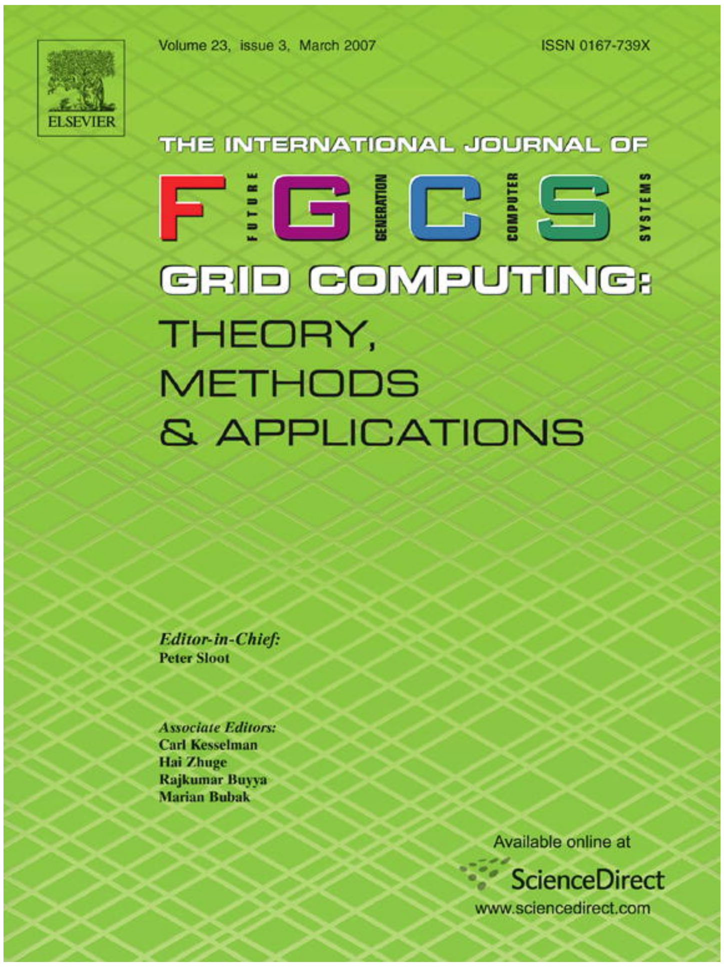
Volume 23, issue 3, March 2007
ISSN 0167-739X
ELSEVIER
THE INTERNATIONAL JOURNAL OF
FGCS
FUTURE GENERATION COMPUTER SYSTEMS
GRID COMPUTING: THEORY, METHODS & APPLICATIONS
Editor-in-Chief: Peter Sloot
Associate Editors: Carl Kesselman, Hai Zhuge, Rajkumar Buyya, Marian Bubak
Available online at ScienceDirect
www.sciencedirect.com

Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 23 (2007) 517–522
FGCS
FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Grid Services Base Library: A high-level, procedural application programming interface for writing Globus-based Grid services

Adam L. Bazinet[a], Daniel S. Myers[a,1], John Fuetsch[a,b,2], Michael P. Cummings[a,*]

[a] Center for Bioinformatics and Computational Biology, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, USA
[b] Department of Mathematics and Computer Science, Pomona College, Claremont, CA 91711, USA

Received 15 December 2005; received in revised form 26 April 2006; accepted 15 July 2006
Available online 22 August 2006

Abstract

The Grid Services Base Library (GSBL) is a procedural application programming interface (API) that abstracts many of the high-level functions performed by Globus Grid services, thus dramatically lowering the barriers to writing Grid services. The library has been extensively tested and used for computational biology research in a Globus Toolkit-based Grid system, in which no fewer than twenty Grid services written with this API are deployed.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Client/server; Parallel processing; Web-based services; Software libraries; Object-oriented programming; Distributed systems

1. Introduction

As the size and complexity of life science data has increased, so have the sophistication and computational complexity of data analysis increased. Entire data types that did not exist a relatively short time ago (e.g., complete genome sequences, large-scale microarray results, large multilocus genotypes) now constitute much of life science data. Similarly, analytical challenges including inference and combinatorial optimization have been attacked with computer-intensive methods (e.g., stochastic simulation, machine learning methods, Bayesian analysis, Markov-chain Monte Carlo sampling). As a consequence, the computational demands of life science research continue to increase. Therefore, some life science researchers are turning to Grid computing to meet their computing resource needs, following a trend toward Grid computing in academia in general [10]. However, there are several barriers to widespread use of Grid computing in the life sciences, including the lack of Grid-enabled applications and the difficulty of producing them, the deficit of Grid computing resources available for life science research, and the difficulty of using Grid computing effectively. Several of these barriers to the use of Grid computing in the life sciences are being addressed [22]. The objective of this paper is to describe middleware tools that address one specific barrier mentioned above, difficulty creating Grid-enabled applications.

