

# RIDGE: Combining Reliability and Performance in Open Grid Platforms\*

Krishnaveni Budati, Abhishek Chandra and Jon Weissman  
Department of Computer Science and Engineering  
University of Minnesota - Twin Cities  
{budati, chandra, jon}@cs.umn.edu

## Abstract

*Large-scale donation-based distributed infrastructures need to cope with the inherent unreliability of participant nodes. A widely-used work scheduling technique in such environments is to redundantly schedule the outsourced computations to a number of nodes. We present the design and implementation of RIDGE, a reliability-aware system which uses a node's prior performance and behavior to make more effective scheduling decisions. We have implemented RIDGE on top of the BOINC distributed computing infrastructure and have evaluated its performance on a live testbed consisting of 120 PlanetLab nodes. Our experimental results show that RIDGE is able to match or surpass the throughput of the best vanilla BOINC configuration under different reliability environments, by automatically adapting to the characteristics of the underlying environment. In addition, RIDGE is able to provide much lower workunit makespans compared to BOINC, which indicates its desirability in service-oriented environments with time constraints.*

## 1 Introduction

Voluntary distributed computing infrastructures have been an active area of research in the past few years. SETI@home [2] was one of the early projects that generated enthusiasm for this paradigm. Today, these infrastructures are being used in a diverse set of application domains such as bioinformatics [9], physics [15], and environment science [7]. BOINC [1] is a generalization of these projects that provides a computing infrastructure for utilizing donated resources. BOINC has a centralized server which distributes tasks to participating worker nodes and collects the results returned by these workers.

A key problem faced by such projects is the inherent unreliability of the infrastructure as nodes have dynamically

changing workloads, may leave and join unexpectedly, or may behave maliciously. BOINC applications (like SETI) use redundant task execution coupled with voting on results to address such unreliability. A major drawback of the BOINC approach is that it relies on the application designer to set the redundancy factor for task replication. Since an application designer has little idea about the actual network dynamics, determining this value is largely guesswork at best. However, the value of replication factor can have a severe impact on the performance of the application: a small value could lead to large number of failed computations, while a large value could lead to wasted resources and reduced throughput. Moreover, the desired value of the replication factor is critically dependent on the reliability of the underlying system, and hence difficult to determine a priori.

This paper presents the design and implementation of RIDGE (*Reliable Infrastructure for Donation-based Grid Environments*), a system designed to combine reliability and performance of the underlying infrastructure. RIDGE achieves this combination by automatically adjusting the degree of replication based on prior node behavior. In an earlier paper, we showed how such intelligent replication and scheduling can improve performance through a simulation study [19]. This paper focuses on the implementation and deployment of the proposed ideas in a live environment, and explores several corresponding systems issues. We have implemented RIDGE by modifying the core BOINC infrastructure to use a different workload allocation strategy. RIDGE makes more informed decisions by observing the past behavior and estimating a reliability rating for each of the worker nodes in the system. The reliability rating of a worker node is characterized to encapsulate the node's behavior in two dimensions: its correctness and timeliness in returning results. The primary motivation for exploring correctness and timeliness together is to provision RIDGE to support service-oriented environments, where getting a timely result is as important as getting a correct result.

We have deployed a prototype of RIDGE and evaluated it on a live distributed testbed consisting of 120 PlanetLab [3] nodes, using the BLAST [4] bioinformatics application. We

---

\*Note for the reviewer: The paper is less than 10 pages of text with about 3 pages of figures and references.

compare the performance of RIDGE to the default BOINC infrastructure running on the same platform, using a combination of synthetic as well as real reliability distributions to emulate node behaviors. Our experimental results show that RIDGE is able to match or surpass the throughput of the best vanilla BOINC configuration under different reliability environments, by automatically adapting to the characteristics of the underlying environment. In addition, RIDGE is able to provide much lower workunit makespans compared to BOINC, which indicates its desirability in service-oriented environments with time constraints.

## 2 System Architecture

RIDGE is implemented on top of the core BOINC architecture, and it utilizes BOINC mechanisms for workload creation, communication with worker nodes, result gathering, etc. We first briefly describe the core BOINC architecture, followed by the RIDGE enhancements and workload allocation strategy.

### 2.1 BOINC Architecture and Work Allocation Policy

The BOINC architecture consists of a centralized server responsible for distributing work to the worker nodes. Each unit of computation (referred to as a “*workunit*”) is replicated into a fixed number of replicas (referred to as “*tasks*”). The replication factor is a static value specified by the application writer. BOINC employs a pull-based work distribution model where the worker nodes query the server for work and are assigned tasks by the server. Results are returned by the workers to the server upon completion of each task execution, and are verified using a verification technique specified by the application designer. M-majority voting and M-first voting are the most common verification techniques used. With M-majority voting, each workunit is replicated into at least  $2M-1$  tasks and the workunit is said to have completed successfully if a minimum of  $M$  out of the  $2M-1$  results match. In M-first voting, each workunit is replicated into at least  $M$  tasks and a workunit is said to have completed successfully as soon as  $M$  results match. The process of voting and verifying each result of the workunit is called *validating* a workunit. If a validation is successful, the workunit is deemed to be completed and any of its tasks that are yet to be scheduled are purged. However, if a validation fails, additional tasks are generated for the workunit in an incremental fashion until the validation is successful. A key limitation of BOINC work assignment policy is that a static replication factor is used for all workunits and the assignment of tasks to worker nodes is arbitrary.

### 2.2 RIDGE Scheduling Framework

RIDGE replaces the default BOINC workload allocation policy with a *Reputation-based scheduling* technique [19]. The idea behind this technique is to collect reliability ratings of individual worker nodes and use this information to group them together more intelligently and thus increase throughput while meeting the desired success-rate<sup>1</sup>. The basic idea is that a node’s reliability rating is based on the number of ‘timely’ and ‘correct’ task executions performed in the past relative to the total number of tasks allocated to it. Using these values, it is possible to determine effective redundancy groups, both in size and in worker composition. More details of this technique can be found in [19]. While the original algorithms are designed for M-majority voting, we have extended them to work for M-first voting in this paper. The RIDGE server employs these scheduling algorithms and is driven by the following key parameters:

- *Target Success-Rate*: It is defined as the minimal success-rate desired from the system and is specified as a value in the range 0-1. The scheduling algorithms form redundancy groups such that each group formed is expected to return a valid result with a probability at least equal to the Target Success-Rate.
- *Execution-Threshold*: It is defined as the maximum time that a task execution is allowed to take for it to be considered ‘timely’. The execution time is calculated as the difference between the time when the result for a task is received at the server and the time when the task was dispatched to the worker.
- *Scheduling-Threshold*: The number of workers for which the RIDGE scheduler (described below) waits for before running the scheduling algorithm. In this paper, we use a threshold of 1 to enable a fair comparison of RIDGE to vanilla BOINC.
- *MinClients*: The minimum number of workers that a workunit should be scheduled to. This value could be dependent on the verification technique used or could be as desired by the application designer. For example, for M-first voting with  $M = 2$ , *MinClients* should be at least 2.
- *MaxClients*: The maximum number of workers that a workunit could be scheduled to. This is defined to prevent RIDGE from forming groups of arbitrarily large sizes in order to meet the *Target Success-Rate*.

