J Grid Computing
DOI 10.1007/s10723-015-9348-5

Subdividing Long-Running, Variable-Length Analyses Into Short, Fixed-Length BOINC Workunits

Adam L. Bazinet · Michael P. Cummings

Received: 23 April 2015 / Accepted: 24 August 2015

Abstract We describe a scheme for subdividing long-running, variable-length analyses into short, fixed-length BOINC workunits using phylogenetic analyses as an example. Fixed-length workunits decrease variance in analysis runtime, improve overall system throughput, and make BOINC a more useful resource for analyses that require a relatively fast turnaround time, such as the phylogenetic analyses submitted by users of the GARLI web service at molecularevolution.org. Additionally, we explain why these changes will benefit volunteers who contribute their processing power to BOINC projects, such as the Lattice BOINC Project (boinc.umiacs.umd.edu). Our results, which demonstrate the advantages of relatively short workunits, should be of general interest to anyone who develops and deploys an application on the BOINC platform.

Keywords BOINC · Volunteer computing · Grid computing · Scheduling · Checkpointing · GARLI · Phylogenetics · Maximum likelihood

A. L. Bazinet (✉) · M. P. Cummings
Biomolecular Sciences Building, University of Maryland, College Park, MD 20742-3360, USA
e-mail: adam.bazinet@umiacs.umd.edu

M. P. Cummings
e-mail: mike@umiacs.umd.edu

1 Introduction

Computing resources volunteered by members of the general public can greatly benefit scientific research, as demonstrated by high-profile research projects in disparate areas such as radio astronomy (SETI@home; setiathome.berkeley.edu), climate modeling (climateprediction.net), protein folding (Rosetta@home; boinc.bakerlab.org/rosetta), and particle accelerator physics (LHC@home; lhcathomeclassic.cern.ch/sixtrack), to name just a few. The most widely-used platform for volunteer computing is, by far, the Berkeley Open Infrastructure for Network Computing, or BOINC [1]. Our research group has made BOINC an addressable computational resource in The Lattice Project [2, 3], a grid computing system built on Globus [4] software. In recent years, our grid system development has increasingly focused on improving phylogenetic analysis capability [5]. Our primary phylogenetic inference application is GARLI [6, 7], a popular maximum likelihood-based program. We have recently made a GARLI web service publicly available on molecularevolution.org [8], which executes GARLI analyses on Lattice Project computing resources. The Lattice BOINC Project (boinc.umiacs.umd.edu) is an outstanding resource for running GARLI analyses: a significant proportion of volunteer computers have an appreciable amount of memory, which GARLI analyses often require; and GARLI automatically checkpoints its state when running on BOINC, which allows for efficient use of

Published online: 15 September 2015

the BOINC platform. Indeed, having the capability to run GARLI analyses on BOINC has been critical to the successful completion of several phylogenetic studies [9–11]. However, thus far it has not been feasible to run GARLI web service analyses on BOINC because it has been difficult to guarantee complete results from BOINC in a timely manner. Here we address this problem by subdividing long-running GARLI analyses into short, fixed-length BOINC *workunits* (the term used for a unit of work on the BOINC platform). This speeds up analysis completion by reducing the variance in workunit runtimes, thus making BOINC a more attractive resource for analyses that require a relatively fast turnaround time. The remainder of the paper is organized as follows. In Section 2, we put the problem in context by providing some background on phylogenetic analysis and our computing systems. In Section 3, we provide a more detailed description of the problem and our proposed solution. In Section 4, we describe our implementation of the steps required to subdivide GARLI analyses into fixed-length BOINC workunits. In Sections 5, 6, and 7, we demonstrate the efficacy of our implementation with large-scale tests using the Lattice BOINC Project. Finally, in Section 8 we make some concluding remarks.

2 Background on Phylogenetic Analysis and Computing Systems

A very common analysis type in evolutionary biology, and increasingly in other areas of biology, is the reconstruction of the evolutionary history of organisms (e.g., species) or elements of organisms that have evolutionary or genealogical relationships (e.g., members of gene families, or sampled alleles in a population), sometimes simply called *operational taxonomic units*. This phylogenetic inference problem is especially computationally intensive when based on statistical methods that use parameter-rich models, as is commonly done with maximum likelihood and Bayesian inference. The combination of increasingly sophisticated models and rapidly increasing data set sizes has prompted the development of strategies that speed up analysis execution. Our own work has focused on decreasing time to results through parallelization of maximum likelihood phylogenetic inference, which is more amenable to atomization than Bayesian inference because the many searches that typically comprise an analysis can be performed separately and concurrently. Specifically, we have chosen to deploy an open-source program for maximum likelihood-based phylogenetic inference — GARLI (Genetic Algorithm for Rapid Likelihood Inference) [6, 7] — in a heterogeneous-resource grid computing environment.

As with all phylogenetic inference programs that analyze more than a small number of operational taxonomic units and use an optimality criterion, GARLI employs a heuristic algorithm to solve the simultaneous optimization problem. Specifically, GARLI uses a stochastic evolutionary algorithm to search for the point of maximum likelihood in the multidimensional space consisting of tree topology, branch lengths, and other model parameters, which we simply call the *best tree*. Because of this stochasticity, it is both usual and recommended to perform multiple searches so as to avoid results that represent local optima, seeking instead to obtain results that more nearly reflect the global optimum. Our system assists with this task by dynamically adjusting the number of search replicates performed so as to be reasonably assured of finding the best tree with a high probability [8]. Furthermore, in addition to searches for the best tree, one typically conducts hundreds or thousands of bootstrap replicate searches to assess uncertainty in the best tree topology. Depending on the size and complexity of the analysis, and the capability of the computational resources used, it may take many hours to complete even a single GARLI search replicate. Thus, running many search replicates in parallel on a grid computing system greatly reduces the time required to complete an analysis.

