for similarity between a query sequence and a group of sequences of the same type (nucleic acid or protein) [32]. This may be the most sensitive method available for similarity searches. It is part of the FASTA package [27].

## 1.7   PERFORMANCE

Given that the middleware provided here is quite thin, it is unsurprising that it has negligible impact on performance. The elapsed time between using globusrun-ws to submit a job to the BOINC server and seeing the work unit corresponding to that job appear in the queue on the sever is minimal, on the order of one or two seconds, and timing runs have shown this delay to be attributable to overhead associated with Globus.

Depending on the number of jobs and on their data requirements, it is conceivable that the BOINC server could find itself required to store and forward large amounts of data. BOINC is able to support multiple data servers, so perhaps default GRAM behavior could be changed such that files are transferred from the submitting node to a cluster of such data servers, from which BOINC clients would then download.

## 1.8   PRODUCTION USE — EXAMPLES

This middleware has been tested and used extensively in our Globus Toolkit-based development and production Grid system, The Lattice Project [3]. We have scaled up to thousands of jobs running concurrently under this framework as described. BOINC was designed to accommodate this type of load, readily scaling up to millions of work units. The limiting variable for this system thus becomes the number of concurrent GRAM jobs that can be effectively managed, and this is something the Globus team

continues to improve. To help in this regard, we are currently working to add support for managing batches of similar work units in a single GRAM job instance.

The Lattice Project Grid system has been used to complete several research projects in computational biology using BOINC as the primary computing resource. Our Globus-BOINC based Grid system performed approximately 17.5 CPU years of computation over several months, during which users submitted jobs intermittently.

Ranjani Varadan in the the laboratory of David Fushman 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 [36]. The computations consumed approximately 12.4 CPU years.

Holly Mortensen and Floyd Reed in 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 polymorphism to estimate the times of divergence and migration rates among ethnically diverse populations in Africa [35]. The computations consumed approximately 5.1 CPU years.

## 1.9 SOFTWARE AVAILABILITY

The current version of our middleware is available for download from our Grid research 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 relevant publications where appropriate.

## 1.10   SUMMARY

We have developed middleware allowing the Globus Toolkit to dispatch work to a Desktop Grid managed by BOINC, tested it using real-world applications used extensively in bioinformatics, and used it in a production Grid system for a variety of computational biology problems. This middleware, though conceptually simple, serves as a bridge between two very different models of Grid computing.

By joining these two models, we deliver substantial advantages to users of both. On the Globus side, Grid users may gain access to a much wider pool of potential resources than was previously possible. On the BOINC side, Grid users may gain a more full-featured system; indeed, one could imagine using Globus outside of a full-scale Grid system simply to provide a convenient interface to a standalone BOINC-based Desktop Grid.

Finally, we believe that the integration of BOINC and the Globus Toolkit will foster a transformation in public computing. Previously, public computing infrastructures were exclusively domain-specific pieces of software, and establishing a public computing project required significant time and experience. Even using BOINC, one still has to develop custom scripts and interfaces so as to be able to manage the flow of work into and out of the BOINC server, which takes significant effort. Our middleware provides a tremendous enhancement to public computing: it allows public computing resources to be accessed using feature-rich and widely-used, standard protocols. It thus dramatically decreases the overhead associated with starting subsequent public computing projects once the initial infrastructure is established. It is our hope that this will allow public computing to be a viable option for a much wider range of research projects than it is currently.

BOINC; the members of the Globus developer-discuss mailing list for their help with Globus; the members of the Condor group at the University of Wisconsin-Madison for the help with Condor-G; and the UMIACS systems staff, who were of great help in setting up the software and hardware used in this project.

## REFERENCES

1. D. P. Anderson, "Public computing: reconnecting people to science," Conference on Shared Knowledge and the Web, Residencia de Estudiantes, Madrid, Spain, Nov. 17-19, 2003.

