Computing the Tree of Life

Leveraging the Power of Desktop and Service Grids

Adam L. Bazinet and Michael P. Cummings
Center for Bioinformatics and Computational Biology
University of Maryland
College Park, MD 20742 USA
pknut777@umiacs.umd.edu, mike@umiacs.umd.edu

***Abstract*—The trend in life sciences research, particularly in molecular evolutionary systematics, is toward larger data sets and ever-more detailed evolutionary models, which can generate substantial computational loads. Over the past several years we have developed a grid computing system aimed at providing researchers the computational power needed to complete such analyses in a timely manner. Our grid system, known as The Lattice Project, was the first to combine two models of grid computing – the service model, which mainly federates large institutional HPC resources, and the desktop model, which harnesses the power of PCs volunteered by the general public. Recently we have developed a "science portal" style web interface that makes it easier than ever for phylogenetic analyses to be completed using GARLI, a popular program that uses a maximum likelihood method to infer the evolutionary history of organisms on the basis of genetic sequence data. This paper describes our approach to scheduling thousands of GARLI jobs with diverse requirements to heterogeneous grid resources, which include volunteer computers running BOINC software. A key component of this system provides *a priori* GARLI runtime estimates using machine learning with random forests.**

Keywords—volunteer computing; BOINC; grid computing; Globus; The Lattice Project; science portal; web interface; scheduling; phylogenetics; GARLI; random forests.

I. INTRODUCTION

The trend in life sciences research, particularly in molecular evolutionary systematics, is toward larger data sets and ever-more detailed evolutionary models, which can generate substantial computational loads. In order to complete the many Assembling the Tree of Life (AToL) projects [1], we have been focused on providing investigators with the computational power required to complete phylogenetic analyses in a timely manner. Our grid system, known as The Lattice Project, was the first to combine two models of grid computing – the service model, which mainly federates large institutional HPC resources, and the desktop model, which harnesses the power of PCs volunteered by the general public [2]. The motivating idea is to utilize powerful dedicated institutional resources when it is convenient or advantageous to do so, but also to rely heavily on the abundance of freely available, otherwise idle volunteer computing resources. In this paper we will describe a new web interface for performing phylogenetic analyses, as well as sophisticated scheduling techniques that allow us to make efficient use of conventional computing resources in service grids as well as the dynamic, heterogeneous resources that make up desktop grids. First, however, we provide some background on the needs of AToL projects and others working in the field of molecular evolution; The Lattice Project, our grid computing system developed to address these needs; and GARLI, our featured phylogenetic analysis program.

II. BACKGROUND

A. Grid, Public, and GPU Computing for the Tree of Life

Our research group has been charged specifically with developing a system that can be used by AToL projects and others to complete large phylogenetic analyses in a timely manner. Such analyses typically consist of a heuristic search through an inexhaustibly large multidimensional space. Programs that perform such analyses seek to improve speed (time to results), performance (quality of solutions), and include support for different evolutionary models. These programs can run for hours, weeks, or months depending on the size of the input data and the complexity of the model, and it is frequently necessary to perform multiple searches for the "best tree", or solution with the highest likelihood, as well as hundreds or thousands of bootstrap searches which assess confidence in the best tree by sampling the original data with replacement [3]. Such jobs can also be memory intensive, requiring multiple gigabytes of memory.

In order to process this massive amount of computation as quickly as possible, we leverage our grid computing system, which is composed of computing resources at the University of Maryland and neighboring institutions, as well as a dynamic pool of volunteer computing resources distributed throughout the world. These resources total well over 5000 CPU cores, and will scale up with demand. In addition to distributing the computation, we have also endeavored to speed up the analysis programs themselves. Hence, we have invested significant time developing BEAGLE (Broad-platform Evolutionary Analysis General Likelihood Evaluator) [4], a library that uses graphics processing units (GPUs) to speed up the likelihood calculations at the heart of most phylogenetic analysis programs. As we will describe further on, we have also emphasized building intuitive user interfaces to the system. Our system delivers all of the needed computational power to the research community, but is also feature-rich and easy to use.

