

# Reputation-Based Scheduling on Unreliable Distributed Infrastructures

Jason Sonnek<sup>†</sup>, Mukesh Nathan, Abhishek Chandra and Jon Weissman  
<sup>†</sup>Sandia National Laboratories      Department of Computer Sc. and Engg.  
Albuquerque, NM 87185      University of Minnesota, Minneapolis, MN 55455  
Email: {sonnek, mukesh, chandra, jon}@cs.umn.edu

## Abstract

This paper presents a design and analysis of scheduling techniques to cope with the inherent unreliability and instability of worker nodes in large-scale donation-based distributed infrastructures such as P2P and Grid systems. In particular, we focus on nodes that execute tasks via donated computational resources and may behave erratically or maliciously. We present a model in which reliability is not a binary property but a statistical one based on a node's prior performance and behavior. We use this model to construct several reputation-based scheduling algorithms that employ estimated reliability ratings of worker nodes for efficient task allocation. Our scheduling algorithms are designed to adapt to changing system conditions as well as non-stationary behavior of node reliability. Through simulation of a BOINC-like distributed computing infrastructure, we demonstrate that our algorithms can significantly improve throughput, while maintaining a very high success rate of task completion. Our results also indicate that reputation-based scheduling can handle a wide variety of worker populations, including non-stationary behavior, with overhead that scales well with system size.

## 1 Introduction

Recently, several distributed infrastructures, including peer-to-peer networks and donation Grids, have been proposed to host large-scale wide-area applications ranging from file sharing/file storage to high performance scientific computing [22, 12, 29, 4, 11, 5]. Despite the attractive features of these platforms (scalability, low cost, reduced cost of ownership, and resilience to local failures), widespread deployment of such systems and applications has been elusive. A key problem is the inherent unreliability of these systems: nodes may leave and join unexpectedly, perform unpredictably due to resource sharing at the node and network level, and behave erratically or maliciously. This paper presents a design and analysis of techniques to cope with the inherent unreliability of nodes that execute tasks via donated computational resources.

We present a model in which reliability is not a binary property but a statistical one based on a node's prior performance and behavior. Such a statistical model is important for two main reasons. First, a node's behavior could change with time and hence nodes cannot be classified as being purely reliable or unreliable always. Moreover, representing reliability as a statistical property allows us to incorporate the uncertainty inherent in the system's knowledge of individual nodes' reliability. We adopt a reliability model based on the accumulation of the direct observation of node behavior over prior task executions. An example of such an environment is BOINC [4], or its forerunner SETI@home [5], in which a server distributes tasks to worker nodes and collects results. Since nodes are not reliable, the server generally cannot be certain that the results returned by any given worker are valid unless application-specific verifiers are provided. Many factors may contribute to the unreliability of a node. It has been shown [27] that cheating has been a considerable problem in the SETI@home project. However, it is also possible that nodes have incorrectly

configured software, are hacked, have poor connections to the server, or are highly loaded and cannot return timely results.

We speculate that when excess resources become a visible standard commodity or utility [22, 33, 31], cheating or hacking nodes will become even more prevalent due to economic incentives. In addition, it seems likely that as distributed systems become larger and more widely dispersed, reliability will also decrease due to more failure-prone components and increased exposure to malicious agents and viruses. To deal with uncertainty in the absence of inexpensive verifiers, outsourced computations can be redundantly scheduled to a number of nodes. If we assume that the space of feasible (but not necessarily verifiable) results is sufficiently large, it is very likely that a result returned by a majority of workers will be valid if node collusion has not occurred. Such a majority result could then be treated as the “correct” result of the computation.

A major drawback of using redundancy is that it may reduce the amount of useful work performed. The degree of redundancy is an important parameter: a small degree of replication could decrease the likelihood that the server will receive a verifiable result. On the other hand, a large degree of replication could result in unnecessary duplication of work by multiple resources. Systems like BOINC rely on the application writer to specify this value for each task. Since the reliability of workers in a distributed environment may be uncertain, it is likely that any statically-chosen redundancy value will reduce the effectiveness of the system.

To overcome this problem, we propose techniques to determine the degree of redundancy based on the estimated reliability of the workers. Intuitively, a smaller degree of replication should be possible if the allocated nodes are collectively more reliable. Using a simple reputation system [34], it is possible to determine the likelihood that a given worker will return a correct and timely result with fairly high accuracy. Using these estimates, we show a simple way to compute the likelihood that a group of workers will return a majority of correct and timely results. These group reliability ratings can be used by the system to intelligently schedule tasks to workers, such that the throughput of the system is improved, while still maintaining the server’s ability to distinguish fraudulent results from valid ones.

Applying these techniques in practice introduces a number of challenges. First, the system must be able to learn the reliability of individual workers. A number of different reputation systems have been proposed for this purpose [25, 15, 24, 2, 7], although selecting the right one is dependent on the characteristics of the environment in which it will be deployed. Given these reliability ratings, the system needs an algorithm or heuristic to determine how to match groups of workers to tasks. Since it is likely that the best scheduling technique will be dependent on the environment, we propose a set of algorithms that are tuned to the characteristics of typical environments.