There is a tradeoff between the desired success-rate and throughput. The higher the replication factor, the greater

<sup>1</sup>Intuitively, ‘Success-Rate’ measures the proportion of instances that a workunit was successfully completed without the need for Re-scheduling.

is the success-rate achieved, while there may be a drop in throughput. Thus, a designer needs to carefully set this value based on their desired success-rate and throughput. In previous work [18], we have developed an adaptive algorithm that allows the designer to weigh the importance of success-rate vs. throughput. A detailed discussion of this tradeoff is beyond the scope of this paper, and here, we assume an externally specified Target success-rate.

### 2.2.1 Component Architecture

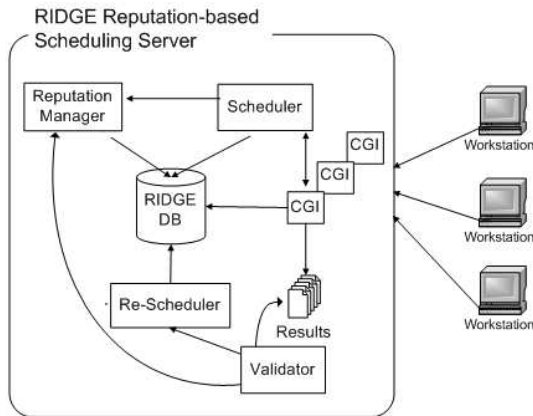


Figure 1: The RIDGE scheduling framework

We now describe the implementation of RIDGE that employs the Reputation-based scheduling algorithms. Figure 1 shows the RIDGE workload distribution engine. The primary components of the RIDGE framework are:

**Scheduler:** The scheduler is responsible for forming redundancy groups of worker nodes based on their reliability ratings, and assigning a workunit to each redundancy group. Before each scheduling instance, the scheduler waits for the *Scheduling Threshold* number of workers to arrive at the server. Effectively, this waiting transforms the work-distribution model from the BOINC work-on-demand to a batch-scheduling model. The scheduling threshold is a system parameter that can be tuned based on the system requirements: a larger value of the threshold results in more “optimal” grouping, but has larger scheduling overhead and waiting time. Once it has enough workers to proceed, the scheduler obtains the reliability ratings for the available workers from the reputation manager (described below). It then runs a Reputation-based scheduling algorithm [19] to form the redundancy groups and assigns tasks to the worker nodes. The scheduler also handles partially completed workunits: ones that have not been validated successfully and thus need to be re-scheduled. As a default, these

re-scheduled workunits are given priority over new ones, the intuition being that they can be successfully completed with much less effort. In contrast, in BOINC no explicit preference is given to the pending workunits, thus resulting in an accumulation of pending work in the presence of large amounts of new work.

**Reputation Manager:** The reputation manager maintains the reliability ratings of the worker nodes. It stores and fetches the reliability ratings of the workers to and from the system database, and caches the recent values. The scheduler uses these reliability ratings in making its scheduling decisions. The reputation manager is also responsible for updating the reliability ratings of worker nodes when a workunit is validated: a node’s rating may be increased or decreased based on the outcome of the validation [19].

**Validator:** This is a part of the BOINC core architecture (while the others are not). The validator initiates the validation process when the required number of results for a workunit arrive at the server, and determines if an agreement is achieved. The outcome of the validation is passed on to the reputation manager which updates the nodes’ reliability ratings accordingly.

**Re-Scheduler:** When a validation fails, the re-scheduler decides the number of additional tasks to be created for the failed workunit, which can be based on factors such as the number of matching results obtained in the validation process, the reliability ratings of participating workers, etc. It could as well discard all the tasks for the workunit and create a fresh one on its behalf, depending on the design choices made. As a default, the re-scheduler creates one additional task incrementally for the failed workunit.

### 2.2.2 RIDGE Workflow

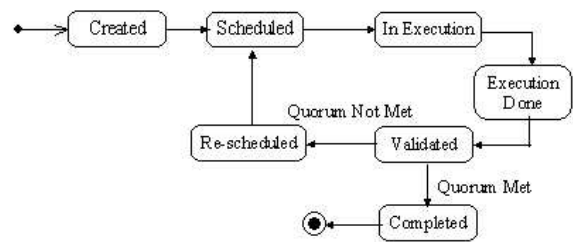


Figure 2: Workunit Life-Cycle

Figure 2 illustrates the workflow in the RIDGE framework through the life-cycle of a workunit. Workunits are *created* and put in the RIDGE database. Additional workunits are created as the work queue empties to maintain a minimum workpool size at the server. Worker nodes arrive at the RIDGE server requesting work. The request handler informs the scheduler about the arrival of a worker node and

blocks the worker node until the scheduler is ready to allocate work. The worker nodes that check in with the scheduler are put in a worker queue. When the number of available workers meets the scheduling threshold, the scheduler performs the allocation of workunits. In our framework, priority is given by default to partially completed workunits whose tasks are assigned to the most reliable available workers. The remaining workers in the worker queue are then grouped into redundancy groups and each group is given tasks of one workunit to execute. The replication factor of each workunit thus varies depending on the associated group size. At this point, the workunit transits to the *scheduled* state. Once the workunit is scheduled, the worker nodes in its associated group pick up their assigned tasks and start executing them. The workunit is now *in execution*.

Each worker node returns the result of the task it executed to the server. When the minimum number of results for a workunit that are required for validation have arrived, the *validation* process is triggered to verify the results using the validation scheme. The outcome of this validation determines the success or failure of the workunit execution. If the validation succeeds, then the workunit is considered to be complete, otherwise, the workunit needs to be *re-scheduled*. The re-scheduler then incrementally creates new tasks for this workunit which are eventually allocated by the scheduler. Also, depending on the outcome of the validation, the reliability ratings of the worker nodes are updated by the reputation manager.

### 2.2.3 Reputation-based Scheduling

Now we briefly describe the worker reliability estimation and grouping algorithms that are the key components driving the RIDGE scheduling framework.

**Worker Reliability Estimation:** The reliability ratings of workers are learned over time based on the results returned by them to the server. A worker’s reliability  $r_i(t)$ , at a given time  $t$ , is given as follows:

$$r_i(t) = \frac{n_i(t) + 1}{N_i(t) + 2},$$

where  $n_i(t)$  and  $N_i(t)$  are respectively the number of valid responses generated and the total number of tasks attempted by the worker by time  $t$ . The rating of a worker is updated each time it is assigned a task, based on the response it returns (a missing or late response is treated as incorrect). Various heuristics have been proposed in [19] that guide how to update the ratings of the workers on each validation. In this paper, we use the ‘Neutral’ heuristic, that updates a worker’s rating only upon a successful validation and remains neutral otherwise.