B. *The Lattice Project*

The Lattice Project is a grid computing system that represents nearly a decade of work. Our first grid computing system built from commodity tools [5] completed a 15 CPU year simulation study of phylogenetic bootstrap and posterior probability values [6] in just a few months. Subsequent development yielded a system that combines multiple types of grid middleware (software that resides between the operating system and the applications). Specifically, our system allows grids based on Globus (the current state of the art in service grid middleware [7]) and on BOINC [8] to inter-operate [9], thus permitting incorporation of a much wider range of resources. BOINC (Berkeley Open Infrastructure for Network Computing) is open-source middleware designed for so-called public computing, i.e., grids incorporating computers volunteered by individuals, as in the well-known SETI@home project [10]. The current incarnation of our system, The Lattice Project [11], encompasses clusters, institutional desktops aggregated into Condor pools [12], and PCs using BOINC. It has been used recently in studies of conservation biology [13], pandemic influenza [14], human evolution [15], protein binding [16], quantifying lineage divergence [17], phylogenetic analyses of various organisms (e.g., Lepidoptera [18-20]; arthropods [21-22]; plants [23-34]; and rotifers [25]), and in total has performed computation in excess of 20,000 CPU years. Our BOINC project has involved 11,142 volunteers and 23,192 public desktop computers in 121 different countries.

C. *GARLI*

GARLI (Genetic Algorithm for Rapid Likelihood Inference) is an open-source phylogenetic inference program that uses the maximum likelihood (ML) criterion. Actively developed by Derrick Zwickl [26, 27], it is loosely based on earlier genetic algorithm work by Paul Lewis [28]. GARLI uses an evolutionary algorithm to search for the likelihood optimum in the joint space of tree topologies, branch lengths and model parameter values. The program was developed with the goals of increasing both the speed of ML inference and the size of the datasets that can reasonably be analyzed. This is achieved primarily through algorithmic techniques that allow for accurate discrimination between trees while performing only a small fraction of the computation required by older methods. The program is being adapted to accommodate novel analysis features of AToL projects by allowing more data types, partitioned models, efficient analysis of incomplete data sets, and topological recombination. The GARLI service available through The Lattice Project allows a user to submit large numbers of GARLI jobs, which in turn execute on our grid resources. For BOINC, we developed a special version of GARLI that includes features such as checkpointing and BOINC client progress bar updates.

III. GARLI Web Interface

For several years, the primary means of grid job submission in our system was through a command line interface on a UNIX machine. We thought that since researchers were used to running these programs from the command line, interfacing with the grid this way would feel natural to them. Thus, grid users would upload input data to this UNIX machine, submit their jobs to a particular grid service, and download their results from this machine when the jobs were finished. We also provided utilities that allowed a user to query the status of jobs in the system, and cancel jobs that were no longer needed.

The command line interface is still perfectly useable, but requires some basic knowledge of UNIX, which is an impediment for many biologists. Given the popularity of science portals such as CIPRES [29] and the Oslo Bioportal [30], we knew that a web-based interface to our services would generate considerably more interest, and would be easier for the majority of people to use. Hence, we developed software that takes an XML description of grid application arguments and options and automatically generates a Drupal [31] web interface for that application (Fig. 1). Building within the Drupal framework affords a number of advantages, such as built-in authentication and authorization, form validation, and easy theming and integration with other Drupal content. Drupal also has a large number of actively developed modules that provide additional functionality such as the ability to masquerade as other web site users, CAPTCHAs, and so on. We have developed our software as a Drupal module so others can plug it in to their Drupal site if they wish, and we intend to integrate our interface generation software with our lower-level Grid Services Generator [32] eventually.