Grid computing is a model of distributed computing that seamlessly links geographically and administratively disparate computational resources, allowing users to access them without having to consider location, operating system, or account administration [12]. The Lattice Project, our grid computing system based on Globus software, incorporates volunteer computers running BOINC, as well as traditional grid computing resources such as Condor pools [13] and compute clusters. The architecture and functionality of the grid system is described extensively elsewhere [14]; fundamentally, however, The Lattice Project provides access to scientific applications (which we call *grid services*), as well as the means to distribute the computation required by these services over thousands of

processing nodes. In recent years, we have enhanced the system by developing a web interface to the GARLI grid service [8], which is currently available at molecularevolution.org. The GARLI grid service has been used in over 65 published phylogenetic studies, with usage having increased dramatically since the release of the GARLI web service in July 2010 [9–11, 15–17] (see lattice.umiacs.umd.edu/publications for the full publication list). As of 18 August 2015, 1,262 GARLI web service users have completed 7,288 analyses comprising well over three million individual search replicates.

As mentioned previously, however, we have not yet been able to use our most novel and potentially most valuable computational resource — our pool of BOINC clients — for processing GARLI web service analyses. The reasons for this are expounded upon in the following section.

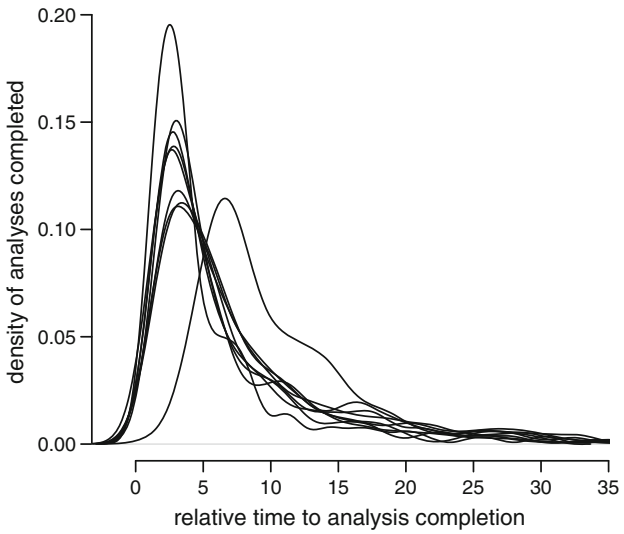

Fig. 1 BOINC analysis batch completion dynamics for nine typical analysis batches as the density of analyses completed by their relative time to completion. All batches exhibit the typical heavy-tail distribution

3 Problem Description and Proposed Solution

Optimal-Length Analyses for Grid Computing There is substantial overhead associated with managing each grid-level analysis submission due to the negotiation of various layers of middleware, latency associated with file transfers, queue wait times, and so on. Thus, *granularity*, the ratio of analysis computation to overhead, is an important consideration. For example, fine-grained tasking, the submission of a large number of short-running analyses, leads to reduced efficiency and reduced overall system throughput, as the majority of each job lifetime is composed of various sources of latency rather than actual scientific computation. Conversely, coarse-grained tasking, the submission of long-running analyses, is not ideal either because longer jobs are more likely than short-running jobs to be interrupted by other processes, computer failures and reboots, and human intervention, which all lead to wasted computation. In the middle of these two undesirable extremes exists a conceptually "optimal" analysis runtime, which maximizes scientific computation by minimizing both unnecessary overhead and potential for interruption. Determining a grid-wide optimal runtime is challenging when one considers the heterogeneity of analyses that are submitted, the heterogeneity of our grid resources, and the complexity of the grid meta-scheduling algorithm. Thus, here we specifically consider only GARLI analyses running on our BOINC resource, as we anticipate that such analyses will benefit greatly from runtime optimization.

Optimal-Length GARLI Analyses for BOINC There are multiple factors that contribute to variance in GARLI analysis runtimes on BOINC. These include factors specific to the BOINC platform, such as variability as to when *result units* (instances of a workunit) are downloaded by volunteer computers, differences in volunteer computer capabilities and reliability, and variation in the computing preferences expressed by volunteers. In addition, the stochastic nature of the GARLI algorithm leads to variable and indeterminate runtimes for individual GARLI search replicates. These and other factors produce analysis batch completion dynamics with a markedly heavy tail (Fig. 1). The later-completing jobs that comprise this heavy tail are sometimes referred to as "stragglers" [18, 19]. By standardizing the length of GARLI workunits, we aim to improve overall analysis batch turnaround time — i.e., reduce the straggler effect — by decreasing the variance in analysis runtimes. The *optimal workunit runtime* maximizes analysis batch throughput (while not taxing system resources, such as storage space, overly much).

Assuming that a near-optimal runtime for GARLI analyses on BOINC can be determined, it is possible to combine multiple short-running GARLI analyses into a

single fixed-length analysis by increasing the number of search replicates (or bootstrap replicates) that a single GARLI invocation performs. This optimization is relatively trivial, and we do not discuss it further here; instead, we focus exclusively on the converse problem of breaking up a single long-running GARLI analysis into multiple short, fixed-length subunits, which GARLI enables by providing a lightweight checkpointing mechanism. If checkpointing is activated, GARLI periodically writes some small text files to disk that contain the information needed to restart a run from that point. This ensures that not much computation is lost if a volunteer computer is rebooted, for example, or if computation is interrupted for some other reason. (Checkpointing is not currently possible during either the initial optimization or final optimization analysis stages of the program, however.) By setting various GARLI parameters appropriately, it is possible to checkpoint the state of a GARLI result unit on a volunteer computer after a fixed length of time has elapsed or amount of computation has been performed (e.g., one hour, or some number of floating point operations, respectively), terminate the analysis, and send the intermediate results back to the BOINC server. The pertinent analysis files and checkpoint files can then be downloaded by another BOINC compute node (simply termed a *host*) to resume computation where it left off, again for the same fixed length of time or amount of computation. Though this type of scheme incurs some additional overhead in terms of required data movement and storage, communication, and record keeping, we expect that these costs will be outweighed by the performance gains, which are potentially substantial and important in several ways.