We consider several different algorithms which can be used to guide scheduling decisions on the basis of reliability ratings associated with groups of workers. To reduce overhead in the system, we introduce a technique for calculating a lower-bound on the likelihood that a group will return a verifiable result given individual worker reliability ratings. Finally, we compare the throughput and computational overhead of each of these techniques through simulation of a BOINC-like distributed computing infrastructure. Our results indicate that reputation-based scheduling can significantly improve the throughput of the system for worker populations modeling several real-world scenarios, including non-stationary behavior, with overhead that scales well with system size.

## Research Contributions

We now summarize the main research contributions made in this paper:

- *Reputation-based scheduling*: We present a novel scheduling technique based on a reputation system to achieve more efficient allocation of tasks in an unreliable distributed system. An important feature of our reputation system is that it incorporates metrics of correctness as well as timeliness to generalize the notion of trust to that of *reliability*. We show that our reputation system not only improves the reliability of the application, but also increases its throughput, and introduces overhead that scales well with system size.
- *Statistical reliability estimation*: Another novel contribution of our reputation-based technique is that unlike most existing techniques, it treats reliability not as a binary property, but as a *statistical measure* to incorporate dynamic variations in the environment as well as uncertainty in estimation. We demonstrate the strength of statistical estimation by simulating a large number of real-world reliability scenarios using different probability distributions. Moreover, we emulate both stationary and non-stationary node reliability behavior.
- *Adaptive scheduling*: We present an adaptive algorithm which adjusts scheduling parameters to match conditions in the system. This adaptive algorithm is able to implicitly adjust to the underlying reliability distribution, and is also responsive to non-stationary behavior of node reliability.

## 2 Background and Related Work

### 2.1 Distributed Computing Infrastructures

Numerous computing infrastructures have been designed to utilize idle distributed resources. These systems can be loosely categorized into two groups: those that utilize resources under administrative control, such as Globus [18] and Condor [29], and those that rely on unsupervised donated resources such as SETI@Home [5] and Folding@Home [16]. In this paper, we mainly focus on the latter, as these environments are much more susceptible to unreliability.

The @Home applications [5, 16] and their generalization, BOINC [4], are instances of a growing number of systems which utilize donated computing cycles to solve massive scientific problems. BOINC provides application designers with a middleware that can be used to design and deploy systems in which a master task server assigns computational tasks to a pool of donated computing resources.

In contrast to BOINC, several unstructured cycle-sharing platforms have been proposed [11, 26, 6] in which nodes can act as both a client and a server. These platforms facilitate the formation of ad hoc communities for solving large-scale computing problems. SHARP [19] introduced a resource-management framework for distributed systems in which resource rights are encapsulated within verifiable resource claims. This framework could be used as the basis for a computational economy in which CPU cycles are sold or traded among peers.

## 2.2 Dealing with Unreliability

Dealing with unreliability is a core design challenge in any distributed system and many techniques have been proposed in the literature. Redundant task allocation combined with voting, as used in Byzantine fault-tolerant (BFT) systems [9], is popular due to its general applicability. This approach is also used by most BOINC [4] applications to verify the results of outsourced computations: if a majority of the workers assigned a task return the same result, then the result is deemed valid.

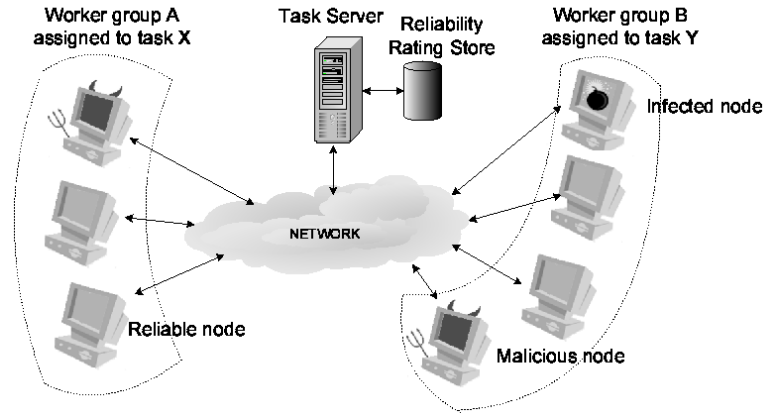
Since task replication could result in lower resource utilization, some techniques have been proposed to verify results for tasks allocated to a single resource. Golle and Mirnov [20] present a verification technique that inserts pre-computed images of special spot-checks called “ringers” into distributed tasks to verify results returned by a worker and identify cheaters. This scheme is extended in [17], where the results are verified by using a Merkle tree of intermediate results stored on the workers. Both these techniques can be used only for verifying computations that exhibit a *one-way* property, and are thus not applicable for general computations. Another verification technique [32, 36] employs pre-computed tasks called ‘quizzes’ that are embedded into a batch of (otherwise indistinguishable) tasks allocated to a worker. When the task server receives a batch of results from a worker, it assumes the results for the real tasks to be correct if the results for all of the quiz tasks are valid. While not dependent on one-way functions, this technique still requires pre-computation of certain tasks, which may be non-trivial or infeasible in many scenarios.