Grid computing has been defined [10] as 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.

Our ongoing research and development in Grid computing has 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 [23], which was used for a large-scale simulation study [9]. Our

* Corresponding author. Tel.: +1 301 405 9903; fax: +1 301 314 1341.
E-mail addresses: pknut777@umiacs.umd.edu (A.L. Bazinet), dsmyers@mit.edu (D.S. Myers), jfuetsch@pdi.com (J. Fuetsch), mike@umiacs.umd.edu (M.P. Cummings).

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

[2] Present address: PDI/DreamWorks, 1800 Seaport Boulevard, Redwood City, CA 94063, USA.

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.07.009

subsequent work has made use of the Grid middleware Globus [12] and the Berkeley Open Infrastructure for Network Computing (BOINC) [3,7], and focused on expanding the reach of Grid computing by creating a system that combines these two models (Myers, Bazinet and Cummings, in preparation). The work described here extends and complements other efforts [22] and represents our approach to making it easier to develop Grid-enabled applications using the Globus Toolkit. Although our focus is on applications used in computational biology and bioinformatics, the middleware solution we developed is general and is applicable to other domains.

The Globus Toolkit 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, on-the-fly resource 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.

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, 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 (which was released in early 2005). The challenge of writing Grid services with the Globus Toolkit has remained constant, however, and we will now describe some of the implementation challenges we have encountered and our approaches to overcoming them.

2. The challenges of working with Globus

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

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

Third, because the Globus Toolkit is research software under continual development, there is always the possibility that the API presented to Grid services will change between versions. This is precisely what happened between 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.

Finally, creating a new Grid service requires creating a number of new files in a very specific directory structure and with very specific names, namespaces, and classes. This is a tedious and error-prone process at best, but one we have to repeat each time we write a Grid service. Moreover, because we 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 it was desirable to reduce the overhead associated with this process as much as possible.

3. Our solution

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

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 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 (see Section 4 for a brief description of our Grid services). A remote 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 Grid Resource Allocation and Management (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.

3.1. Initial configuration of client–service interaction

There are several steps that a Globus Grid client needs to take in order to establish communication with a Grid service.

Because our Grid services are implemented with the WS-Resource Framework (Web Services Resource Framework, WSRF), these services provide users with the ability to access and manipulate state (i.e., data values that persist across service interactions). Following standard Globus Toolkit 4 conventions, each of our Grid services is composed of a Factory service and an Instance service. When a client requests resource creation, it contacts the Factory service. When a client requests that an operation be performed on a specific resource, it contacts the corresponding Instance service.

Thus, assuming the WS-Resource Factory pattern is in use, the client first contacts a Factory service which in turn creates and initializes a new resource. The Factory service returns an endpoint reference to a WS-Resource composed of an Instance service and the recently created resource. The interface of the Instance service object has been defined in Web Services Description Language (WSDL), and the associated resource provides state for this particular Grid service Instance.

This process requires a significant amount of relatively dense code, code which is nearly identical between Grid services. Unfortunately, although the overall logic remains constant, the classes involved do change, because each Grid service is uniquely typed. Moreover, there is no super-type for the classes, and the names of the functions to be called depend on the name of the service (for example, one has to call get[ServiceName]FactoryPortTypePort() and get[ServiceName]PortTypePort()), so placing this logic into a library is not straightforward: neither subclassing nor templating is effective.

In order to place this code in a library, we made use of the Java reflection APIs. The constructor for the Grid client base class takes as parameters a Class object representing the type of the class used to contact the Factory service, and a Class object representing the type of the class used to contact the Instance service; using these objects, it can create new instances of these classes without prior knowledge of their type. To call the creation method (whose name varies based on the name of the Grid service), we use the reflection API to search the methods of the locator object for a method whose name and signature match that which we know we need; then we obtain a reference to this method and call it on our object.