Benefits of Fixed-Length Analyses The performance gains that result from the standardization of GARLI analysis runtime on the BOINC platform are realized both for BOINC volunteers, as well as researchers who use the GARLI web service.

BOINC volunteers tend to prefer uniform-length workunits, an expectation derived from participation in BOINC projects that have a practically unlimited supply of homogeneous workunits (e.g., SETI@home; setiathome.berkeley.edu). Hence, dividing variable-length, long-running analyses into short, fixed-length workunits better meets the expectations of BOINC volunteers and increases their enthusiasm about running GARLI analyses, which in turn leads to greater volunteer participation and retention. Furthermore, long-running analyses of unknown runtime create many opportunities for failure and interruption, as well as uncertainty and anxiety about when the analyses will finish, all of which causes some BOINC volunteers to abort such analyses prematurely. Thus, by shortening and standardizing the length of GARLI workunits, we make our system much more appealing to volunteers. Finally, standard-length workunits afford the opportunity to grant a fixed amount of credit per workunit, an inherently fair procedure that volunteers tend to favor.

For researchers using the GARLI web service, GARLI analysis runtime optimization provides performance benefits as well. As already mentioned, subdividing long-running analyses into shorter-length workunits increases reliability by decreasing the probability of premature workunit termination. It also provides a natural load-balancing mechanism by affording the most capable BOINC hosts more opportunities to process more workunits, thus lightening the tail in Fig. 1 by shifting the distribution toward shorter completion times. This decrease in the variance of completion times should result in increased overall system throughput and decreased time to results for GARLI web service users.

Related Work There has been some research into optimizing BOINC scheduling policies [20–25], often with the assistance of simulated grids [26]. However, these studies attempt to solve a more general scheduling problem than we do here, and thus model many different factors: heterogeneity in host capabilities and computing preferences, variation in workunit properties and deadlines, requirements of multiple simultaneously connected BOINC projects, and so on. One such study [23] specifically focused on optimizing scheduling policies for "medium-grained" tasks (tasks that take minutes or hours), which is relevant to our present work because we target tasks of this length. Although we do not change or optimize any BOINC scheduling policies ourselves, we do benefit from any such optimizations that already exist, especially ones targeted at relatively short tasks. Complementary strategies have also been described, such as one that uses a cloud of dedicated resources to process the small fraction of tasks that do not complete on a volunteer grid in a reasonable amount of time [27]. Here we take the current

BOINC scheduling policies as a given and demonstrate that reducing workunit runtimes leads to faster turnaround time for analysis batches.

4 Implementation of Fixed-Length GARLI Workunits

To implement this scheme, we divide each GARLI analysis into at least three workunits: the *initial* workunit, which performs the initial optimization phase of the analysis; 1 to *n main* workunits, which perform the bulk of the search; and the *final* workunit, which performs the final optimization phase of the analysis. As mentioned previously, checkpointing is not available during the initial or final optimization phases, so we are unable to precisely control the runtime of the initial or final workunits. However, these program phases are typically short, and do not account for more than 10 % of the overall program execution time. The main workunits, on the other hand, comprise the majority of the runtime, and their maximum execution time can be precisely controlled.

To divide program execution into phases, a `workphasedivision` option was added to GARLI version 2.1. When `workphasedivision=1`, GARLI automatically checkpoints and terminates immediately after initial optimization is complete, and immediately before final optimization begins. Additionally, the `stoptime` parameter, which is a positive number of seconds after which an analysis should be terminated, was redefined in GARLI version 2.1 to be relative to the time an analysis was most recently restarted, as opposed to its very beginning. Thus, by setting `stoptime=3600`, for example, one may cause a GARLI main workunit to terminate after one hour of runtime. (Note: `stoptime` is ignored during the initial and final optimization phases.)

GARLI Checkpoint Files and BOINC Homogeneous Redundancy Unfortunately, GARLI checkpoint files are not portable between operating systems, and may not even be portable between 32 bit and 64 bit variants of the same operating system. This presents a major implementation obstacle, as one may not simply mix and match execution hosts indiscriminately. To deal with this issue, we made use of a BOINC feature called *homogeneous redundancy* (HR), which was originally developed to ensure that multiple instances of the same workunit (termed *result units*) would run on the same "class" of host. This guaranteed that the numerical output from multiple result units would match exactly, which was required to use a voting scheme to verify that results were computed correctly. Depending on how a particular application was compiled and what computations it performed, host classes could be more or less broadly defined. Maximally inclusive host classes are desirable because having more hosts available to run any particular workunit improves overall system throughput. BOINC currently defines two HR types: a *coarse-grained* type in which there are four host classes (Windows, Linux, Mac-PowerPC, and Mac-Intel), and a *fine-grained* type in which there are 80 host classes (four operating system and 20 CPU types). For our testing, we enabled coarse-grained HR for GARLI, along with a BOINC feature called *homogeneous app version* (HAV) that ensured consistent use of either the 32 bit or 64 bit version of GARLI. These settings did not completely eliminate errors related to checkpoint portability, but allowed testing to proceed with a sufficiently low error rate (less than 2 %).