**Scheduling Algorithms:** Various grouping algorithms that depend on different heuristics and that have different cost

vs. performance tradeoffs have been proposed and evaluated through a simulation study in [19]. Examples of scheduling algorithms proposed include First\_Fit, Best\_Fit, and Random\_Fit. In this paper, we use the Random\_Fit algorithm for workload allocation. The idea behind this algorithm is to randomly select workers for replicating a workunit until the redundancy group formed meets the *Target Success-Rate* or the workunit’s group-size reaches the *MaxClients* size. In our implementation, we simply select the workers in the order they arrive at the server. The detailed algorithm is shown below:

---

**Algorithm 1** Random-Fit ( $G$  worker-group,  $\tau$  workunit-list,  $R_{max}$  Maximum Group-size,  $R_{min}$  Minimum Group-size,  $\lambda_{target}$  Target Success-Rate)

---

```

1: while  $|\tau| \geq 1$  do
2:   Select workunit  $\tau_i$  from  $\tau$ 
3:   repeat
4:     Wait for a worker  $w_i$  to arrive
5:     Add worker  $w_i$  to  $G$ 
6:     Schedule task of workunit  $\tau_i$  to worker  $w_i$ 
7:     Calculate  $\lambda$ , likelihood of successful validation from
       group  $G$ 
8:   until  $(|G| \geq R_{min} \wedge \lambda \geq \lambda_{target}) \vee |G| = R_{max}$ 
9:   Clear worker-group  $G$ 
10: end while

```

---

## 3 Evaluation

In this section, we evaluate the RIDGE framework and present a comprehensive performance comparison of RIDGE against vanilla BOINC. We first describe our experimental setup along with the metrics used, followed by the evaluation results.

### 3.1 Experimental Setup

We have deployed BOINC/RIDGE on PlanetLab [3]—a shared distributed infrastructure consisting of donated machines. Our Grid consists of 120 nodes which serve as the worker nodes. The BOINC/RIDGE server runs on a dedicated machine outside the PlanetLab infrastructure. We used the BLAST (Basic Local Alignment Search Tool) [4] bioinformatics application as our test application. In our setup, BLAST is run as a BOINC project by writing a BOINC-specific wrapper around it. Each workunit consists of a BLAST database file and an input sequence that has to be compared with each sequence in the database file. BLAST performs the sequence comparison and generates an output file result which is returned to the server. We have used a standard BLAST database file igSeqNt, with sizes of 28MB and 55MB for our experiments. The input sequence

was a randomly selected sequence from the database file and is of length 770 bytes. M-first voting is used as the verification technique. To isolate the impact of RIDGE vs. BOINC scheduling, we have disabled ‘Re-scheduling’ in the initial results presented (later, we re-enable it). Thus, in our first set of experiments, a workunit whose validation is not successful for the first time is deemed to have failed and is discarded from the work queue. Each experiment is run for 2 hours and is repeated 3 times to smooth the effects of the underlying load fluctuations in PlanetLab.

### 3.2 Evaluation Metrics

To evaluate the performance of RIDGE against BOINC, we use the following metrics:

- *Success-Rate*: The success-rate for a run is defined as the ratio of the number of successful workunit completions (without re-scheduling) to the total number of workunits allocated during the run.
- *Throughput*: The throughput for a run is defined as the total number of workunits completed during the run.
- *Makespan*: The makespan of a workunit is defined as the difference between the dispatch time of the workunit’s first task and the completion time of the workunit.

Here are some additional metrics we use to quantify the resource utilization of BOINC and RIDGE schedulers:

- *Group-Size*: The Group-Size of a workunit is defined as the number of tasks of the workunit that are actually scheduled to workers.
- *Quorum-Size*: The Quorum-Size of a workunit is defined as the number of tasks of the workunit completed when the validation succeeds.

Note that at the time of validation, some tasks may not have been scheduled yet, which are discarded. The number of tasks scheduled by the time of validation correspond to the Group-size, while the number of tasks that return by the time of validation correspond to the Quorum-size.

### 3.3 Correctness Evaluation of Reliability

We first evaluate the performance of BOINC against that of RIDGE w.r.t. to the correctness behavior of workers in a throughput-oriented environment, where the primary objective is to obtain correct results and maximizing throughput. In this environment, all the results returned by the workers are considered to be ‘timely’, however hosts could return incorrect results. Thus, the reliability of a worker reduces to the probability of returning a correct result. The goal of this set of experiments is to evaluate the following:

given a desired success-rate, how effectively do BOINC and RIDGE achieve it with respect to resource consumption and throughput.

Since the nodes in the PlanetLab testbed are completely reliable w.r.t. correctness, we emulate the reliabilities of the nodes using synthetic distributions in this set of experiments. M-first voting, with  $M=2$  is used as the verification technique so that if at least two results match, the matched result could be considered ‘correct’. The desired success-rate was set to be 0.75 (i.e, at least 75% of the workunits should be completed without re-scheduling.). While a success-rate of 0.75 may appear to be a relatively lax requirement, note that this is the success-rate only for completion with first-time validations. Moreover, selecting this value allows us to explore the success rate-throughput trade-off more clearly in our experiments, as we will show below. In fact, our results will show that a high success rate target may not always be desirable in a throughput-oriented environment. To emulate various real-world reliability scenarios, we generated individual worker reliabilities from 3 different probability distributions:

- **HighRE**: A highly reliable environment (HighRE) where a majority of the workers are reliable was emulated using the complement of a heavy-tailed distribution (1-Pareto(a,b) with parameters  $a=1, b=0.1$ ).
- **LowRE**: A highly unreliable environment (LowRE) where a majority of the workers are unreliable was emulated using a heavy-tailed distribution (Pareto(a,b) with parameters  $a=1, b=0.2$ ).
- **ModRE**: A moderately reliable environment (ModRE) with a mix of reliable and unreliable workers was emulated using a Uniform distribution with  $\text{mean}=0.5$ .

#### 3.3.1 Impact of BOINC Replication Factor

We begin by evaluating the performance of BOINC for various fixed replication factors for the three different reliability environments mentioned above. The replication factor is varied from a minimum of 2 to a maximum of 6, to determine the best static BOINC configuration that achieves the desired success-rate for each reliability distribution.