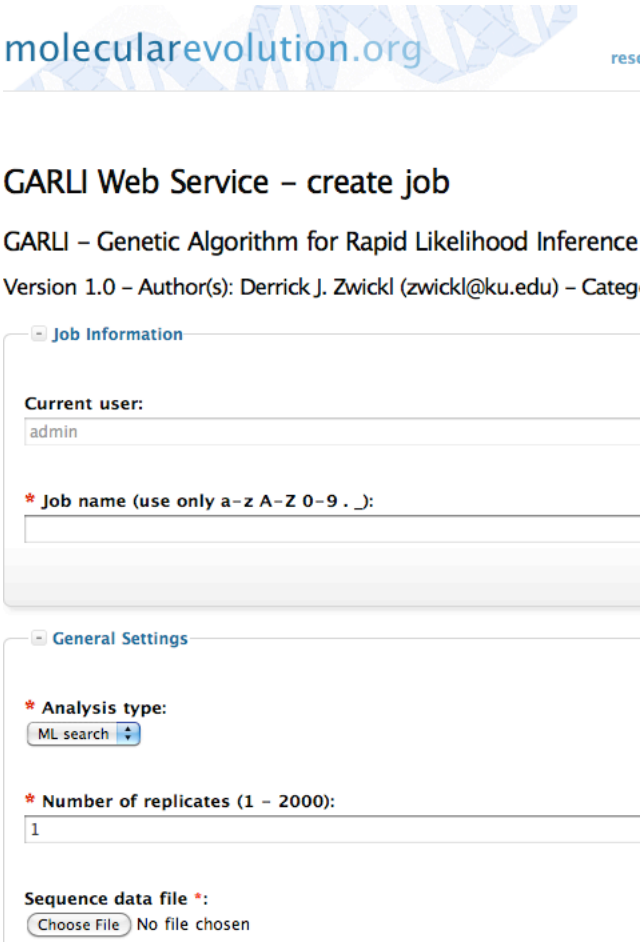

Figure 1. Screenshot of the GARLI web interface showing job creation at http://www.molecularevolution.org/software/phylogenetics/garli/garli_create_job.

A. Using the GARLI Web Interface

An investigator may use the GARLI web interface in a guest mode, in which they provide their email address for identification, or as a registered user which allows for more sophisticated job tracking features. The form allows one to upload any necessary input files (genetic sequence data, starting tree, or constraint file) and specify the desired application parameter settings. Our basic interface is similar to others in providing these features. What makes it uniquely powerful, however, is the ability to submit up to 2000 job replicates with a single submission. Before any jobs are scheduled, the system uses a special GARLI validation mode to ensure there are no problems with the data files and parameters specified. Each replicate is then scheduled to run in parallel on a separate processor in our grid system. The user is notified via email about important status updates (such as job completion or job failure). After all the job replicates are finished, the system automatically runs some post-processing on the results and makes them available in a single zip file for the user to download.

B. The Power of Distributed Computing

When a portal user submits a large number of jobs, the grid system breaks these up into smaller batches and may schedule each of these batches to a different grid computing resource. Whereas other science portals generally allow you to use only one processor or maybe a small handful, and often restrict how long a job may run for, our system currently has no such limitations. Eventually we may need to impose such limits on use, but on balance we expect to be able to offer significantly more computing power than other science portals. This is because we our resource base will automatically scale up to meet with demand by attracting more volunteer computers that run BOINC. The following section describes our grid resources in more detail.

IV. Grid 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. We argue that the large heterogeneity in types of research problems is best met with heterogeneous computational resources. For example, some problems may require a closely coupled parallel computing environment (e.g., a cluster with low latency, high bandwidth interconnections between nodes). Other problems are easily broken up into wholly independent processes, the results of which can be united to form a composite result (e.g., a parameter sweep). These are often called “embarrassingly parallel” problems, and are appropriately handled by desktop computing resources. Hence, our system includes a variety of these resource types.

Resource building is one of the principal activities associated with creating and expanding a grid system, and as such it has been one of our highest priorities. Beyond simply aggregating CPU power, resource building usually involves collaboration between different people and organizations. Such people may not know anything about grid computing, in which case we take the time to explain the goals of the project, the benefits of being involved with it, and the technical details that enable these groups to effectively contribute their local resources to the project. However, it is important to define what constitutes a local resource, how one can be created, and the policies and procedures that govern its use.

We define a *local resource* as an established computing resource administered in one domain and capable of functioning independently from the grid system. Users of a local resource submit and monitor compute jobs using a local resource manager (LRM), often simply called a "scheduler". Pools of computers running Condor software or dedicated clusters running Portable Batch System (PBS) software are common local resources. A typical grid system contains a grid-level scheduler, which assigns a computational job submitted at the grid level to an eligible local resource, where the job is then subsequently scheduled locally. This sort of hierarchical functionality is what makes grid computing appealing: it is the ability to use many different resources simultaneously and efficiently, wherein the grid system handles user authentication and authorization, job scheduling and monitoring, and data placement. The Lattice Project provides these features, thus enabling researchers to perform a large amount of computation in a short amount of time without having to worry about low-level details. In addition, the user gains access to computational resources outside of their administrative domain.