With normal use of HR, each BOINC workunit may have a different HR class; it is the various result units associated with a particular workunit that must have the same HR class. Thus, the HR class for a given workunit is not usually determined until its first result unit is assigned to a particular host. In our scheme, the main and final workunits associated with a particular GARLI analysis must have the same HR class as that of the initial workunit, so we needed to set the HR class of the main and final workunits at the time of their creation. To accomplish this, we used a new argument to the BOINC `create_work` program. In addition, we added the `hr_class_static` tag to the BOINC configuration file, which suppresses the mechanism that clears the HR class of a workunit if a result unit fails when there are no other result units for that workunit in progress or already completed.

Modifications to Grid System Components The implementation of this scheme was relatively complex and involved changes to several different grid system components. The component that was the most heavily modified was the BOINC job manager, the Perl

script responsible for transforming a generic Globus job description into appropriate BOINC workunits [2]. The BOINC job manager checks the GARLI configuration file for `workphasedivision=1`; upon finding it, the script creates the initial workunit and writes three separate workunit templates and assimilator scripts for the analysis, one for each workunit type (initial, main, and final). The workunit templates specify the input and output files for each workunit, which vary depending on the workunit type; they also specify that input and output files associated with initial and main workunits are not allowed to be deleted immediately after such workunits complete, unlike files associated with regular GARLI workunits. The appropriate assimilator script is invoked when a workunit of a particular type completes successfully. The *initial* assimilator script sets `restart=1` in the GARLI configuration file, which causes the main and final workunits to restart from checkpoint files. It also moves the checkpoint files and the standard output (associated with the canonical result of the initial workunit) from the BOINC upload directory to the download directory, so these can be used as additional input files to the first main workunit. Finally, the initial assimilator script creates the first main workunit using the correct templates and other parameters, and sets its HR class to that of the initial workunit. The *main* assimilator script parses the GARLI log file to determine if the analysis is ready for final optimization; if so, it creates the final workunit; if not, it creates the next main workunit. The *final* assimilator script copies the final output files to the location where Globus expects them, removes all intermediate output files that may be resident on disk from associated initial or main workunits, and updates the BOINC database.

Numerous changes were made to other grid system components as well; a few examples follow. The BOINC scheduler event generator (SEG), a Globus component that periodically queries the BOINC database for the status of jobs [14], was modified to include final workunits in its queries, but to exclude initial and main workunits from such queries. The BOINC validator, a daemon that verifies that GARLI results returned by BOINC clients include a valid tree file [14], was modified to ignore results from initial or main workunits. The BOINC assimilator, a daemon that processes successfully completed workunits [14], was modified so that the number of the main workunit was passed to our custom assimilator scripts, among other minor changes.

Although not discussed here in detail, we made additional modifications to support *analysis batches*, which allowed multiple initial workunits to be created simultaneously and to be associated with one another as a batch of analyses. Each initial workunit still generates its own main and final workunits that are tracked and updated independently of those associated with other initial workunits in the batch. This functionality allowed us to quickly and easily submit batches of thousands of workunits, which was the order of magnitude required to properly evaluate the performance of this scheme.

The development described up to this point was sufficient to enable large-scale testing of the

Table 1 Attributes of large-scale BOINC tests of fixed-length vs. full-length GARLI workunits

Date of test	Test type	Result units per WU	Hosts granted credit[a,b]	Hosts reporting[a,b]	Result units in progress[a]
14 Jan 2015	fixed-length	one	$\sim 1,300$	3,813	37
21 Jan 2015	full-length	one	$\sim 1,325$	3,755	26
31 Jan 2015	fixed-length	one	$\sim 1,300$	3,715	165
02 Feb 2015	full-length	one	$\sim 1,400$	3,724	123
05 Feb 2015	fixed-length	two	$\sim 1,350$	3,698	331
08 Feb 2015	full-length	two	$\sim 1,420$	3,732	145
13 Feb 2015	fixed-length	two	$\sim 1,430$	3,724	172
15 Feb 2015	full-length	two	$\sim 1,480$	3,695	134

[a]Conditions at the time of submission

[b]Tallied over the previous 30 days

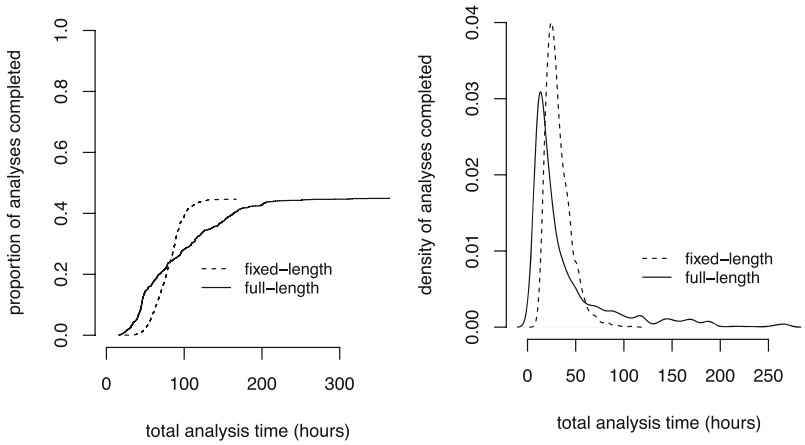

Fig. 2 Total analysis time, in hours, for fixed-length and full-length analysis batches that used one result unit per workunit. A cumulative distribution plot gives the proportion of analyses completed by total analysis time, and a density plot gives the density of analyses completed by total analysis time. Each *line* shown is derived from a series of at most 2,000 points (1,000 from each test replication), where each point represents an individual GARLI analysis

fixed-length workunit paradigm, and to compare it to the normal, "full-length" paradigm in which a single BOINC workunit executes an entire GARLI analysis from start to finish. We describe this testing in the following sections.