Figures 3(a) and 3(b) show BOINC’s performance in terms of success-rate and throughput respectively. Figure 3(a) shows that for a given reliability environment, the success-rate monotonically increases with the replication factor. This is expected since  $M=2$  is fixed and the greater the number of replicas, the higher is the probability of getting at least 2 results that match. However, from Figure 3(b), we observe that as the replication factor increases, the throughput initially increases, but then decreases beyond a point. The low throughput for small replication factors is a consequence of the low success-rate for these

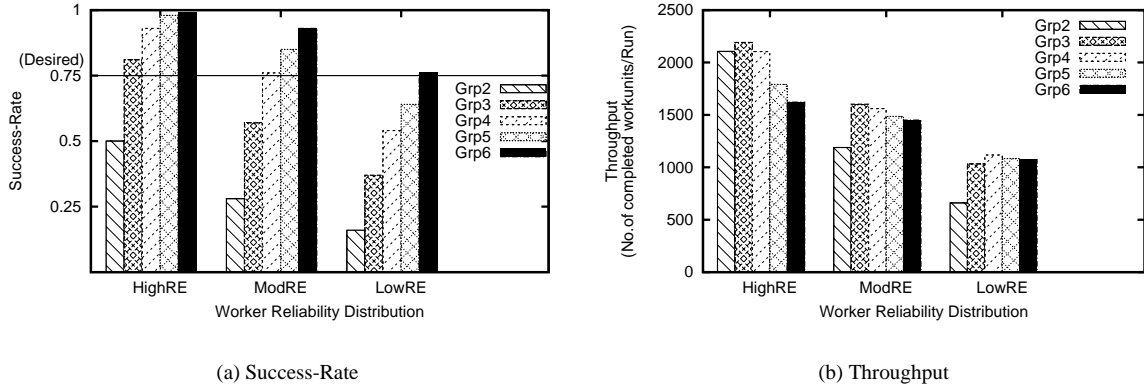


Figure 3: Performance Comparison of different BOINC configurations.

values, as too little replication results in too many failures and hence low throughput. However, beyond a certain degree of replication, the increase in success-rate comes at the cost of a drop in throughput. This is because resources are being overprovisioned leading to poor resource utilization and thus reduced throughput. Moreover, we observe that for a success-rate of 0.75, the optimal replication factors for HighRE, ModRE, and LowRE are 3, 4, and 6 respectively, providing a (success-rate, throughput) combinations of (0.81, 2191), (0.76, 1560) and (0.76, 1073) respectively. Thus, our results show that *given a desired success-rate, there is an optimal fixed replication factor, which further depends on the underlying reliability distribution.*

Since the reliability of the underlying environment may not be known a priori, and also to accommodate varying reliability conditions over time, a BOINC application writer (user) is faced with the choice of a suitable replication factor. A conservative user might select a large replication factor that achieves the desired success-rate in the worst environment possible. However, such a conservative value (6 in the above scenario) results in degraded throughput if the underlying environment is more reliable than expected. Conversely, an optimistic choice would be to select a small replication factor that achieves high throughput in most scenarios. However, such a choice (3 in our experiments) results in a low success-rate if the actual environment is worse than expected. Thus, *the same fixed replication factor does not perform well in all reliability environments.*

Another observation we can make from these results is that for a given reliability environment, even the optimal fixed replication factor might not match the desired success-rate. In fact, it could be much greater than the desired success-rate leading to a compromise in throughput. For example, for HighRE, the optimal replication factor of 3 results in a success-rate of 0.81, which is higher than the

desired value of 0.75. This is a limitation of using *a static replication factor for all workunits.* By dynamically adjusting the replication factor on a per-workunit basis depending on the worker composition, the throughput could be improved while closely matching the desired success-rate. This is precisely the strategy used by RIDGE to schedule work for increased throughput while maintaining the desired success-rate, as we show next.

### 3.3.2 BOINC vs. RIDGE Comparison

We now evaluate the performance of RIDGE against BOINC for the three reliability environments, to achieve the desired success-rate of 0.75. The advantage of RIDGE is that *Target Success-Rate* is a system parameter that can be set to the desired success-rate as needed. This is not possible in BOINC and one has to manually configure it to meet a desired success-rate. The RIDGE server is configured with the *Target Success-Rate* set to 0.75, *MinClients* to 2 and *MaxClients* to 6, corresponding to the static replication factors used for BOINC. For our initial set of results, we assume that the RIDGE server already has the actual worker reliability ratings. Note that under normal execution, RIDGE would learn these ratings first. We omit the learning phase here to separate the effects of the learning algorithm from the performance benefits of the RIDGE scheduling algorithm (We show the learning phase explicitly in the next section).

The goal of these experiments is to evaluate how well RIDGE uses the knowledge of the underlying characteristics of the worker population to dynamically size the redundancy groups and perform close to BOINC's best static configuration (referred to as BOINC\*) for each particular reliability environment, while achieving the desired success-rate. For example, for the HighRE distribution, RIDGE is

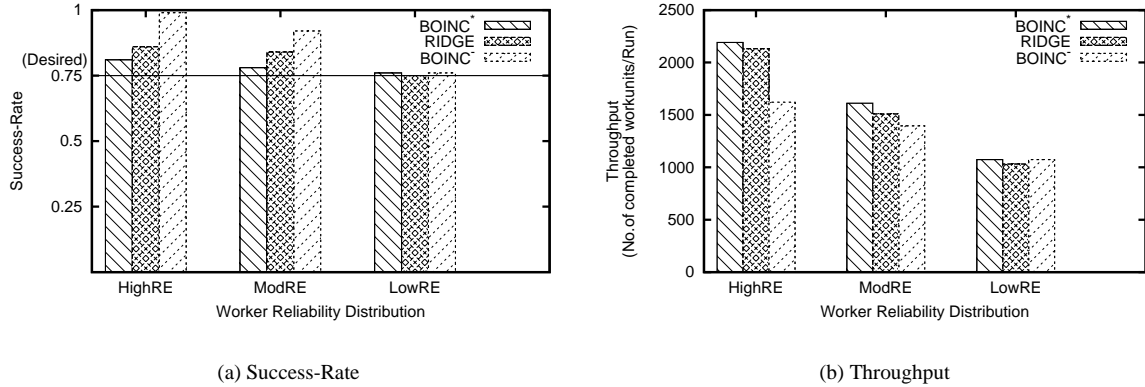


Figure 4: Comparison of RIDGE with optimal (BOINC\*) and conservative (BOINC<sup>-</sup>) configurations of BOINC.

expected to perform close to or dominate the performance achieved by BOINC configured with a replication factor of 3, since that is the optimal for that distribution.

**Performance Comparison:** Figures 4(a) and 4(b) compare the performance of RIDGE with BOINC\* and the conservative BOINC configuration (BOINC<sup>-</sup>) that uses a replication factor of 6 to achieve the minimal desired success-rate of 0.75 for all three reliability environments. The idea is to evaluate how close RIDGE performs to BOINC\* while outperforming BOINC<sup>-</sup>. We observe that the success-rate achieved by RIDGE is at least 0.75 for all the three environments. Thus, RIDGE is able to meet the desired success-rate. Also, the throughput achieved by RIDGE is comparable to that of BOINC\*, for all the three cases. We also notice that RIDGE performs at least as well or better than BOINC<sup>-</sup> for all cases. The small throughput gap between RIDGE and BOINC\* is because of the following reasons:

- The slightly higher resource consumption of RIDGE when compared to BOINC\* as discussed below.
- There is a small overhead in the RIDGE server due to the workers having to first check-in with the scheduler and waiting for a scheduling decision before requesting work. This overhead is evident in the total number of tasks scheduled by RIDGE, which was about 2% less than BOINC\* in all cases.