To reiterate, when this initial setup is complete, a new Grid resource will be created for this particular job request and a handle to an Instance service will be returned to the client. This handle is used to contact the Grid service when performing the operations discussed in the next few sections.

3.2. Argument processing

The applications most often used in our computational biology research can frequently accept a large number of command-line arguments (e.g., SSEARCH, part of the FASTA package [27] has ∼24 arguments). The straightforward Globus solution to representing these parameters in a Grid services context would be to create a complex type to hold them, and pass an instance of this complex type from the client to the Grid service. Although this approach is adequate in many cases, it does not fully meet our needs. Defining a type to handle configuration parameters is helpful, but when such a type has dozens of fields, some sort of additional support is needed: manually copying user input into and out of such a type becomes tedious and error-prone.

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

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

The WSDL required to describe the complex type corresponding to the arguments is automatically generated from this XML file using our Grid Services Generator (see Section 3.5). Perl scripts that accept the configuration parameters as command-line arguments, write them out to a specifically-formatted file, and then execute the Grid service client are also automatically generated. GSBL provides a class for the Grid service client that will read in this file and initialize an Instance of the custom type.

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

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

3.3. File transfers

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

Our original Globus Toolkit 3-based Grid system used the Global Access to Secondary Storage (GASS) protocol to send files between the client and the server, but we are now using GridFTP in conjunction with Reliable File Transfer (RFT) in our Globus Toolkit 4-based system. The GridFTP protocol provides for secure, robust, fast and efficient transfer of data. The Globus Toolkit also provides the most commonly used implementation of that protocol, composed of a server implementation and a scriptable command-line client. In our system, a GridFTP server runs on both the client and the service side, thus enabling file transfer between them. RFT is a WSRF compliant web service that provides scheduler-like functionality for data movement. Provided with a list of source and destination URLs (e.g., gsiftp://localhost/foo), the service writes the file transfer description into a database and then moves the files on the user's behalf using the underlying GridFTP software. At the highest level, therefore, GSBL negotiates with the RFT service to initiate file transfers, which in turn makes recourse to GridFTP to actually move data.

In GSBL, the ReliableFileTransferManager class is used to initiate and monitor file transfers. It accepts a list of files, an upload or download operation, and a local and remote endpoint. Once the transfer is initialized, one calls beginTransfer() to start the transfer in a separate thread. This call should be immediately followed by a call to waitComplete(), which will block until the file transfer job object has issued its "transfer complete" notification. Using these two simple function calls, file transfer can be made to look procedural; at no point do developers have to concern themselves with event-handling. As a side note, the ability to transfer a batch of files in one method call marks an improvement over our Globus Toolkit 3-based system.

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

3.4. Creating and monitoring GRAM jobs

Our Grid services need to submit GRAM jobs to remote computational resources on behalf of the client. These jobs may have to wait in a remote queuing system for some period of time, and even once execution begins, processing can take a long time. As such, the Globus API for submitting GRAM jobs is an asynchronous, event-based construction.

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

When a client calls runService(), passing along the complex argument type discussed in Section 3.2, this Grid service method prepares to create the GRAM job and returns immediately to the client, which may then terminate. From this point on, the service is in charge of submitting and monitoring the job, and is also responsible for transferring output files back to the client host when the job is finished.

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

3.5. 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), 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, 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 WSDL file for both the Factory service and the Instance service, 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. The following section describes the various Grid services that we have built to date.

4. Grid-enabled applications

We have built twenty 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 BLAST [1,2], ClustalW [36], CNS [8], *gsi* (Bazinet and Cummings in preparation; Cummings, Neel and Shaw in preparation), IM [16], LAMARC [19–21], MARXAN [4, 29], MDIV [26], Migrate [5,6], Modeltest [28], MrBayes [33], ms [18], Muscle [11], PAUP* [35], Phyml [15], Pknots [32], Seq-Gen [31], Snn [17], SSEARCH [27,34], and Structure [30].

The bioinformatics 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. Some of these bioinformatic analyses have not yet been addressed by other Grid systems. However, GSBL is generally applicable to the development of any computational Grid service under Globus.

5. Research projects using the Grid

A significant amount of scientific research computing was completed using our Globus Toolkit 3-based Grid system by several groups at the University of Maryland: the Laboratory of David Fushman, the Laboratory of Sarah Tishkoff, the Laboratory of Maile Neel, and the Laboratory of 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 [37]. The computation for this problem was primarily done using BOINC, and the accumulated processing time was approximately 12.4 CPU years (achieved in only a few months real-time).