5 Fixed-Length vs. Full-Length GARLI Workunit Tests

We decided that a comparison of the new "fixed-length workunit" paradigm to the standard "full-length workunit" paradigm was best accomplished with large-scale BOINC testing — i.e., we would assess runtimes and other performance characteristics using thousands of analyses, which would exercise the BOINC client pool in a realistic manner. For these tests, we analyzed an 82-taxon, 13-mitochondrial-gene data set with GARLI using a codon model. If uninterrupted, the runtime of this analysis on an average computer was approximately 10 to 15 hours, and the 1024 MB memory requirement was low enough that the majority of clients could participate. We used the following GARLI settings: `randseed=42`; `availablemem=1024`; and `stopgen=30000`. Additionally, for fixed-length analyses, we set `stoptime=3600`, which caused main workunits to terminate after one hour. BOINC

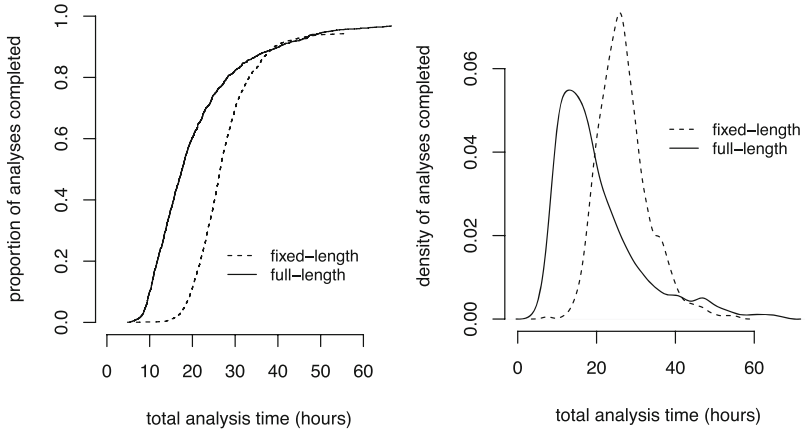

Fig. 3 Total analysis time, in hours, for fixed-length and full-length analysis batches that used two result units per workunit. A cumulative distribution plot gives the proportion of analyses completed by total analysis time, and a density plot gives the density of analyses completed by total analysis time. Each *line* shown is derived from a series of at most 2,000 points (1,000 from each test replication), where each point represents an individual GARLI analysis

Table 2 Results of large-scale BOINC tests of fixed-length vs. full-length GARLI workunits

Date of test	Test type	Result units per WU	Analyses included in results	Mean analysis time (hr)	Median analysis time (hr)	Standard deviation (hr)
14 Jan 2015	fixed-length	one	966	37.0	35.2	14.3
21 Jan 2015	full-length	one	962	33.4	24.2	24.6
31 Jan 2015	fixed-length	one	968	26.1	25.5	5.5
02 Feb 2015	full-length	one	959	21.1	17.8	11.5
05 Feb 2015	fixed-length	two	978	27.1	25.2	9.5
08 Feb 2015	full-length	two	994	26.7	18.0	21.7
13 Feb 2015	fixed-length	two	920	28.0	26.8	7.4
15 Feb 2015	full-length	two	977	19.4	16.7	10.0

workunit wall clock deadlines were set to two days for initial and final workunits, and six hours for main workunits. Full-length workunits were given a deadline of one week, which roughly corresponded to the total wall clock time allocated to fixed-length workunit series (two days for the initial workunit, plus six hours for each of approximately 12 main workunits, plus two days for the final workunit). Each test began with the submission of 1,000 workunits (either 1,000 initial workunits or 1,000 full-length workunits). Other test attributes, including the date of the test, the number of result units per workunit, and the status of the hosts in the BOINC pool at the time of submission are given in Table 1.

The purpose of this series of tests was, first and foremost, to compare the performance of series of fixed-length workunits to standard, full-length workunits. Secondarily, we also sought to measure the effect of using task replication, a strategy that has been previously shown to be effective in improving grid performance [28]. Performance advantages of task replication result from two associated properties: i) increase in reliability as each additional replicate decreases the probability of no result returning; and ii) shorter time to completion because the first completed result among replicates can be used immediately. A cost of task replication is the waste associated with not using later-arriving, but valid, results. Here we examine the minimum possible task replication of two result units per workunit. For each combination of fixed-length or full-length, and one result unit or two result units, we performed two large-scale tests to increase the overall precision of our assessment; this totaled eight tests (Table 1).

For evaluation purposes, we measured *total analysis time* as follows. For a fixed-length workunit series,

Table 3 Attributes of large-scale BOINC tests to determine optimal-length GARLI workunits

Date of test	Main WU length (minutes)	Main WU deadline (minutes)	Hosts granted credit[a,b]	Hosts reporting[a,b]	Result units in progress[a]
19 Feb 2015	30	180	$\sim 1,460$	3,687	233
22 Feb 2015	60	360	$\sim 1,520$	3,695	133
24 Feb 2015	120	720	$\sim 1,580$	3,707	123
26 Feb 2015	240	1,440	$\sim 1,545$	3,721	123
02 Mar 2015	240	1,440	$\sim 1,620$	3,776	24
05 Mar 2015	120	720	$\sim 1,630$	3,771	69
07 Mar 2015	60	360	$\sim 1,630$	3,772	45
10 Mar 2015	30	180	$\sim 1,635$	3,794	76