Distr	Group Size		Quorum Size	
	BOINC*	RIDGE	BOINC*	RIDGE
HighRE	2.85	3.06	2.37	2.42
ModRE	3.74	4.20	2.86	3.05
LowRE	5.52	5.54	3.89	3.75

Table 1: BOINC\* vs RIDGE Resource Utilization

**Resource Utilization Comparison:** To understand how closely RIDGE tunes the group-sizes to that of BOINC\*, we compare the resource utilization of RIDGE against BOINC\* in Table 1. From Table 1, we observe the following for each reliability environment:

- The average group-size of BOINC\* is slightly less than the pre-specified fixed replication factor (3, 4, and 6 respectively for HighRE, ModRE, and LowRE). This is because even though BOINC replicates each workunit to the fixed replication factor number of tasks, not all of them are scheduled to workers. In BOINC, a worker could pick up any task from the workpool available, resulting in a small degree of randomization in scheduling (that is, it is not serial). Moreover, as soon as a workunit is validated successfully, its unscheduled tasks are taken off the work queue, thus resulting in lower resource-consumption than expected.
- The Group-size of RIDGE is slightly greater than BOINC\* for two reasons. First, RIDGE serially schedules work, unlike BOINC, so that if RIDGE decides to replicate a workunit to ‘n’ tasks, these tasks are scheduled to the ‘n’ workers that arrive consecutively. Thus, unlike BOINC, there is a high probability that all the tasks for a workunit are scheduled in RIDGE. Second, the reputation-based scheduling algorithms used in RIDGE ensure that each group formed strictly surpasses the *Target success-rate*. This conservative approach results in additional workers being added to a group even if the group is very close to meeting the *Target success-rate*.
- The average quorum-sizes for RIDGE and BOINC\* are very close (with RIDGE having a smaller quorum-size for LowRE). This indicates that RIDGE requires about the same number of workers to return results for a successful validation. This result could potentially help in improving the overall utilization if already allocated tasks could be pre-empted.

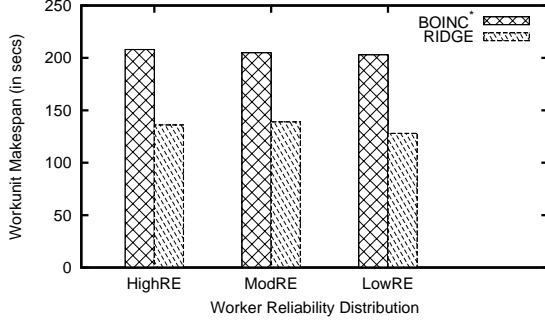


Figure 5: Makespan Comparison

**Makespan Comparison:** Makespan is an important metric when the time taken to complete a number of workunits or tasks must be bounded such as in a service-oriented environment. Figure 5 compares the average makespan of each workunit in RIDGE with BOINC\* for the three reliability environments. We notice that RIDGE has an approximate 35% decrease in makespan when compared to BOINC\*. This is a consequence of the serial scheduling in RIDGE vs. the randomized scheduling in BOINC.

In summary, the results in this section show that *despite the minor overheads incurred, RIDGE performs comparable to BOINC\*, using automatic dynamic replication, and is superior to the conservative BOINC\**.

### 3.4 Timeliness Evaluation of Reliability

In this section, we evaluate the performance of BOINC against RIDGE w.r.t. the timeliness of workers in an environment where getting work done within certain time-constraints is the primary objective. Here, we assume that every worker is 100% reliable w.r.t. correctness and hence the reliability of a worker reduces to the probability that it returns a result in a ‘timely’ manner. The timeliness of a task is determined by an ‘*Execution-Threshold*’ parameter, which is defined as the maximum task execution time beyond which a task is considered late and discarded. The ultimate goal is to use these reliability ratings to do sophisticated scheduling to support deadline-oriented service environments.

Since all nodes are assumed to be correct in this scenario, a workunit is said to be completed as soon as one scheduled task returns within the ‘*Execution-Threshold*’ time. In other words, M-first voting with M=1 is used as the verification technique. Since M is just 1, now comparably higher success-rates could be achieved for smaller replication factors, and hence, we use a desired success-rate of 0.90 for these experiments.

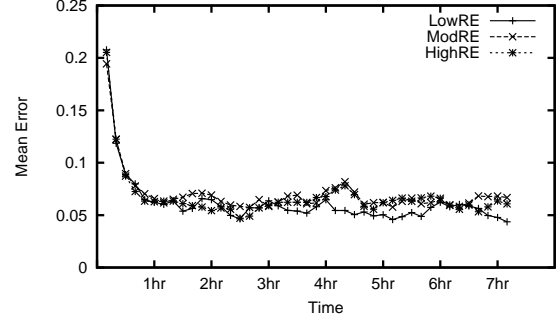


Figure 7: Learning Behavior of RIDGE

### 3.4.1 Emulation of different Reliability Environments

In this set of experiments, instead of using synthetic reliability distributions, we used the actual timeliness values of the results returned by nodes in our PlanetLab testbed to determine their reliability ratings. To emulate different reliability environments, we used different values of Execution-Threshold, so that higher values of Execution-Threshold corresponded to more reliable environments and vice-versa. Figures 6(a), 6(b) and 6(c) show the reliability distributions produced by using Execution-Threshold of 120s, 180s and 240s respectively. We refer to these distributions as LowRE, ModRE and HighRE respectively (not to be confused with the synthetic distributions in the previous section). These graphs are obtained by running the server for 3 hours and calculating the reliabilities of the workers based on their performance during that period. From these graphs, we observe an interesting fact that most of the nodes are either highly reliable or unreliable w.r.t. a given Execution-Threshold and there are a very small fraction of nodes with reliabilities in the range 0.2-0.9. This implies that given an Execution-Threshold, learning the reliability of nodes is indeed useful since the node reliabilities are relatively stable over time intervals in the order of a few hours.

### 3.4.2 Learning Behavior of RIDGE

We first evaluate the learning behavior of RIDGE for worker reliability estimation for the three reliability environments described above. For this estimation, the RIDGE server was run for about 7 hours and the actual execution times as well as the reliability ratings estimated by RIDGE for each worker was logged every 10 minutes. The actual reliability of a worker at time  $t$  was approximated by the average reliability of the worker through the 1 hour time interval around  $t$ . The mean error at time  $t$  is then calculated as the difference between the actual reliability and the estimated reliability averaged over all workers. Figure 7 shows the mean error over the 7-hour period. From the figure, we observe