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 polymorphism to estimate the times of divergence and migration rates among ethnically diverse populations in Africa. The computations were done using our globally-distributed BOINC resources (Myers, Bazinet and Cummings, in preparation), and the accumulated processing time was approximately 5.1 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 [18] 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 (Bazinet and Cummings, in preparation; Cummings, Neel and Shaw, in preparation). The many millions of individual analyses required consumed over 94 CPU years.

Across these three studies our GT3-based production Grid system performed approximately 112 CPU years of work in about 7 months of wall-clock time. Such studies are only possible using large collections of resources such as a Grid computing system.

As we transition to our Globus Toolkit 4-based system, we have made the Grid system available for limited production use for testing purposes. Two researchers are studying problems involved in the design of reserve networks for biological conservation using MARXAN, and collectively have consumed several CPU centuries 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 [24,25]. Joanna Grand, a National Science Foundation Post Doctoral Fellow in Biological Informatics, studies the consequences of incomplete and partial data in the design of conservation reserve networks.

6. Current efforts

We are completing an upgrade of our production Grid system to use Globus Toolkit 4 and the most recent version of BOINC [7], which we combine (Myers, Bazinet and Cummings, in preparation). We are currently testing GSBL in this new Globus Toolkit 4 environment, and we are updating surrounding infrastructure to put the new system into production. 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 [22].

7. Conclusions

We have designed, developed, tested, and put into production a library for building Globus Grid services and Grid clients. GSBL dramatically lowers the barriers to writing Grid services by providing a high-level, procedural API, and has been used to create Grid services for a number of bioinformatics applications that are extensively used in computational biology studies.

8. Software Availability

Recent versions of GSBL and the GSG are 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 where appropriate.

Acknowledgments

We thank Deji Akinyemi, Jonathan Howard, Stephen McLellan and Christopher Milliron for developing some of the Grid services; Borja Sotomayor for his tutorials on creating Globus Grid services [13,14]; and the UMIACS systems staff, who were of great help in setting up the software and hardware used in this project.

References

[1] S. Altschul, W. Gish, W. Miller, E. Myers, D.J. Lipman, Basic local alignment search tool, J. Mol. Biol. 215 (1990) 403–410.
[2] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, D.J. Lipman, Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, Nucleic Acids Res. 25 (1997) 3389–3402.
[3] D.P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, D. Werthimer, SETI@home: An experiment in public-resource computing, Commun. ACM 45 (11) (2002) 56–61.