We have attempted to be all-inclusive with regard to the types of resources that can be integrated into our system. Currently we support resources federated with Condor, PBS, Sun Grid Engine (SGE) [33], or BOINC. Once a scheduler is installed on a local resource, that resource must be tied into the grid system by installing Globus components on a resource node that has the capability to submit jobs to the local scheduler. One of these components is called a *scheduler adapter* (formerly *job manager*). There exists a different scheduler adapter for each resource type. This is typically a collection of scripts responsible for translating a generic job description in Globus Resource Specification Language (RSL) or Job Submission Description Language (JSDL) format into a resource-specific job description (e.g., a Condor or PBS submit file). We have customized and extended the stock versions of the PBS and Condor adapters that come bundled with the Globus Toolkit, and have assembled an SGE adapter from various sources. We wrote our BOINC scheduler adapter completely from scratch. Another important resource-specific component is the *scheduler provider*, which collects information about the current state of a resource – e.g., number of free CPU cores, total RAM, total disk space, and so on. This information is used for job scheduling and monitoring.

As for some specific details about our computing resources, we support three major computing platforms: Linux, Windows, and Mac OS. Four different institutions are currently tied in to The Lattice Project: University of Maryland, Bowie State University, Coppin State University, and the Smithsonian Institution. Our resources include four Condor pools, four computing clusters, and an international pool of BOINC clients. As users of the grid system increase, so will demand for resources. The BOINC client pool can easily grow to meet this demand, but it will be necessary to continue to seek out other resources and new institutional partners. These resources are

already paid for; they are simply underutilized. Additionally, computing resources in a grid system with an intelligent grid-level scheduler are used more efficiently than if each resource was used independently. For example, jobs with large memory requirements can be sent to clusters with large memory nodes, and tightly coupled jobs (e.g., MPI jobs [34]) can be sent to clusters with fast interconnects. Pleasingly parallel jobs can be sent to a Condor or BOINC pool, and so on. The following section discusses grid-level scheduling in more detail.

V. GRID-LEVEL SCHEDULING

The scheduling component of any grid system is likely to be one of the most important and logically complex, since to a large extent it determines the overall efficiency of the system. This component is often called a *meta-scheduler* because it performs scheduling at the level above local resources. The scheduler must be informed about the present state of remote resources, a function performed by the Globus Monitoring and Discovery Service (MDS). For example, consider a Globus installation for which MDS has been configured to monitor the status of a Condor pool. In that case, the Condor scheduler provider will periodically parse the output of the Condor command "condor_status" to discover the total number of nodes in the pool, the number of nodes that are actually free (not bound to a machine owner or another computational process), and other information about the pool. This information is stored in XML format in the Globus container memory space and is valid for a short lifetime, typically on the order of minutes. The information in this MDS database can be periodically propagated to another MDS database running in another Globus container process. Using this mechanism, we aggregate all of the data about remote grid resources in a central MDS database, and query it to make scheduling decisions. Next we describe the scheduling algorithm in detail.

A. General Scheduling Algorithm

First, the scheduler needs to know which resources are reporting. If a remote installation goes offline, any jobs sent there will fail, so we cannot safely assume that our resources are always up and running. Thus, if we cease to receive MDS information from a certain resource, we mark the resource as "offline" and make sure no new jobs are scheduled there. Then the question becomes: of the resources that are reporting, to which one do we send a particular job? We recognize that a given job may not run on all resources, so the scheduler matches on various attributes to narrow down the possibilities. For example, the system keeps track of which CPU architecture and operating system combinations each application is compiled for (e.g., Intel/Mac OS X), and compares this list against the platforms each resource is advertising. Likewise, if the job has a minimum memory requirement, we filter out resources that do not meet the minimum memory criterion. Other resource requirements are also considered if necessary, such as MPI-capability and the presence or absence of software dependencies (e.g., Java). One can imagine any number of additional filtering and ranking criteria, especially around complex issues like policy – determining which users may access a particular resource, which users have priority over other users, when a particular resource may be used and for how long, and so on. We have not yet placed any such policy restrictions on resource use at the grid level, though it may be necessary to do so in the future. However, it is important to stress that when grid jobs run at the local resource level, they are always subject to the local policies that govern use of that resource by the local grid user account.