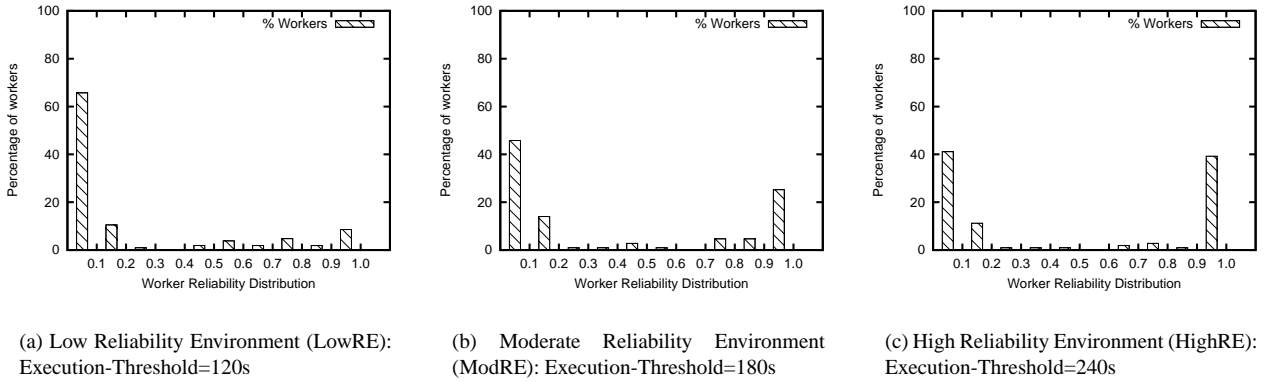


Figure 6: Different reliability environments

that the error converges within 1 hour and settles down to less than 0.1 for all three environments.

### 3.4.3 Performance of BOINC

In this section, we evaluate the performance of BOINC for various fixed replication factors for different reliability environments discussed above. The replication factor is varied from a minimum of 1 to a maximum of 6, to determine the optimal replication factor for a desired success-rate as before. Due to time constraints, BOINC was not run for each combination of reliability distribution and replication factor, and we only evaluated the optimal case and the two cases on either side of the optimal in detail for each reliability environment.

Figures 8(a) and 8(b) show the performance of BOINC for different replication factors, for different reliability environments. We observe similar trends w.r.t. success-rate and throughput metrics for varying replication factors as observed in the results for reliability w.r.t. correctness. The same explanations for the observed trends hold here too. We observe that the optimal replication factor values in this case are 2, 3, and 5 respectively for HighRE, ModRE and LowRE with corresponding (success-rate, throughput) combinations of (0.95, 2378), (0.96, 1723) and (0.91, 1020) respectively. However, as discussed before, since the underlying distribution may not be known a priori, a conservative application designer might operate at a fixed replication factor of 5, to get a minimal success-rate of 0.90 for all reliability environments.

### 3.4.4 BOINC vs. RIDGE Comparison

We now compare the performance of RIDGE and BOINC. Each run of BOINC was set at 2 hrs, while RIDGE was run for 3 hrs with 1 hr for the learning period. The RIDGE

server is configured with a *Target Success-Rate* of 0.90, and *MinClients* and *MaxClients* set to 1 and 5 respectively.

**Performance Comparison:** Figures 9(a) and 9(b) illustrate the performance comparison of BOINC\*, RIDGE and BOINC<sup>-</sup> (that uses a replication factor of 5). We observe that RIDGE meets the desired success-rate of 0.90 in all three environments. Also, from the Throughput comparison graph, we notice that RIDGE in fact has higher throughput than BOINC\*. This is because of two reasons. First, RIDGE can actually form groups of size 1 which is not possible in BOINC (discussed in detail in the following section). This increases the resource utilization and hence throughput compared to BOINC\*. Second, RIDGE does fast serial scheduling. Since only one result is needed for validation, serial scheduling increases the likelihood that the workunit would be completed faster, thus increasing throughput.

Distr	Group Size		Quorum Size	
	BOINC*	RIDGE	BOINC*	RIDGE
HighRE	1.67	1.46	1.05	1.01
ModRE	2.31	1.89	1.12	1.03
LowRE	3.68	3.22	1.41	1.08

Table 2: BOINC\* vs RIDGE Resource Utilization

**Resource Utilization Comparison:** Table 2 illustrates the Resource Utilization of BOINC\* with that of RIDGE for the three reliability environments. An interesting observation is that the Group-Size of RIDGE is *less than* that of BOINC\*. This is because, as discussed before, depending on the value of *Threshold-Time*, there is a high percentage of very highly reliable workers in all three reliability environments. RIDGE actually has the option to create groups

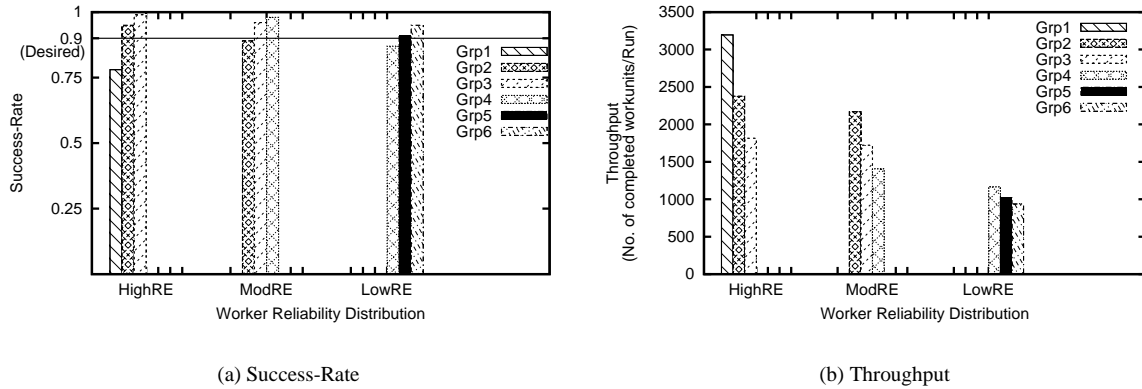


Figure 8: Comparison of different BOINC configurations.

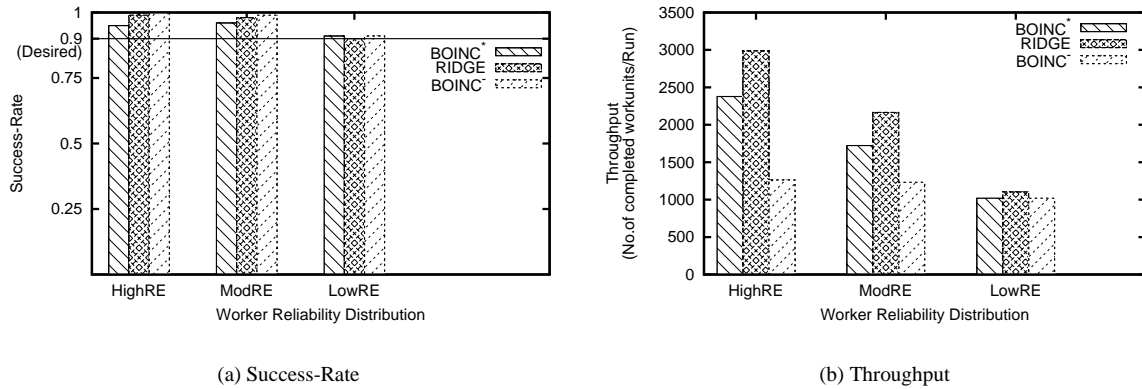


Figure 9: Comparison of RIDGE with BOINC optimal and conservative configurations.

of size exactly 1 using such highly reliable workers, thus, lowering the average group-size. Depending on the reliability of the environment, there may be large number of such single-worker groups, having a positive performance impact, despite the minor overhead in RIDGE due to oversized groups or performance overheads as discussed before. However, forming such small groups is not possible for BOINC\*, since it operates at a fixed replication factor for all the workunits (with an optimal replication factor of at least 2 in all cases).