## 2.3 Reputation-Based Scheduling

Reputation systems [30] are commonly applied in peer-to-peer networks to gauge the reliability of nodes [1, 15, 24]. Trust or reputation systems are a general technique for predicting the behavior of distributed entities based on past interactions with these entities. Due to the anonymity inherent in most P2P systems, participants in P2P file-sharing systems lack a means for discriminating between reputable content providers and those who may be trying to distribute viruses or other malware. Using reputation systems, information about the past behavior of peers in the network can be aggregated for the purpose of determining which peers are distributing valid content.

The concept of trust-aware resource management for the Grid was proposed in [7], where a technique is presented for computing trust ratings in a Grid using a weighted combination of past experience and reputation. GridEigenTrust [2] combines this trust-computation technique with the EigenTrust reputation system [24] to provide a mechanism for rating resources in a Grid. This work presents an architecture for managing reputation ratings in a Grid, and proposes using these ratings to perform reputation-based resource selection. However, it does not provide any specific algorithms for reputation-based scheduling. Zhao and Lo [36] propose augmenting peer-to-peer cycle sharing systems with a reputation system to reduce the degree of replication required to verify results. However, their work makes several assumptions: nodes are either strictly trustworthy or untrustworthy, the number of nodes is large relative to the workload which allows nodes to be discarded if untrustworthy, and node behavior is fixed (for the results presented). These assumptions may not often hold in practical scenarios.

Overall, most existing reputation-based scheduling schemes have focused on correctness as the primary metric, and have dealt mainly with binary trust values. The unique elements of our approach include a more general statistical



**Figure 1:** The system model: a server maintains a reliability rating store and uses the ratings to assign tasks to groups of workers.

representation of reliability that includes timeliness as well as correctness, and the use of this metric to improve application and system performance.

### 3 System Model

#### 3.1 Computational Model

Our distributed computing model consists of a central server that assigns computational tasks to a set of worker nodes as illustrated in Figure 1. The worker nodes in this computation model are not centrally-controlled, and could be participating for various reasons. For instance, they may be donating their idle resources voluntarily (e.g.: PlanetLab [12]), or they may be providing their resources in return for some incentive, such as monetary remuneration [23, 28, 22], credit [5, 4], or use of other nodes’ resources in return [3, 13, 19]. Our system model does not make any assumptions about the incentive scheme for worker participation or the workload generation methodology: the computation tasks could either be pre-generated on the server by a project designer, or they may be submitted by users accessing a common service. We assume that the set of tasks that need to be computed by the available set of worker nodes is large enough to keep all workers busy for the duration of the application.

#### 3.2 Reliability Model

Since the participation of worker nodes is voluntary and outside the server’s control, workers may not return correct results in a timely manner for several reasons. First, a node may be overloaded or behind a slow connection, resulting in slow response. Another reason may be that a node is misconfigured, hacked, or infected by a virus, resulting in incorrect computation. Finally, a node may be malicious (deliberately trying to disrupt a computation) or cheating (to gain an advantage in a remuneration scheme, such as gaining extra credit [27]), thus returning wrong results.

We model such unreliable behavior by assigning to each worker a probability of returning a correct response within a “reasonable” time frame. This probability need not be fixed, and could change with time. For instance, nodes

may go offline and come back up again, or some malicious nodes may change their behavior with time—returning correct results for a while to improve their reputation and then deliberately injecting bad results into the system. When modeling these unreliable workers, we assume that each worker acts independently, and that there is no collusion between them.

### 3.3 Redundant Computation and Result Verification

A key consideration in our model is that the server may not have an efficient way of independently verifying each worker response for correctness. While several techniques [20, 17, 21] have been proposed to verify the correctness of results, these techniques are application-specific and are not applicable to general computational scenarios. Results of several computational problems may not even be verifiable by the server without performing the computation itself.

In our system model, we employ a verification technique based on *redundant computation* coupled with *voting*. This technique is adopted by several general computing systems such as BOINC [4]. Under this verification technique, each task is redundantly assigned to a set of worker nodes. Once the workers respond, the server conducts a “vote” among the returned results. If a quorum of workers agrees on a result, the server treats that result to be correct. In the absence of a quorum after voting, the task is rescheduled. While the quorum size could be application-dependent, *majority* is typically used to determine the correct answer. Note that such a voting-based verification scheme does not require any application-specific support or knowledge.

Having described the system model at a high level, we now present some definitions and specify our assumptions more formally.

### 3.4 Definitions and Assumptions

**Definition 1 Task ( $\tau_i$ ):** *A task is defined as a self-contained computational activity that can be carried out by a worker node. Upon completion, each task generates a well-defined result that is returned to the server.*

A task would typically correspond to an independent unit of a larger computation. For example, a task may correspond to computing the determinant of a submatrix, and the result of the task would be the value of the determinant. Another example of a task could be to match a DNA sequence against a subset of gene sequences from a genetic database. In this case, the result could be the best matching gene and the similarity score.