[a]Conditions at the time of submission

[b]Tallied over the previous 30 days

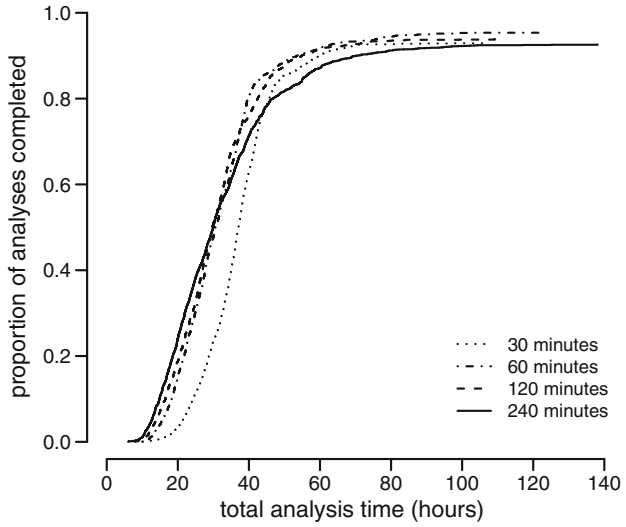

Fig. 4 Total analysis time, in hours, for analysis batches of different main workunit lengths. A cumulative distribution plot gives the proportion of analyses completed by total analysis time. Each line shown is derived from a series of at most 2,000 points (1,000 from each test replication), where each point represents an individual GARLI analysis

total analysis time was measured as the time interval beginning when the initial workunit was created, and ending when valid results from the final workunit were returned to our BOINC server. For a full-length workunit, total analysis time was measured simply as the time interval beginning when the workunit was created, and ending when valid results from the workunit were returned to our BOINC server. In Figs. 2 and 3, we compare the total analysis time of fixed-length and full-length analysis batches; Fig. 2 includes the tests that used one result unit per workunit, and Fig. 3 includes the tests that used two result units per workunit.

The one-result unit comparison (Fig. 2) shows the general pattern that we expected to observe: the variance in total analysis time is lower in the fixed-length workunit scheme. Thus, while the fixed-length scheme takes longer to complete $\sim$ 70 % of the analyses, it completes all of its analyses $\sim 2.3\times$ more quickly than the equivalent number of full-length analyses.

The effect of task replication, in this case doubling the number of result units per workunit (Fig. 3), is also apparent: the analysis batches complete more quickly, as faster hosts in the pool are given the opportunity to process more work. The effect is greatest for the full-length analysis batches, which complete $\sim 3.3\times$ more quickly in the two-result unit tests. Comparing the fixed-length scheme to the full-length scheme in the two-result unit case, however, we observe a performance pattern that is similar to the one-result unit case, as the performance of the fixed-length scheme begins to equal or outperform the full-length scheme at a large proportion of analyses completed.

Thus, here we demonstrate two ways of improving performance: i) using a series of fixed-length workunits instead of a single full-length workunit, which incurs no additional cost in terms of BOINC client resources; and ii) using task replication by doubling the number of result units per workunit, which incurs twice the cost in BOINC client resources. Supporting summary statistics for these tests are given in Table 2.

6 Optimal-Length GARLI Workunit Tests

The next round of large-scale tests was intended to approximately determine an efficient length for GARLI main workunits. For these tests, all of which

Table 4 Results of large-scale BOINC tests to determine optimal-length GARLI workunits

Date of test	Main WU length (minutes)	Main WU deadline (minutes)	Analyses included in results	Mean analysis time (hr)	Median analysis time (hr)	Standard deviation (hr)
19 Feb 2015	30	180	896	35.3	34.8	10.2
22 Feb 2015	60	360	945	29.6	28.1	11.6
24 Feb 2015	120	720	898	30.6	29.9	12.3
26 Feb 2015	240	1,440	888	30.3	25.9	16.3
02 Mar 2015	240	1,440	964	32.5	30.4	16.1
05 Mar 2015	120	720	979	30.3	28.1	12.6
07 Mar 2015	60	360	963	33.7	32.1	13.8
10 Mar 2015	30	180	963	38.0	38.1	10.3

Table 5 Attributes of final large-scale tests of fixed-length vs. full-length GARLI workunits

Date of test	Test type	Result units per WU	Hosts granted credit[a,b]	Hosts reporting[a,b]	Result units in progress[a]
23 Mar 2015	full-length	one	$\sim 1,610$	3,865	3
31 Mar 2015	fixed-length	one	$\sim 1,480$	3,875	270

[a]Conditions at the time of submission

[b]Tallied over the previous 30 days

were fixed-length, we used the same 82-taxon, 13-mitochondrial-gene data set as before, together with the same GARLI settings, except that we varied `stoptime` so as to test main workunit lengths of 30 minutes, 60 minutes, 120 minutes, and 240 minutes. As before, the wall clock deadline for initial and final workunits was set to two days; for main workunits, the deadline was scaled proportionally to the main workunit length (Table 3). Each test began with the submission of 1,000 initial workunits. Other test attributes, including the date of the test, the main workunit length and wall clock deadline, and the status of the hosts in the BOINC pool at the time of submission are given in Table 3. For each main workunit length evaluated, we performed two large-scale tests to increase the overall precision of our assessment; this totaled eight tests (Table 3).

For our evaluation, we measured total analysis time as in our previous round of testing. Figure 4 compares total analysis time of analysis batches of varying main workunit lengths.

We observe, in general, that varying the main workunit length does not impact the performance characteristics of analysis batches especially greatly, at least at the main workunit lengths we tested. As before, we observe less variance in analysis time with shorter main workunit lengths. Once we go as low as 30 minutes, however, we notice some deleterious effects of overhead associated with generating the increased number of workunits and input files that are required. Indeed, each successive halving of main workunit runtime incurs twice as much file system and database storage cost, and doubles the processing load on our servers. Thus, as the 60-minute and 120-minute runtimes performed comparably, we would choose a main workunit runtime of 120 minutes (two hours) to minimize overhead costs. A two-hour runtime is certainly in keeping with our *a priori* expectation of a reasonable main workunit length, an expectation based on many years of interaction with our BOINC volunteers. Supporting summary statistics for these tests are given in Table 4.