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

3. A. L. Bazinet, and M. P. Cummings, "The Lattice Project: a Grid research and production environment combining multiple Grid computing models," In Weber, M. H. W. (Ed.) *Distributed & Grid Computing — Science Made Transparent for Everyone. Principles, Applications and Supporting Communities*, Tectum. To appear.

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

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

6. P. Beerli, and J. Felsenstein, "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 (2001).

7. A. T. Brünger, P. D. Adams, G. M. Clore, W. L. DeLano, P. Gros, R. W. Grosse-Kunstleve, J.-S. Jiang, J. Kuszewski, M. Nilges, N. S. Pannu, R. J. Read, L. M.

Rice, T. Simonson, and G. L. Warren, "Crystallography & NMR system: a new software suite for macromolecular structure determination," *Acta Cryst.*, **D54,** 905–921 (1998).

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

9. M. P. Cummings, and J. C. Huskamp, "Grid computing", *EDUCAUSE Review*, **40,** 116–117 (2005).

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

11. I. Foster, and C. Kesselman, "Globus: a toolkit-based Grid architecture," In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, Los Altos, 1999.

12. J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *J. Cluster Computing*, **5,** 237–246 (2002).

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

14. J. Hey, and R. Nielsen, "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 (2004).

15. R. R. Hudson, "A new statistic for detecting genetic differentiation," *Genetics* **155,** 2011–2014 (2000).

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

17. G. Hunt, and D. Brubacher, "Detours: Binary Interception of Win32 Functions," In *Proc. 3rd USENIX Windows NT Symposium*, Seattle, WA, July 1999. USENIX.

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

19. M. K. Kuhner, J. Yamato, and J. Felsenstein, "Maximum likelihood estimates of population growth rates based on the coalescent," *Genetics*, **149,** 429–434 (1998).

20. M. K. Kuhner, J. Yamato, and J. Felsenstein, "Maximum likelihood estimation of recombination rates from population data," *Genetics*, **156,** 1393–1401 (2000).

21. S. M. Larson, C. D. Snow, M. Shirts, and V. S. Pande, "Folding@Home and Genome@Home: using distributed computing to tackle previously intractable problems in computational biology," In R. Grant, editor, *Computational Genomics: Theory and Applications*, Horizon Scientific Press, Norfolk, 2004.

22. M. Litzkow, M. Livny, and M. Mutka, "Condor — a hunter of idle workstations," In *Proc. 8th International Conference of Distributed Computing Systems*, San Jose, 1988.

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

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

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

26. R. Nielsen, and J. Wakeley, "Distinguishing migration from isolation. A Markov chain Monte Carlo approach," *Genetics*, **158,** 885–896 (2001).

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

28. D. Posada, and K. A. Crandall, "Modeltest: testing the model of DNA substitution," *Bioinformatics*, **14,** 817–818 (1998).

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

30. E. Rivas, and S. R. Eddy, "A dynamic programming algorithm for RNA structure prediction including pseudoknots," *J. Mol. Biol.*, **285,** 2053–2068 (1999).

31. F. Ronquist, and J. P. Huelsenbeck, "MrBayes 3: Bayesian phylogenetic inference under mixed models," *Bioinformatics*, **19,** 1572–1574 (2003).

32. T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, **147,** 195–197 (1981).

33. D. L. Swofford, "PAUP*: Phylogenetic analysis using parsimony (*and other methods), version 4," Sinauer Associates, Sunderland, Massachusetts, USA, 2003.

34. J. D. Thompson, D. G. Higgins, and T. J. Gibson, "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 (1994).

35. S. A. Tishkoff, M. K. Gonder, M. Brenna, B. M. Henn, H. Mortensen, N. Fernandopulle, C. Gignoux, G. Lema, T. B. Nyambo, P. A. Underhill, U. Ramakrishnan, F. A. Reed, and J. L. Mountain, "History of click-speaking populations of Africa inferred from mtDNA and Y chromosome genetic variation," Submitted.

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

37. http://boinc.berkeley.edu/.

38. http://fightaidsathome.scripps.edu/.

39. http://www.openpbs.org/.