**Definition 2 Solution Space ( $\Sigma$ ):** *The solution space of a task is the set of potential result values that can be returned for the task.*

For instance, a task whose answer is Boolean has a two-element solution space,  $\Sigma = \{true, false\}$ . On the other hand, a task whose answer is drawn from the set of integers has an infinite solution space,  $\Sigma = I$ . We assume that the solution space for the tasks in our model is of sufficiently large cardinality, so that it is unlikely that two workers will independently return the same wrong result.

**Definition 3 Reliability ( $r_i$ ):** *Reliability of a worker  $i$  is defined as the probability that the worker returns a correct result within a (system-defined) time period.*

Note that reliability is not a binary property—a node could return the correct result some of the time, and a wrong result at other times. Moreover, the reliability property of a worker could also change with time (e.g.: due to outages, fluctuating load, malicious node behavior, etc.).

**Definition 4 Redundancy Group<sup>1</sup>( $G_i$ ):** *Redundancy group for a task  $\tau_i$  is defined as the group of worker nodes assigned to compute the task.*

In most existing systems, the size of each redundancy group is typically set to a fixed static value selected by the application designer or the system administrator. This value may be determined empirically, although often it is simply based on a rule of thumb. In our system model, the redundancy factor for each group can be different and dynamically determined, and is dependent on the reliability of the group’s constituent worker nodes.

**Definition 5 Quorum:** *We say that a group  $G_i$  has reached quorum if some number of worker nodes, which may be fixed or dependent on the group size, return the same result.*

In our system model, we say a group has reached quorum if a majority of the workers return the same result. In general, the quorum size could be dependent on the cardinality of the solution space, for instance, a binary solution space would likely require a larger quorum.

**Definition 6 Likelihood-of-Correctness ( $\lambda_i$ ):** *Likelihood-of-correctness for a group  $G_i$  is defined as the probability that the group would return a correct result based on majority voting.*

The likelihood-of-correctness  $\lambda_i$  for a group represents the collective reliability of the group. This value is dependent on the individual reliability values of the constituent nodes of the group. We will see in the next section how this value can be computed for groups using the reliability of individual workers.

## 4 Reputation-based Scheduling

Having described the system model, we now present a reputation-based scheduling algorithm for distributing the server workload among the worker nodes. This algorithm employs reliability ratings of individual worker nodes for task assignment in order to improve the overall throughput and success rate of task completions. This reputation-based task scheduling algorithm consists of the following steps:

- Estimating reliability ratings of individual worker nodes.
- Using the estimated worker reliability ratings to compute the likelihood-of-correctness (LOC) of possible groups.
- Grouping workers for task assignment based on LOC estimates to maximize the throughput and success rate of task completions.

We will now describe each of these steps in more detail, discussing the various techniques and algorithms employed in each case.

---

<sup>1</sup>In the rest of the paper, we would refer to a redundancy group simply as a *group* unless required to avoid confusion.

## 4.1 Estimating Reliability Ratings

We use a *reputation system* to estimate the reliability ratings of individual worker nodes. These reliability ratings are learned over time based on the results returned by the workers to the server. All workers report their results to a centralized server, so a local reputation system can be employed. We estimate a worker’s reliability  $r_i(t)$ , at a given time  $t$ , 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 correct responses generated and the total number of tasks attempted by the worker by time  $t$ . By this formula, the reliability rating of a worker is initialized to  $\frac{1}{2}$ , corresponding to having no knowledge about its actual reliability. The rating of each 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). While we assume that the server learns the reliability ratings from scratch using its own observations, it is possible to use a peer-to-peer reputation reporting system [24, 15] to improve the accuracy of these ratings or reduce the time to learn them. For example, in the case of BOINC, a worker node may be multiplexed across different project servers which could share their local estimates of the worker’s rating among themselves.

Recall from Section 3.3 that the server employs a majority-based voting scheme to determine the correctness of a task. Thus, if the workers in a group reach a majority on their results, the server accepts the majority answer as the “correct” result. In this case, it would increase the reliability ratings of the workers that are part of the majority, and decrease those of the remaining workers treating their responses to be incorrect.

However, this still raises the question of how to update the ratings of workers in a group that doesn’t reach quorum. We present four different heuristics to handle this case:

- *Neutral*: If no quorum is achieved for a group, the ratings of its workers are not changed. Intuitively, in the absence of a quorum, this heuristic maintains neutrality towards the reputation of all workers in the group.
- *Pessimistic*: If a group does not achieve a quorum, all of its workers are given negative ratings. Intuitively, in the absence of a quorum, this heuristic penalizes all the workers, and is more pessimistic than the Neutral heuristic.
- *Optimistic*: In the absence of a quorum, this heuristic increases the reliability ratings of any set of workers that agree on the result value. It penalizes those worker whose answers do not match any other answers from the group. Intuitively, this heuristic is based on the assumption that the probability of two workers returning the same wrong result independently is negligible, thus treating any matching answers to be pseudo-correct.
- *Verification-based*: This heuristic assumes the presence of an independent verifier to determine the correct answer even in the absence of a quorum. This heuristic improves the reliability ratings of those workers whose responses are verified to be correct. Note that our system model is not dependent on the use of such a verifier, and we use this heuristic as an upper bound for the other heuristics, as it relies on perfect knowledge of the results’ correctness.