From the final set of eligible resources, the scheduler chooses a resource based on three criteria: current load on the resource, resource speed, and resource stability. The scheduler attempts to keep jobs from backing up on any single resource by spreading work around fairly evenly; however, such a naïve algorithm does not use resources very efficiently because it does not take into account resource speed. We can correct for resource speed if we can somehow measure it consistently, which is challenging for large, heterogeneous, often dynamic computing resources. Our procedure for actually accomplishing this varies, but the basic method is to run a short GARLI job on each unique individual machine that is part of a resource, and average the runtimes we collect. We compare this averaged runtime to the runtime from a "reference computer", which is arbitrarily assigned a speed of 1.0. If the job runs in half the time on the resource we are benchmarking, that resource is assigned a speed of 2.0 – in twice the time, a speed of 0.5 – and so forth. The final criterion, stability, refers to the likelihood that a particular job will be interrupted after some period of time. A "stable" resource will accommodate long-running jobs, whereas "unstable" resources will be suitable for shorter jobs. Thus, in order to choose a resource for a job, one must have an accurate, *a priori* job runtime estimate, which we discuss in the following section.

VI. GARLI RUNTIME ESTIMATES WITH RANDOM FORESTS

A. Motivation

Having accurate GARLI runtimes in advance of actual job scheduling is essential for a number of reasons. First, as mentioned above, it prevents long-running jobs from ending up on a resource where they do not have a chance of completing. Interruptions can occur because of interference from human users or other computational processes, limits on resource use, technical failures, and a variety of other reasons. Currently we simply designate a resource as either stable, or unstable – if it is unstable, we do not send it jobs estimated to take longer than n hours, where n is currently set to 10.

Second, having a runtime estimate allows us to deal with BOINC-specific scheduling issues. For example, we can programmatically specify reasonable workunit deadlines, which are needed on a volunteer computing platform to periodically reissue work if results are not received in a timely manner. To date, we have had to fill in this value manually for each batch of work we run through BOINC, which is not feasible if we wish to use public computing for the wide variety of GARLI jobs submitted via the web interface. In a similar vein, accurate runtime estimates allow the BOINC scheduler to hand out the proper amount of work when a client makes a work request, thus leading to greater efficiency since the client may not need to check in as often.

Third, if we find that someone has submitted jobs that are very short, e.g. a few minutes, we can ratchet up the number of search replicates each individual GARLI job will perform. Otherwise, for very short running jobs, the overhead of submitting each one independently substantially and negatively impacts performance gains from parallelization.

Fourth, in combination with other data, runtime estimates help us provide researchers with an idea of how long it will take for their jobs to complete, which is of great use in project planning and time management.

B. Challenges and Solutions

GARLI is a particularly challenging program to compute runtime estimates for. For one thing, the size of the input data can vary from modest (a few taxa, short sequences) to massive (hundreds or thousands of taxa, sequences thousands of characters in length). Furthermore, the program supports a variety of evolutionary models, some much more computationally intense than others. For example, amino acid and codon models tend to take substantially longer than nucleotide models to analyze the same data. Finally, since the program is genetic algorithm-like, there are numerous options for controlling the behavior of the algorithm itself that can have an impact on runtime. Our solution is to isolate all of the variables of interest and let a machine learning algorithm estimate, for a particular combination of input parameters, how long the job is going to take based on training data collected from previous GARLI runs. There are many machine learning methods for classification and regression [35]. The particular method we use is random forests, which we provide additional background on in the following section. The runtime estimate provided by the random forests model is scaled by the resource speed value described previously. This approach is in contrast to machine learning techniques for runtime prediction that are based solely on historical workload traces [36, 37].

An alternative to computing runtime estimates would be to modify GARLI to terminate after a predetermined period of time (e.g., one hour). After the job terminates, intermediate output would be collected and a new job would be sent out, picking up at the point where the previous job had left off via a checkpointing mechanism. We intend to explore this approach eventually, but we anticipate significant overhead resulting from terminating jobs and rescheduling them, and moving large amounts of data around in this process.

C. Random Forests

Random forests [38-41] is a machine learning technique developed by Breiman and Cutler to perform classification and regression using an ensemble of tree-based statistical models (hence, "forest") instead of just one, thus producing more accurate results. Final predictions are obtained by a voting scheme using the ensemble. Bagging [38] is an early example of this technique in which each tree is constructed from a bootstrap sample [42] drawn with replacement from the training data. Bagging reduces prediction error for unstable predictors, such as trees, by reducing the variance through averaging [38, 43]. Minimizing the correlation between the quantities being averaged can favorably enhance this effect, so random forests seek to effect such correlation reduction by a further injection of randomness. Instead of determining the optimal split of a given node of a constituent tree by evaluating all allowable splits on all covariates, as is done with single tree methods or bagging, a subset of the covariates drawn at random is employed. Breiman [40, 41] argues that random forests (a) display exceptional prediction accuracy, (b) that this accuracy is attained for a wide range of settings of the single tuning parameter employed, and (c) that overfitting does not arise due to the independent generation of ensemble members.