[4] I.R. Ball, H.P. Possingham, Marine Reserve Design Using Spatially Explicit Annealing, a Manual, MARXAN (V1.8.2) (2000).
[5] P. Beerli, J. Felsenstein, Maximum likelihood estimation of migration rates and effective population numbers in two populations using a coalescent approach, Genetics 152 (1999) 763–773.
[6] P. Beerli, 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 (2001) 4563–4568.
[7] Berkeley Open Infrastructure for Network Computing, http://boinc.berkeley.edu/.
[8] 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, G.L. Warren, Crystallography & NMR system: A new software suite for macromolecular structure determination, Acta Crystallogr. D 54 (1998) 905–921.
[9] M.P. Cummings, S.A. Handley, D.S. Myers, D.L. Reed, A. Rokas, K. Winka, Comparing bootstrap and posterior probability values in the four taxon case, Syst. Biol. 52 (2003) 477–487.
[10] M.P. Cummings, J.C. Huskamp, Grid computing, EDUCAUSE Review 40 (2005) 116–117.
[11] R.C. Edgar, MUSCLE: Multiple sequence alignment with high accuracy and high throughput, Nucleic Acids Res. 32 (2004) 1792–1797.
[12] I. Foster, C. Kesselman, Globus: A toolkit-based Grid architecture, in: I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, Los Altos, CA, 1999, pp. 259–278.
[13] Globus Toolkit 3 Tutorial, http://gdp.globus.org/gt3-tutorial/.
[14] Globus Toolkit 4 Tutorial, http://gdp.globus.org/gt4-tutorial/.
[15] S. Guindon, O. Gascuel, A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood, Syst. Biol. 52 (2003) 696–704.
[16] J. Hey, 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 (2004) 747–760.
[17] R.R. Hudson, A new statistic for detecting genetic differentiation, Genetics 155 (2000) 2011–2014.
[18] R.R. Hudson, Generating samples under a Wright-Fisher neutral model of genetic variation, Bioinformatics 18 (2002) 337–338.
[19] M.K. Kuhner, J. Yamato, J. Felsenstein, Estimating effective population size and mutation rate from sequence data using Metropolis-Hastings sampling, Genetics 140 (1995) 1421–1430.
[20] M.K. Kuhner, J. Yamato, J. Felsenstein, Maximum likelihood estimates of population growth rates based on the coalescent, Genetics 149 (1998) 429–434.
[21] M.K. Kuhner, J. Yamato, J. Felsenstein, Maximum likelihood estimation of recombination rates from population data, Genetics 156 (2000) 1393–1401.
[22] S. Lee, T.D. Wang, N. Hashmi, M.P. Cummings, Bio-STEER: A semantic web workflow tool for Grid computing in the life sciences, Future Gener. Comput. Syst., in press (doi:10.1016/j.future.2006.07.011).
[23] D.S. Myers, M.P. Cummings, Necessity is the mother of invention: A simple Grid computing system using commodity tools, J. Parallel Distr. Comt. 63 (2003) 578–589.
[24] M.C. Neel, M.P. Cummings, Effectiveness of conservation targets in capturing genetic diversity, Conserv. Biol. 17 (2003) 219–229.
[25] M.C. Neel, M.P. Cummings, Genetic consequences of ecological reserve design guidelines: An empirical investigation, Conserv. Genet. 4 (2003) 427–439.
[26] R. Nielsen, J. Wakeley, Distinguishing migration from isolation. A Markov chain Monte Carlo approach, Genetics 158 (2001) 885–896.
[27] W.R. Pearson, D.J. Lipman, Improved tools for biological sequence comparison, Proc. Natl Acad. Sci. USA 85 (1988) 2444–2448.
[28] D. Posada, K.A. Crandall, Modeltest: Testing the model of DNA substitution, Bioinformatics 14 (1998) 817–818.
[29] H.P. Possingham, I.R. Ball, S. Andelman, Mathematical methods for identifying representative reserve networks, in: S. Ferson, M. Burgman (Eds.), Quantitative methods for conservation biology, Springer-Verlag, New York, 2000, pp. 291–305.
[30] J.D. Pritchard, M. Stephens, P. Donnelly, Inference of population structure using multilocus genotype data, Genetics 155 (2000) 945–959.
[31] A. Rambaut, N.C. Grassly, Seq-Gen: An application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees, Comput. Appl. Biosci. 13 (1997) 235–238.
[32] E. Rivas, S.R. Eddy, A dynamic programming algorithm for RNA structure prediction including pseudoknots, J. Mol. Biol. 285 (1999) 2053–2068.
[33] F. Ronquist, J.P. Huelsenbeck, MrBayes 3: Bayesian phylogenetic inference under mixed models, Bioinformatics 19 (2003) 1572–1574.
[34] T.F. Smith, M.S. Waterman, Identification of common molecular subsequences, J. Mol. Biol. 147 (1981) 195–197.
[35] D.L. Swofford, PAUP*: Phylogenetic analysis using parsimony (*and other methods), version 4, Sinauer Associates. Sunderland, MA, USA.
[36] J.D. Thompson, D.G. Higgins, 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 (1994) 4673–4680.
[37] R. Varadan, M. Assfalg, S. Raasi, C. Pickart, D. Fushman, Structural determinants for selective recognition of a Lys48-linked polyubiquitin chain by a UBA domain, Mol. Cell 18 (2005) 687–698.

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.

Daniel S. Myers received a B.A. degree in computer science from Pomona College, Claremont, California; worked as a Research Assistant in the Center for Bioinformatics and Computational Biology, University of Maryland; and conducted research in France as a Fulbright Scholar. He is currently pursuing a graduate degree in computer science from the Massachusetts Institute of Technology. His interests include systems and Grid computing.

John Fuetsch received a B.A. degree in computer science from Pomona College, Claremont, California, and worked as a Research Assistant in the Center for Bioinformatics and Computational Biology, University of Maryland. He is currently a Technical Director at Dreamworks Animation working on Shrek the Third. His research interests include, appropriately, computer graphics and animation.

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.