## 4.2 Computing the Likelihood-of-Correctness

The likelihood-of-correctness (LOC) of a group represents the probability of getting a correct answer from that group using the majority-based voting criterion of verification. This value can be computed using the individual reliability ratings of the members of the group, as estimated above. Consider a group  $G = \{w_1, \dots, w_{2k+1}\}$  consisting of workers  $w_i, i = 1 \dots 2k + 1$ <sup>2</sup>. Let  $r_i$  be the reliability rating of a worker  $w_i, i = 1 \dots 2k + 1$ , at a given point in time. Then, the LOC  $\lambda$  of the group  $G$  is given by:

$$\lambda = \sum_{m=k+1}^{2k+1} \sum_{\{\epsilon: |\epsilon|=m\}} \prod_{i=1}^{2k+1} r_i^{\epsilon_i} \cdot (1 - r_i)^{1-\epsilon_i} \quad (1)$$

where  $\epsilon = \{\epsilon_1, \dots, \epsilon_{2k+1}\}$  is a vector of responses from the workers in the group, with 1 representing a correct response, and 0 representing an incorrect response. The criterion for determining correctness is based on achieving a majority, as described above. For example, for a group  $G$  consisting of 5 workers  $w_1$  through  $w_5$ , one possible vector could be  $\{1, 1, 0, 0, 1\}$ , indicating correct responses from workers  $w_1, w_2$ , and  $w_5$ . Intuitively, Equation 1 considers all possible subsets of the given set of workers in which a majority of workers could respond correctly. It then computes the probability of occurrence of each of these subsets as a function of the reliability rating of the workers. Note that the likelihood of the false-positive case where a majority of workers return the same wrong answer is negligible, and hence ignored in Equation 1.

### 4.2.1 Lower Bound for Likelihood-of-Correctness

As can be seen from Equation 1, calculating the likelihood-of-correctness for a group results in a combinatorial explosion of the possible subsets that need to be enumerated. In fact, the complexity of computing the  $\lambda$  value can be shown to be  $O(2^{2k})$ , which is infeasible for most practical purposes. To reduce the cost of computing  $\lambda$  values for multiple groups, we use a lower bound  $\lambda^{lb}$  for  $\lambda$  that is much simpler and more efficient to compute. This is obtained from Equation 1 using the arithmetic-geometric means inequality which is a special case of Jensen's Inequality [14].

$$\lambda^{lb} \geq \sum_{m=k+1}^{2k+1} \binom{2k+1}{m} \cdot \prod_{i=1}^{2k+1} r_i^{\alpha_m} \cdot (1 - r_i)^{1-\alpha_m}, \quad (2)$$

where  $\alpha_m = \frac{\binom{2k}{m-1}}{\binom{2k+1}{m}}$ . It can be shown that the complexity of computing  $\lambda^{lb}$  is  $O(n^2)$ , and is thus much more efficient to compute than the actual value of  $\lambda$ . The grouping algorithms, which we will describe shortly, use the lower bound function to compute  $\lambda$ .

### 4.2.2 The Role of LOC in Task Scheduling

To determine the size and composition of the groups, the system relies on a parameter indicating whether or not the LOC for a proposed group is acceptable. That is, we require some value  $\lambda_{target}$  such that if  $\lambda \geq \lambda_{target}$ , then we conclude that  $G$  is an acceptable group. We refer to  $\lambda_{target}$  as the *target LOC*. Since  $\lambda_{target}$  represents a lower-bound on the likelihood that a group will return a successful result, it can be thought of as a target success rate for the system.

<sup>2</sup>We consider odd-sized groups to avoid ambiguity in defining majority for even-sized groups.

Choosing a proper value for  $\lambda_{target}$  is critical to maximizing the benefit derived from the system. If  $\lambda_{target}$  is too small, many groups may return incorrect results, causing the tasks to be rescheduled. If it is set too high, the scheduler will be unable to form groups which meet the target, and the scheduler will degenerate to forming large fixed-size groups, adversely affecting the system throughput. Thus, the target LOC must be carefully selected to fit the reliability distribution of the workers.

### 4.3 Forming Redundancy Groups

So far, we have described heuristics for estimating individual worker ratings, and provided a mechanism for combining these ratings to determine the reliability of groups. We now present algorithms to assign workers into groups for task allocation, using these heuristics and mechanisms. The goal of forming these groups is to maximize both the throughput of successful tasks completions (those that result in correct results) and the rate of successful task completion (success rate) given a set of individual worker ratings.