To estimate GARLI runtimes, we generated random forests made up of 1×10^4 individual trees constructed by sub-sampling nine predictor variables at each node. Variable importance was assessed by measuring the increase in group purity when partitioning data based on a variable. We used the R package randomForest [44, 45].

D. Model Building

In order to construct a model with random forests, it is necessary to decide which analysis parameters will be included. Based on a combination of our experience using GARLI, program documentation, and correspondence with the program author, we isolated all of the parameters that could possibly affect runtime, and excluded those that we do not allow users to modify via the GARLI web interface. Unlike other machine learning methods, random forests do not require variable

Figure 2. Importance of phylogenetic analysis parameters in predicting GARLI runtime as determined by random forest analysis and measured in terms of percent increase in mean square error.

(attribute) selection. Rather, they allow use of all possible variables, and the importance of each variable is quantified. This is illustrated in our model where the most important analysis parameter affecting runtime is use of a substitution rate heterogeneity model, with a difference in mean square error of 89.7%, followed by data type (nucleotide, amino acid, or codon) at 72.4%. In contrast, the number of rate categories has almost no importance. All analysis parameters and their effect on runtime prediction, as measured by percent increase in mean square error, are shown in Figure 2.

Approximately 150 GARLI jobs were used as training data; these represent a great diversity of "real" jobs that had been previously submitted by researchers using the grid system. The values of the nine predictor variables were recorded from each job, along with *runtime*, the response variable (measured in seconds). These values were arranged in a matrix and used to build a model with randomForest, which for this amount of data takes very little time to compute. The percentage of variance explained by these nine variables is approximately 93%, an excellent result. The model itself, stored as an R object, can subsequently be used to make a runtime prediction for a new set of predictor values. In our cross-validation testing, predicted runtimes matched the actual runtimes closely enough to greatly improve scheduling effectiveness. Because the model is an ensemble of 1×10^4 individual regression trees, it is not easily depicted here.

It should now be clear why random forests is a reasonable choice for this problem: (a) its runtime estimation performance is excellent, (b) it automatically produces a measure of variable importance, which allows us to better understand how parameters impact runtime, (c) it easily incorporates categorical and continuous variables in the analysis, and (d) it does not take much computational time to build or update the model.

E. Integration with the Grid-level Scheduler

After building the model, we programmed the system to use the model to produce a runtime estimate for each GARLI job submitted. More specifically, the system collects the values of the nine predictor variables and produces a runtime estimate via the predict() function in the randomForest package. This runtime estimate is subsequently used to do the things mentioned previously: (a) determine whether to send this job to a stable or unstable resource, scaling the runtime estimate used to make this decision by the speed of each resource, (b) give an accurate BOINC runtime estimate and wall clock deadline, (c) increase the number of search replicates in the case of very short-running jobs, and (d) provide the researcher a time estimate of how long it will take their submission to complete.

Since our training data did not cover the entire spectrum of possible values for the nine predictor values, and since GARLI itself is under constant development, we would like to continuously update the model based on information collected from incoming jobs. To do this, we simply fork off a single job replicate on our reference computer, which is actually a cluster of homogeneous machines, and add the observed runtime and values of the predictor variables to the matrix we use to build the model. Then we simply rebuild the model, which is immediately available for use with incoming jobs. In this manner the model is continually improved.

VII. CONCLUSION

We have described various aspects of The Lattice Project, a grid computing system specializing in molecular phylogenetic analyses. In particular, we have described a web interface that allows researchers to submit large numbers of GARLI jobs quickly and easily; the computing resources those jobs run on; and mechanisms for scheduling those jobs. Finally, we have described a novel method that produces runtime estimates for GARLI jobs using random forests. Such runtime estimates are critical in order to use our grid computing resources efficiently.

ACKNOWLEDGMENT

We gratefully acknowledge Derrick Zwickl for his help with GARLI and modifications to the program, Barry Dutton as the primary developer of the GARLI web interface, and Bradley Senft for incorporating GARLI runtime estimates into the BOINC scheduler adapter. The U.S. National Science Foundation award 0755048 provided funding for this project.