**Makespan Comparison:** Figure 10 compares the average makespan of a workunit in BOINC\* with that of RIDGE for the different reliability environments. We notice that RIDGE has a lower makespan when compared to BOINC\* for all of the environments. The reasons for this result are similar to those for results corresponding to reliability w.r.t correctness. Another interesting observation from the figure

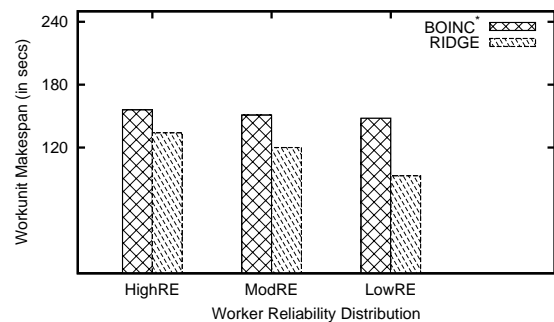


Figure 10: Makespan Comparison

is that in the case of RIDGE, the makespan of a workunit is bounded by the given Execution-Threshold. Thus, by having a bound on the execution time on a per-task basis, we are able to achieve an upper bound on the makespan of the entire workunit. On the other hand, BOINC provides no such bound on the makespan, as is evident in the LowRE environment, where it exceeds the Execution-Threshold of 120s. This is due to the randomization in its scheduling which prevents any upper bound on its execution delay.

### 3.5 Evaluation for Service-Oriented Environments

In this section, we evaluate how BOINC and RIDGE perform in service-oriented environments. We characterize such environments by a high-level unit of work, a *service request*, that is defined as a set of workunits. A request is said to be completed when all its constituent workunits are completed successfully. For this set of experiments, we enable the ‘Re-scheduling’ component of BOINC and RIDGE, so that a workunit that has failed in its first validation is not discarded, but is re-scheduled until it is successfully completed. The definition of some of the metrics used in the previous set of experiments are extended to the ‘Request’ level as follows:

- *Throughput*: Throughput during a run is defined as the number of requests completed in the run.
- *Makespan*: The makespan of a request is defined as the difference between the dispatch time of its first workunit and the completion time of the last workunit.

We reconsider the scenario mentioned in Section 3.3, with a desired success-rate of 0.75. Since the optimal BOINC configuration BOINC\* has already been identified, we compare only the performance of BOINC\* against RIDGE. To emulate ‘Service Request’ behavior, each set of consecutive workunits in the workpool are bundled to model a ‘Service Request’. The performance comparison is shown for two reliability environments, HighRE and LowRE. The results for ModRE are similar and are omitted due to space constraints.

**Makespan Comparison:** Request makespan is a key metric in a service-oriented environment since a service request is not complete until all of its component workunits are complete. Figures 11(a) and 11(b) show the request makespan for BOINC\* and RIDGE as the number of workunits per request is varied from 1 to 8. We observe that as the request size is increased, the makespan for BOINC\* increases much more rapidly when compared to RIDGE. This is explained by the way BOINC and RIDGE schedule and re-schedule work. As mentioned, randomization in scheduling is one factor. Another is that when a validation fails, BOINC puts the additional task in the workpool

and no explicit preference is given to the pending tasks. However, since RIDGE gives preference to pending work compared to new work, RIDGE achieves better requests makespans. This is another factor that supports RIDGE in a service-oriented environment.

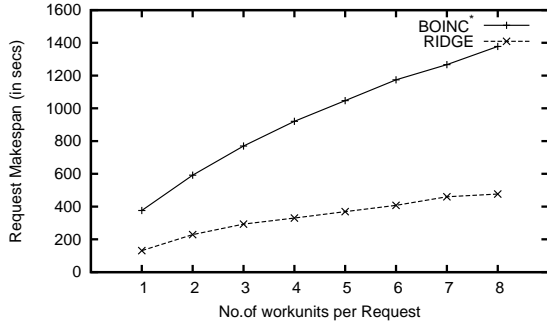
**Throughput Comparison:** Figures 12(a) and 12(b) show the throughput comparison of BOINC\* and RIDGE as the number of workunits per request is varied, for two different reliability environments. From both the graphs, we observe that the request throughput is comparable to BOINC\*. This is a consequence of both serial scheduling and preference to pending workunits adapted by RIDGE.

Our results indicate that *RIDGE not only minimizes the makespan but also maintains the request throughput.*

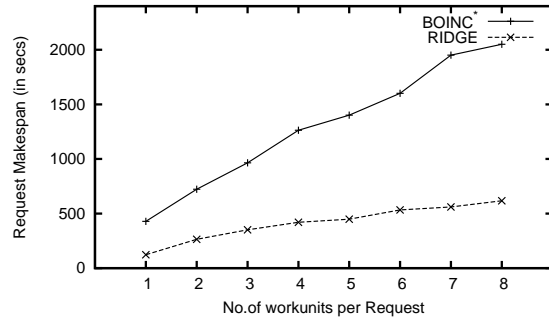
## 4 Related Work

The primary challenge faced by volunteer distributed computing infrastructures, such as BOINC [1] is the inherent unreliability of the participant nodes. While redundant task allocation combined with voting is a popular technique used to deal with such unreliability, the major drawback of such a solution is the low resource utilization due to task replication. Several techniques have been proposed to minimize such redundant task execution and increase resource utilization. Golle and Mirnov [10, 8] present a verification technique that inserts pre-computed images of special spot-checks into distributed tasks to verify results returned by a worker and identify cheaters. Another verification technique [17, 21] employs pre-computed tasks called ‘quizzes’ that are embedded into a batch of (otherwise indistinguishable) tasks allocated to a worker. Such verification techniques avoid the need for replication to validate the correctness of the results. The concept of reducing the redundancy factor in a volunteer computing environment has been proposed in [17] by spot-checking and blacklisting volunteer resources. Another technique that is popularly used is to employ Reputation Systems to gauge the reliability of nodes based on the past interactions. Thus learnt reliability ratings are used for work assignment to workers. Zhao and Lo [21] propose augmenting peer-to-peer cycle sharing systems with a reputation system to reduce the degree of replication required to verify results. All of these proposed techniques deal with unreliability w.r.t correctness alone.