There is a natural trade-off between the throughput of successful task completion and the success rate. By forming larger groups, we generally increase the likelihood that an individual group will return a correct answer, but we decrease the number of tasks attempted, which may in turn decrease the throughput of successful tasks. Conversely, decreasing the average group size will make each group less likely to return correct results, but may increase the number of successful tasks completed due to the increase in the number of tasks attempted. One can imagine scenarios in which either metric would be preferred over the other. In a service-oriented environment in which end-clients are waiting for responses to specific tasks, we may prefer an algorithm which produces 17/20 verifiable results to one which produces 18/30. However, in an environment where all tasks are equally important, an algorithm which produces 28/33 correct results may be more desirable than one which produces 21/22. In short, if we wish to bound the latency experienced by individual tasks, success rate is a more important metric than throughput. If we simply wish to maximize the number of tasks completed, throughput is more important.

More formally, given a set of workers  $W = \{w_1, \dots, w_n\}$ , a *group formation* algorithm would produce a partitioning  $G = \{G_i\}$ , where a task  $\tau_i$  is assigned to each group  $G_i$ . A *throughput-optimal group formation* algorithm would generate a partitioning  $G$  that maximizes the number of successful task executions given a desired success rate  $\lambda_{target}$ . A *success rate-optimal group formation* algorithm would generate a partitioning  $G$  that maximizes success rate given a desired throughput rate. For simplicity of exposition, in this paper, we focus on throughput maximization. However, we also discuss techniques to maximize both metrics simultaneously in a later section.

We now present several reputation-based scheduling algorithms that form groups in a tractable manner. Each algorithm attempts to form groups  $G_i$  from the pool of available worker nodes such that  $\lambda_i$  of each  $G_i$  satisfies  $\lambda_{target}$ , thus achieving two goals: (a) increasing the likelihood of obtaining a correct result from the worker group working on the assigned task (in turn decreasing the likelihood of re-scheduling a task) and (b) increasing resource utilization by forming worker groups whose size varies based on the reliability rating of its members. The only property of the workers used by the algorithms is their reliability ratings.

### 4.3.1 Fixed-Size

This is the baseline algorithm for our system model as it represents “standard-best-practice” exhibited in systems such as BOINC. The Fixed-size algorithm randomly assigns workers to groups of size  $R_{max}$ , where  $R_{max}$  is a statically-defined constant. Every worker of a given group  $G_i$  is assigned the same task. This algorithm does not use the reliability ratings  $r_i$  of workers to size  $R_i$  in an intelligent way. For a given set of workers, this algorithm will form a fixed number of groups, irrespective of  $r_i$ .

---

**Algorithm 1** Fixed-Size ( $w$  worker-list,  $\tau$  task-list,  $R_{max}$  group-size)

---

```
1: while  $|w| \geq R_{max}$  do
2:   Select task  $\tau_i$  from  $\tau$ 
3:   Assign set of  $R_{max}$  workers  $w_{max}$  from  $w$  to  $G_i$ 
4:    $w \leftarrow w - w_{max}$ 
5: end while
```

---

### 4.3.2 First-Fit

In the First-fit algorithm, the available workers are sorted by decreasing reliability rating. Starting with the most reliable, workers are assigned to group  $G_i$  until either  $\lambda_i \geq \lambda_{target}$  or until the maximum group size  $R_{max}$  is reached. This process is repeated until all the available workers are assigned to a group. Intuitively, First-fit attempts to form the first group that satisfies  $\lambda_{target}$  from the available workers in a greedy fashion. By bounding the size of  $G_i$  with  $R_{max}$ , we ensure that First-fit forms bounded groups and degenerates to the Fixed-size heuristic in the absence of sufficient number of reliable workers.

---

**Algorithm 2** First-Fit ( $w$  worker-list,  $\tau$  task-list,  $\lambda_{target}$  target LOC,  $R_{min}$  min-group-size,  $R_{max}$  max-group-size)

---

```
1: Sort the list  $w$  of all available workers on the basis of the reliability ratings  $r_i$ 
2: while  $|w| \geq R_{min}$  do
3:   Select task  $\tau_i$  from  $\tau$ 
4:   repeat
5:     Assign the most reliable worker  $w_r$  from  $w$  to  $G_i$ 
6:      $w \leftarrow w - w_r$ 
7:     Update  $\lambda_i$ 
8:   until  $(\lambda_i \geq \lambda_{target} \wedge |G_i| \geq R_{min}) \vee |G_i| = R_{max}$ 
9: end while
```

---

### 4.3.3 Best-Fit

The Best-fit algorithm attempts to form groups  $G_i$  such that  $\lambda_i$  is as close as possible to  $\lambda_{target}$ . The Best-fit algorithm searches the space of available workers and groupings to find a  $G_i$  that exceeds  $\lambda_{target}$  by the smallest possible margin. If no group of size  $R_{max}$  or smaller meets  $\lambda_{target}$ , the algorithm forms a group that falls short of the target by the smallest amount. Intuitively, this algorithm attempts to form the best-fit of worker nodes for a given  $\lambda_{target}$ . As a result, tasks are not overprovisioned with more reliable resources than necessary, and well-balanced groups are formed.