REFERENCES

[1] Assembling the Tree of Life (AToL) initiative, http://www.phylo.org/atol/.

[2] A. Bazinet and M. Cummings, "The Lattice Project: a Grid research and production environment combining multiple Grid computing models," in Distributed & Grid Computing — Science Made Transparent for Everyone. Principles, Applications and Supporting Communities, M. Weber, Ed. Marburg: Rechenkraft.net, 2008, ch. 1, pp. 2–13.

[3] J Felsenstein, "Confidence limits on phylogenies: An approach using the bootstrap", Evolution, 39: 783-791, 1985.

[4] beagle-lib, http://code.google.com/p/beagle-lib/.

[5] D. Myers and M. Cummings, "Necessity is the mother of invention: a simple grid computing system using commodity tools," J Parallel Distr Com, vol. 63, no. 5, pp. 578–589, May 2003.

[6] M. Cummings, S. Handley, D. Myers, D. Reed, A. Rokas, and K. Winka, "Comparing bootstrap and posterior probability values in the four-taxon case," Syst Biol, vol. 52, no. 4, pp. 477–87, Aug 2003.

[7] I. Foster and C. Kesselman, "Globus: a toolkit-based Grid architecture," in The Grid: Blueprint for a New Computing Infrastructure, I. Foster and C. Kesselman, Eds. Morgan-Kaufmann, 1999, pp. 259–278.

[8] Berkeley Open Infrastucture for Network Computing (BOINC), http://boinc.berkeley.edu/.

[9] D. Myers, A. Bazinet, and M. Cummings, "Expanding the reach of Grid computing: combining Globus- and BOINC-based systems," in Grids for Bioinformatics and Computational Biology, ser. Wiley Book Series on Bioinformatics: Computational Techniques and Engineering, E.-G. Talbi and A. Zomaya, Eds. Hoboken: Wiley-Interscience, 2008, ch. 4, pp. 71–85.

[10] SETI@home, http://setiathome.ssl.berkeley.edu.

[11] A. Bazinet, "The Lattice Project: A multi-model Grid computing system", Master's thesis, University of Maryland, http://hdl.handle.net/1903/9892, 2009.

[12] M. Litzkow, M. Livny, and M. Mutka, "Condor — a hunter of idle workstations," Proceedings of the 8th International Conference on Distributed Computing Systems, pp. 104–111, 1988.

[13] J. Grand, M. Cummings, T. Rebelo, T. Ricketts, and M. Neel, "Biased data reduce efficiency and effectiveness of conservation reserve networks," Ecol Lett, vol. 10, no. 5, pp. 364–74, May 2007.

[14] C. Dibble, S. Wendel, and K. Carle, "Simulating pandemic influenza risks of us cities," in Winter Simulation Conference, S. G. Henderson, B.

Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, Eds. WSC, pp. 1548–1550, 2007.

[15] S. A. Tishkoff, M. K. Gonder, B. M. Henn, H. Mortensen, A. Knight, C. Gignoux, N. Fer- nandopulle, G. Lema, T. B. Nyambo, U. Ramakrishnan, F. A. Reed, and J. L. Mountain, "History of click-speaking populations of Africa inferred from mtDNA and Y chromosome genetic variation," Mol Biol Evol, vol. 24, no. 10, pp. 2180–95, Oct 2007.

[16] 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," Molecular Cell, vol. 60, pp. 687–698, 2005.

[17] M. Cummings, M. Neel, and K. Shaw, "A genealogical approach to quantifying lineage divergence," Evolution, vol. 62, no. 9, pp. 2411–22, Sep 2008.

[18] J. Regier, A. Zwick, M. Cummings, A. Kawahara, S. Cho, S. Weller, A. Roe, J. Baixeras, J. Brown, C. Parr, D. Davis, M. Epstein, W. Hallwachs, A. Hausmann, D. Janzen, I. Kitching, M. Solis, S.-H. Yen, A. Bazinet, and C. Mitter, "Toward reconstructing the evolution of advanced moths and butterflies (Lepidoptera: Ditrysia): an initial molecular study," BMC Evol Biol, vol. 9, p. 280, 2009.

[19] A. Zwick, J. Regier, C. Mitter, and M. Cummings, "Increased gene sampling yields robust support for higher-level clades within Bombycoidea (Lepidoptera)," Syst Entomol, vol. in press, 2010.