On the other hand, there have been techniques that have emerged to deal with unreliability w.r.t. timeliness alone. [12] use host and CPU availability information of the participant workers to propose techniques for resource selection and design scheduling heuristics based on them, for rapid turn-around times for short-lived applications in Desktop Grid environments. [6] uses a similar mechanism

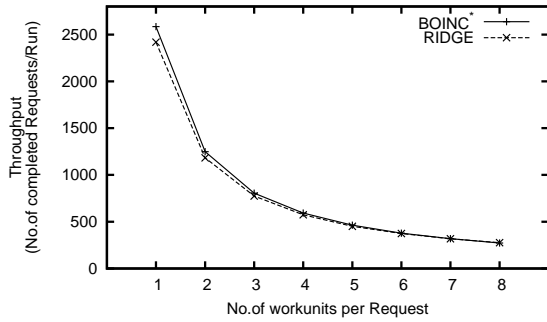


(a) HighRE Makespan

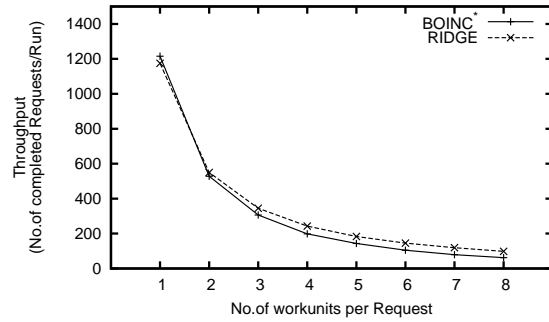


(b) LowRE Makespan

Figure 11: Comparison of Request Makespan for different reliability environments



(a) HighRE Throughput



(b) LowRE Throughput

Figure 12: Comparison of Request Throughput for different reliability environments

to develop a fault tolerant scheduling mechanism in desktop grid environments. [16] present methods for resource availability predictions in cycle-sharing systems that could be used for designing fault-tolerant scheduling mechanisms in such environments. With a similar motive, [11] discusses methods to predict resource availability using a resource life-cycle model. Also, there have been a number of statistical techniques [5, 13, 14] that have emerged independently, discussing ways to estimate a node's CPU/host availability, that could be used for designing effective scheduling mechanisms. While most of the techniques concentrate on either correctness or timeliness, a desirable method would be to learn the behavior of a node with respect to both timeliness and correctness and use this information in scheduling. One such technique is [20] that characterizes a node behavior w.r.t. both correctness and timeliness, classifies them into different categories depending on their reliabili-

ties, and accordingly proposes to tune the replication factor proportionally. Our work uses a more sophisticated approach to combine correctness and timeliness. In particular, our approach provides the user freedom to select either of these properties as desired within a single framework. Moreover, our scheduling techniques [19] also provide flexibility along performance-reliability tradeoffs [18] based on user requirements.

## 5 Conclusions and Future Work

In this paper we have presented the design and implementation of RIDGE, a reliability-aware scheduling system that exploits prior node history to size redundancy groups and select workers effectively. We have implemented RIDGE on top of the BOINC distributed computing infrastructure and have evaluated its performance on a

live testbed consisting of 120 PlanetLab nodes. Our experimental results showed that RIDGE is able to match or surpass the throughput of the best vanilla BOINC configuration under different reliability environments, by automatically adapting to the characteristics of the underlying environment. In addition, RIDGE was able to provide much lower workunit makespans compared to BOINC, which indicates its desirability in service-oriented environments with time constraints. In the future, we intend to explore the interactions between correctness and timeliness in more detail, and explore other systems issues such as the impact of the scheduling-threshold and node selection ordering on the system overheads.

## References

- [1] D. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004)*, 2004.
- [2] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An Experiment in Public-Resource Computing. *Communications of the ACM*, 45(11), 2002.
- [3] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *Proceedings of the Fifth Symposium on Networked Systems Design and Implementation (NSDI'04)*, 2004.
- [4] BLAST. <http://www.ncbi.nlm.nih.gov/blast>.
- [5] J. Brevik, D. Nurmi, and R. Wolski. Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-Peer Systems. In *Proceedings of the of the IEEE International Symposium on Cluster Computing and the Grid (CC-Grid'04)*, Apr. 2004.
- [6] S. Choi, M. Baik, and C. Hwang. Volunteer Availability based Fault Tolerant Scheduling Mechanism in Desktop Grid Computing Environment. In *Proceedings of the Third IEEE International Symposium on Network Computing and Applications*, 2004.
- [7] Climate Prediction Network. <http://www.climateprediction.net/>.
- [8] W. Du, J. Jia, M. Mangal, and M. Murugesan. Uncheatable Grid Computing. In *24th IEEE International Conference on Distributed Computing Systems*, pages 4–11, Mar. 2004.
- [9] Folding@home distributing computing project. <http://folding.stanford.edu>.
- [10] P. Golle and I. Mironov. Uncheatable Distributed Computations. In *Proceedings of CT-RSA*, Apr. 2001.
- [11] B. T. B. Khoo, B. Veeravalli, and T. Hung. Cluster Computing and Grid2005 Works in progress: Dynamic Estimation Scheme for Fault Free Scheduling in Grid Systems. *IEEE Distributed Systems Online*, 6(9), 2005.
- [12] D. Kondo, A. Chien, and H. Casanova. Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids. In *Proceedings of SuperComputing*, 2004.
- [13] D. Nurmi, J. Brevik, and R. Wolski. Minimizing the Network Overhead of Checkpointing in Cycle-harvesting Cluster Environment. In *Proceedings of Cluster 2005*, Sept. 2005.
- [14] D. Nurmi, J. Brevik, and R. Wolski. Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments. In *Proceedings of EUROPAR 2005*, Aug. 2005.
- [15] PPDG: Particle Physics Data Grid. <http://www.ppdg.net>.
- [16] X. Ren, S. Lee, R. Eigenmann, and S. Bagchi. Resource Availability Prediction in Fine-Grained Cycle Sharing Systems. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*, 2006.
- [17] L. F. G. Sarmata. Sabotage-Tolerance Mechanisms for Volunteer Computing Systems. In *Proceedings of the First International Symposium on Cluster Computing and the Grid (CCGrid)*, 2001.
- [18] J. Sonnek, M. Nathan, A. Chandra, and J. Weissman. Reputation-Based Scheduling on Unreliable Distributed Infrastructures. Technical Report 05-036, Dept. of CSE, Univ. of Minnesota, Nov. 2005.
- [19] J. Sonnek, M. Nathan, A. Chandra, and J. Weissman. Reputation-Based Scheduling on Unreliable Distributed Infrastructures. In *Proceedings of the 26th International Conference on Distributed Computing Systems*, July 2006.
- [20] M. Taufer, P. J. Teller, D. P. Anderson, and C. L. Brooks. Metrics for Effective Resource Management in Global Computing Environments. In *Proceedings of the First International Conference on e-Science and Grid Computing*, 2005.
- [21] S. Zhao and V. Lo. Result Verification and Trust Based Scheduling in Open Peer-to-Peer Cycle Sharing Systems. In *Proceedings of the IEEE Fifth International Conference on Peer-to-Peer Systems*, Sept. 2005.