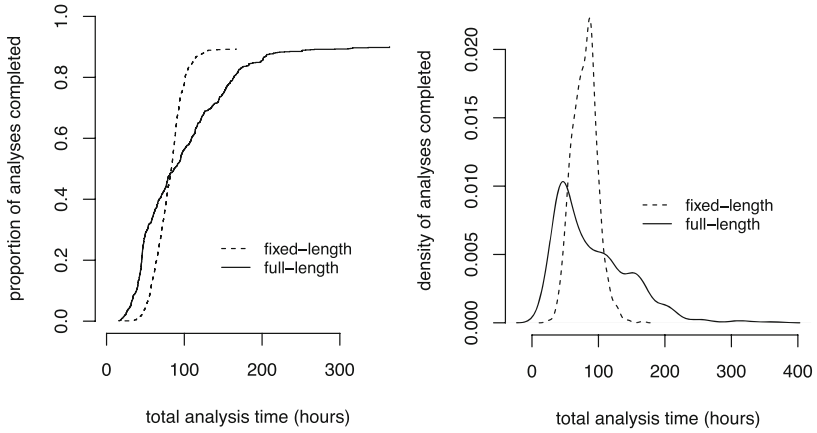

Fig. 5 Total analysis time, in hours, for fixed-length and full-length analysis batches. A cumulative distribution plot gives the proportion of analyses completed by total analysis time, and a density plot gives the density of analyses completed by total analysis time. Each *line* shown is derived from a series of at most 1,000 points, where each point represents an individual GARLI analysis

Table 6 Results of final large-scale tests of fixed-length vs. full-length GARLI workunits

Date of test	Test type	Result units per WU	Analyses included in results	Mean analysis time (hr)	Median analysis time (hr)	Standard deviation (hr)
23 Mar 2015	full-length	one	900	91.9	77.3	56.1
31 Mar 2015	fixed-length	one	893	79.4	80.1	18.6

7 Final Fixed-Length vs. Full-Length GARLI Workunit Test

The final round of large-scale tests was intended to measure the performance of the fixed-length scheme against the full-length scheme with a longer GARLI analysis. For these tests, we used the same 82-taxon, 13-mitochondrial-gene data set as before, along with the same GARLI settings, except that we set `stopgen=60000` and `enforcetermconditions=0`, which together roughly doubled the length of the analysis. For the fixed-length test, we set `stoptime=7200`, which was the best-performing main workunit runtime determined from the previous round of tests. BOINC wall clock deadlines were set to two days for initial and final workunits, and 12 hours for main workunits. Full-length workunits were given a deadline of one week. Each test began with the submission of 1,000 workunits (either 1,000 initial workunits or 1,000 full-length workunits). Other test attributes, including the date of the test, the number of result units per workunit, and the status of the hosts in the BOINC pool at the time of submission are given in Table 5.

For our evaluation, we measured total analysis time the same way as in our previous rounds of testing. Figure 5 compares the total analysis time of fixed-length and full-length analysis batches.

We observe the same pattern that we did in previous tests: the variance in total analysis time is significantly reduced in the fixed-length workunit test. Thus, while the fixed-length scheme takes longer to complete $\sim$ 50 % of the analyses, it completes all of its analyses $\sim 1.8\times$ more quickly than the equivalent number of full-length analyses. Thus, we note that the relative performance of the fixed-length paradigm improves as overall analysis length increases. Supporting summary statistics for these tests are given in Table 6.

8 Conclusion

As the preceding test results demonstrate, the reduction in analysis time variance achieved by subdividing long-running GARLI analyses into short, fixed-length BOINC workunits results in faster completion times for analysis batches. Furthermore, taking a number of factors into consideration, we arrived at a best-performing main workunit length of two hours. We also demonstrated how the relative performance of the fixed-length workunit scheme improves as overall analysis length increases. Although with a highly heterogeneous pool of consumer-grade computers there will always be some degree of variance in analysis completion times, our results suggest that the heavy tail on analysis batches (i.e., the straggler effect; Fig. 1) can be substantially reduced by subdividing analyses into short workunits. Furthermore, there is a marked performance improvement with $2\times$ task replication (Table 2; Fig. 2 cf. Fig. 3), although it requires twice as much computation. We would expect these results to generalize to other BOINC applications as well. Thus, other BOINC projects, even those whose applications checkpoint, may be motivated by these results to shorten their workunits. In our case, we are optimistic that this reduction in runtime variance, along with strategies such as submitting more than the required number of analyses and using the first results that become available, will make BOINC a viable and effective resource for processing GARLI web service analyses.

Acknowledgments We thank Derrick Zwickl for adding necessary features to GARLI that enabled this work. Similarly, we thank David Anderson for adding new features to BOINC that aided with testing. This work was funded by National Science Foundation award DBI-1356562.