[20] A. Y. Kawahara, A. A. Mignault, J. C. Regier, I. J. Kitching and C. Mitter, "Phylogeny and biogeography of hawkmoths (Lepidoptera: Sphingidae): evidence from five nuclear genes", PLoS ONE, 4(5): e5719, 2009.

[21] J. Regier, J. Shultz, A. Ganley, A. Hussey, D. Shi, B. Ball, A. Zwick, J. Stajich, M. Cummings, J. Martin, and C. Cunningham, "Resolving arthropod phylogeny: exploring phylogenetic signal within 41 kb of protein-coding nuclear gene sequence," Syst Biol, vol. 57, no. 6, pp. 920–38, Dec 2008.

[22] J. C. Regier, J. W. Shultz, A. Zwick, A. Hussey, B. Ball, R. Wetzer, J. W. Martin, and C. W. Cunningham, "Arthropod relationships revealed by phylogenomic analysis of nuclear protein-coding sequences," Nature, vol. 463, no. 7284, pp. 1079–83, Feb 2010.

[23] M. Neel and M. Cummings, "Section-level relationships of North American *Agalinis* (Orobanchaceae) based on DNA sequence analysis of three chloroplast gene regions," BMC Evol Biol, vol. 4, p. 15, Jun 2004.

[24] S. Marten-Rodrıguez, C. B. Fenster, I. Agnarsson, L. E. Skog, and E. A. Zimmer, "Evolutionary breakdown of pollination specialization in a Caribbean plant radiation," New Phytologist, vol 188, no. 2, pp.403-17, Jun 2010.

[25] M. Reyna-Fabian, J. Laclette, M. Cummings, and M. Garcia-Varela, "Validating the systematic position of *Plationus* Segers, Murugan & Dumont, 1993 (Rotifera: Brachionidae) using sequences of the large subunit of the nuclear ribosomal DNA and of cytochrome C oxidase," Hydrobiologia, vol. 644, no. 1, pp. 361–370, May 2010.

[26] Genetic Algorithm for Rapid Likelihood Inference, http://www.bio.utexas.edu/faculty/anti-sense/garli/garli.html.

[27] D. J. Zwickl, "Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion", PhD dissertation, The University of Texas at Austin, 2006.

[28] P. O. Lewis, "A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data", Molecular Biology and Evolution, 15:277–283, 1998.

[29] Cyberinfrastructure for Phylogenetic Research (CIPRES), http://www.phylo.org/.

[30] University of Oslo Bioportal, http://www.bioportal.uio.no/.

[31] Drupal - Open Source CMS, http://www.drupal.org/.

[32] Bazinet, A. L., D. S. Myers, J. Fuetsch and M. P. Cummings, "Grid Services Base Library: a high-level, procedural application program interface for writing Globus-based Grid services", Future Generation Computer Systems 23:517–522, 2007.

[33] Sun Grid Engine, http://gridengine.sunsource.net/.

[34] W. Gropp, E. Lusk, and A. Skjellum, "Using MPI, portable parallel programming with the Message-Passing Interface", MIT Press, Cambridge, MA, 1994.

[35] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques", Informatica, p. 249, 2007.

[36] H. Li, D. Groep, L. Wolters, "An evaluation of learning and heuristic techniques for application run time predictions", Proceedings of 11th Annual Conference of the Advance School for Computing and Imaging (ASCI), 2005.

[37] C. Glasner and J. Volkert, "An architecture for an adaptive run-time prediction system", Proceedings of the 7th International Symposium on Parallel and Distributed Computing (ISPDC'08), 2008.

[38] L. Breiman, "Bagging predictors", Machine Learning 24:123-140, 1996.

[39] L. Breiman, "Arching classifiers (with discussion)", Ann Stat 26:801-849, 1998.

[40] L. Breiman, "Random Forests", Machine Learning 45:5-32, 2001.

[41] L. Breiman, "Statistical modeling: the two cultures", Stat Sci 16:199-215, 2001.

[42] B Efron, R Tibshirani, "An Introduction to the Bootstrap", New York: Chapman & Hall, 1993.

[43] TJ Hastie, R Tibshirani, J. H. Friedman, "The Elements of Statistical Learning", New York: Springer, 2001.

[44] The R Project for Statistical Computing, http://www.r-project.org/.

[45] L Breiman, A Cutler, A Liaw, M Wiener, randomForest, http://cran.r-project.org/web/packages/randomForest/.