---

**Algorithm 3** Best-Fit ( $w$  worker-list,  $\tau$  task-list,  $\lambda_{target}$  target LOC,  $R_{min}$  min-group-size,  $R_{max}$  max-group-size)

---

- 1: Sort the list  $w$  of all available workers on the basis of the reliability ratings  $r_i$
  - 2: **while**  $|w| \geq R_{min}$  **do**
  - 3:   Select task  $\tau_i$  from  $\tau$
  - 4:   Search for a set  $s$  of  $n$  workers  $w_n$  from  $w$  such that  $\lambda_s$  exceeds  $\lambda_{target}$  minimally
  - 5:   **if** such a set  $s$  is found **then**
  - 6:     Assign the  $w_n$  workers to  $G_i$
  - 7:   **else**
  - 8:     Select the set of  $n$  workers  $s$  for which  $\lambda_{target} - \lambda_s$  is minimized
  - 9:     Assign the  $w_n$  workers to  $G_i$
  - 10:   **end if**
  - 11:    $w \leftarrow w - w_n$
  - 12: **end while**
- 

#### 4.3.4 Random-Fit

The Random-fit algorithm uses reliability ratings to form groups by randomly adding workers to a group  $G_i$  until either  $\lambda_i$  meets  $\lambda_{target}$  or the group has  $R_{max}$  workers. It differs from First-fit in that workers are added to groups randomly, rather than in sorted order.

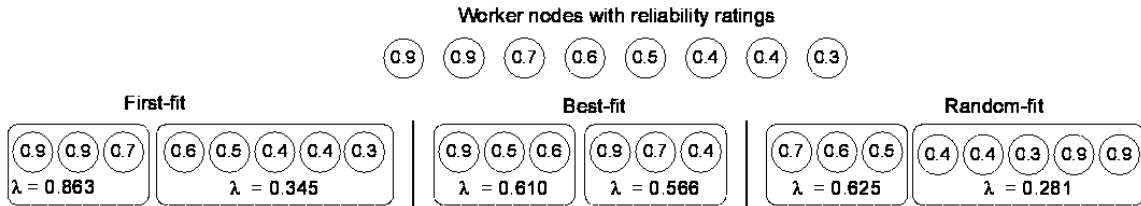
---

**Algorithm 4** Random-Fit ( $w$  worker-list,  $\tau$  task-list,  $\lambda_{target}$  target LOC)

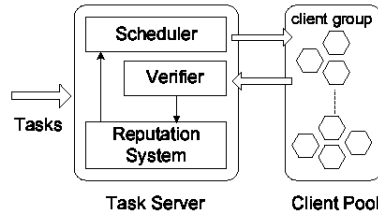
---

- 1: **while**  $|w| \geq R_{min}$  **do**
  - 2:   Select task  $\tau_i$  from  $\tau$
  - 3:   **repeat**
  - 4:     Assign a *random* worker  $w_{ran}$  from  $w$  to  $G_i$
  - 5:      $w \leftarrow w - w_{ran}$
  - 6:     Update  $\lambda_i$
  - 7:     **until**  $(\lambda_i \geq \lambda_{target} \wedge |G_i| \geq R_{min}) \vee |G_i| = R_{max}$
  - 8: **end while**
- 

Given a set of workers and a  $\lambda_{target}$ , each algorithm is likely to produce different groups. A simple example for  $\lambda_{target} = 0.5$  is illustrated in Figure 2. In this example, First-fit is only able to form a single group which meets  $\lambda_{target}$  because it uses all of the highly reliable workers in the first group. Similarly, Random-fit also produces only one group that is able to meet the target as it assigns workers randomly to groups. In contrast, Best-fit is able to form two groups which meet  $\lambda_{target}$  because it searches for groupings whose  $\lambda$  values deviate from the target by the smallest amount.



**Figure 2:** Example node groupings produced by different algorithms for  $\lambda_{target} = 0.5$ ,  $R_{min} = 3$  and  $R_{max} = 5$ .



**Figure 3:** Simulation Components

## 5 Evaluation

In this section, we evaluate the performance of the rating techniques and grouping algorithms described in the previous section through simulation of a donation-based distributed computing platform. In our simulations, we model a large number of real-world scenarios using different distributions for worker reliability values.

Through our simulations, we first measure the rating error associated with various reliability estimation techniques. Next, we combine the most promising of these techniques with our reputation-based scheduling algorithms to evaluate their throughput and success rate of task completion. We then present results for adaptive techniques that dynamically select system parameters to best fit the underlying system. Finally, we evaluate the proposed scheduling algorithms in terms of the overhead introduced to the system, and investigate the impact of the approximate LOC function on the effectiveness of the system.

### 5.1 Evaluation Methodology

Our evaluation is based on a simulator loosely modeled around the BOINC [4] distributed computing infrastructure, which consists of a task server and some number of worker machines. The task server consists of three primary components: the scheduler, the result verifier, and the reputation system. The scheduler assigns tasks to groups of workers based on a task allocation policy. Workers return responses to the task server, which are forwarded to the result verifier. The result verifier uses a verification technique to determine whether or not a result is valid (we use majority voting). The result of this verification process is passed on to the reputation system, which uses the information to update reliability ratings for the workers. Finally, these reliability ratings are fed back into the scheduler for use in its task allocation policy.

We make two simplifying assumptions to enable fair comparison between different grouping algorithms.