References

1. Anderson, D.P.: BOINC: a system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, GRID '04, pp. 4–10. USA, Washington, DC (2004)
2. Myers, D.S., Bazinet, A.L., Cummings, M.P.: Expanding the reach of Grid computing: combining Globus- and BOINC-based systems. In: Talbi, E.-G., Zomaya, A.Y. (eds.) Grids for Bioinformatics and Computational Biology, Wiley Book Series on Bioinformatics: Computational Techniques and Engineering, Chapter 4, pp. 71–85. Wiley-Interscience, Hoboken (2008)
3. Bazinet, A.L., Cummings, M.P.: 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, Chapter 1, pp. 2–13. Rechenkraft.net, Marburg (2008)
4. Foster, I., Kesselman, C.: Globus: a toolkit-based grid architecture. In: Foster, I., Kesselman, C. (eds.) The Grid: Blueprint for a New Computing Infrastructure, pp. 259–278. Morgan-Kaufmann (1999)
5. Bazinet, A.L., Cummings, M.P.: Computing the tree of life: Leveraging the power of desktop and service grids, pp. 1896–1902. IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (2011)
6. Zwickl, D.J.: GARLI 2.0 https://www.nescent.org/wg_garli/main_page (2011)
7. Zwickl, D.J.: Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion. PhD thesis. The University of Texas at Austin (2006)
8. Bazinet, A.L., Zwickl, D.J., Cummings, M.P.: A gateway for phylogenetic analysis powered by grid computing featuring GARLI 2.0. Syst. Biol. **63**(5), 812–8 (2014)
9. Regier, J.C., Zwick, A., Cummings, M.P., Kawahara, A.Y., Cho, S., Weller, S., Roe, A., Baixeras, J., Brown, J.W., Parr, C., Davis, D.R., Epstein, M., Hallwachs, W., Hausmann, A., Janzen, D.H., Kitching, I.J., Solis, M.A., Yen, S.-H., Bazinet, A.L., Mitter, C.: Toward reconstructing the evolution of advanced moths and butterflies (Lepidoptera: Ditrysia): an initial molecular study. BMC Evol. Biol. **9**, 280 (2009)
10. Bazinet, A.L., Cummings, M.P., Mitter, K.T., Mitter, C.W.: Can RNA-Seq resolve the rapid radiation of advanced moths and butterflies (Hexapoda: Lepidoptera: Apoditrysia)? An exploratory study. PLoS ONE **8**(12), e82615 (2013)
11. Regier, J.C., Mitter, C., Zwick, A., Bazinet, A.L., Cummings, M.P., Kawahara, A.Y., Sohn, J.-C., Zwickl, D.J., Cho, S., Davis, D.R., Baixeras, J., Brown, J., Parr, C., Weller, S., Lees, D.C., Mitter, K.T.: A large-scale, higher-level, molecular phylogenetic study of the insect order Lepidoptera (moths and butterflies). PLoS ONE **8**(3), e58568 (2013)
12. Cummings, M.P., JC Huskamp, J.C.: Grid computing. EDUCAUSE Review **40**, 116–117 (2005)
13. Litzkow, M.J., Livny, M., Mutka, M.W.: Condor–a hunter of idle workstations. In: 8th International Conference on Distributed Computing Systems, pp. 104–111 (1988)
14. Bazinet, A.L.: The Lattice Project: A multi-model grid computing system. Master's thesis. University of Maryland, College Park (2009)
15. Myers, D.S., Cummings, M.P.: Necessity is the mother of invention: a simple grid computing system using commodity tools. J. Parallel Distr. Com. **63**(5), 578–589 (2003)
16. Kawahara, A.Y., Ohshima, I., Kawakita, A., Regier, J.C., Mitter, C., Cummings, M.P., Davis, D.R., Wagner, D.L., De Prins, J., Lopez-Vaamonde, C.: Increased gene sampling provides stronger support for higher-level groups within moths, gracillariid leaf mining relatives (Lepidoptera, Gracillariidae). BMC Evol. Biol. **11**, 182 (2011)
17. Sohn, J.-C., Regier, J.C., Mitter, C., Davis, D., Landry, J.-F., Zwick, A., Cummings, M.P.: A molecular phylogeny for Yponomeutoidea (Insecta, Lepidoptera, Ditrysia) and its implications for classification, biogeography and the evolution of host plant use. PLoS ONE **8**(1), e55066 (2013)
18. Huang, S.W., Huang, T.-C., Lyu, S.-R., Shieh, C.-K., Chou, Y.-S.: Improving speculative execution performance with coworker for cloud computing. In: IEEE 17th International Conference Parallel and Distributed Systems (ICPADS), pp. 1004–1009 (2011)
19. Ananthanarayanan, G., Ghodsi, A., Shenker, S., Stoica, I.: Effective straggler mitigation: attack of the clones. In: USENIX Symposium on Networked Systems Design and Implementation, vol. 13, pp. 185–198 (2013)
20. Anderson, D.P.: Emulating volunteer computing scheduling policies. In: IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), pp. 1839–1846 (2011)
21. Kondo, D., Anderson, D.P., McLeod, J.: Performance evaluation of scheduling policies for volunteer computing. In: IEEE International Conference on e-Science and Grid Computing, pp. 415–422 (2007)
22. Estrada, T., Flores, D.A., Taufer, M., Teller, P.J., Kerstens, A., Anderson, D.P.: The effectiveness of threshold-based scheduling policies in BOINC projects. In: Second IEEE International Conference on e-Science and Grid Computing, 2006. e-Science '06, pp. 88–88 (2006)
23. Heien, E.M., Fujimoto, N., Hagihara, K.: Computing low latency batches with unreliable workers in volunteer computing environments. In: IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008, pp. 1–8 (2008)
24. Toth, D., Finkel, D.: Improving the productivity of volunteer computing by using the most effective task retrieval policies. J. Grid Comput. **7**(4), 519–535 (2009)

25. Rood, B., Lewis, M.J.: Grid resource availability prediction-based scheduling and task replication. J. Grid Comput. **7**(4), 479–500 (2009)
26. Estrada, T., Taufer, M., Anderson, D.P.: Performance prediction and analysis of BOINC projects: an empirical study with EmBOINC. J. Grid Comput **7**(4), 537–554 (2009)
27. Kovács, J., Marosi, A.-C., Visegrádi, A., Farkas, Z., Kacsuk, P., Lovas, R.: Boosting gLite with cloud augmented volunteer computing. Futur. Gener. Comput. Syst. **43–44**, 12–23 (2015)
28. Kondo, D., Chien, A.A., Casanova, H.: Scheduling task parallel applications for rapid turnaround on enterprise desktop grids. J. Grid Comput. **5**(4), 379–405 (2007)