- The simulator is *round-based*—work assignment and verification is done periodically in fixed-duration time periods called rounds. The task server assigns work to all the workers at the beginning of a round, and then waits for the workers to return their results. At the end of each round, the server collects and verifies the received results, updates the reliability ratings, and re-forms groups for task allocation in the next round. Workers who fail to respond by the end of a round are assumed to have returned incorrect results. In the results shown here, we ran our simulations for a total of 1000 rounds each. In practice, the length of a round would be linked to the expected execution time of the tasks within it.

Name	Distribution (over [0,1])	Real-world Scenario
Uniform	Uniform	General environment
Heavy-tail-high	1-Pareto( $a = 1, b = 0.1$ )	Majority of reliable workers; a few unreliable workers
Heavy-tail-low	Pareto( $a = 1, b = 0.2$ )	Majority of workers unreliable; major virus/outage
Normal-high	Normal: $\mu = 0.9, \sigma = 0.05$	Reliable environment; most workers reliable
Bimodal	Bi-Normal: $\mu = 0.2/0.8, \sigma = 0.1$	50% reliable workers, 50% unreliable
Normal-low	Normal: $\mu = 0.3, \sigma = 0.1$	Hostile environment, e.g., military scenarios

**Table 1:** Probability distributions used in the simulations to emulate different real-world reliability scenarios.

- The task server has an extremely large pool of work relative to the number of workers available. This assumption is consistent with the projects hosted by the BOINC infrastructure, and is likely to be true for future large-scale scientific computing applications as well. As a result, the task server will always attempt to utilize all of the available workers, and workers will never have to wait for work.

An individual worker’s reliability is modeled by assigning it a probability  $r_i$  of returning a correct result within a round. When a worker is assigned a task, it returns the correct result with probability  $r_i$ . These probabilities are known only to the workers - the task server has no knowledge of these values a priori.

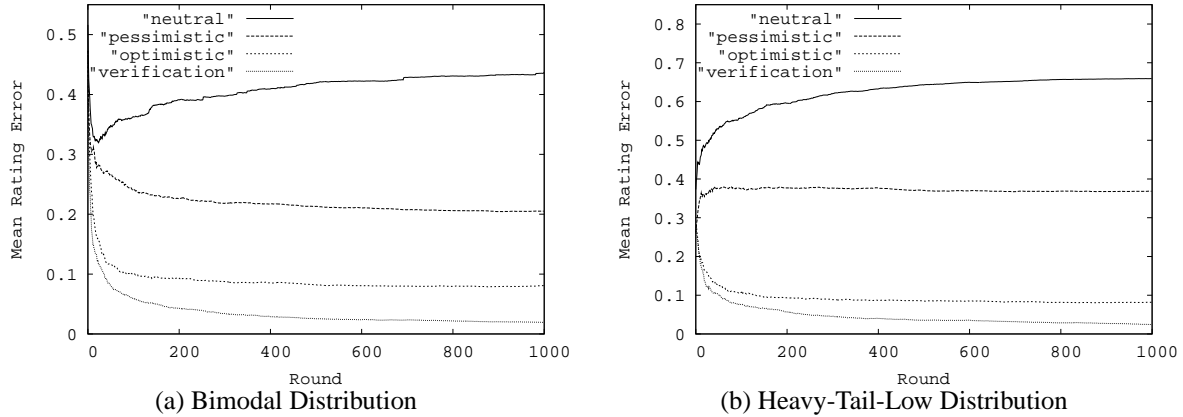
To simulate various real-world reliability scenarios, we generate individual worker probabilities from several different probability distributions. Table 1 lists some of the distributions used in our simulations and the corresponding scenarios modeled by each of them. For instance, we use a normal distribution with a high mean to emulate a highly-reliable system, where most workers are well-connected and return correct results most of the time. On the other hand, we use a bimodal distribution to represent a system that has a mix of highly-reliable workers and compromised or poorly-connected nodes.

## 5.2 Reliability Rating Estimation

In Section 4.1, we presented several heuristics for estimating the reliability ratings of workers. In this section, we evaluate the effectiveness of each of these heuristics. For each of the reliability distributions listed above, we measured the mean rating error across all workers during a simulation of 1000 rounds. We assume that the server has no information about the workers at the beginning of a simulation. Thus, each worker starts with an initial estimated reliability rating of 0.5, indicating that they are as likely to return a correct answer as not. At the end of each round, we update each worker’s rating and compare the estimated ratings to the actual reliability values. The rating error of each worker is computed as  $|actual\_rating - estimated\_rating|$ .

Figures 4(a) and (b) show the results obtained for simulations using the bimodal and heavy-tail-low distributions respectively. Results for other distributions are similar and omitted due to space constraints.

The Neutral heuristic is the least accurate, with a mean rating error of about 40% for the bimodal distribution and 60% for the heavy-tail-low distribution. Since Neutral does not update ratings of a group when a majority is not found, it is unable to learn the ratings of unreliable workers quickly, resulting in highly inaccurate reliability estimates. The Pessimistic heuristic performs better than the Neutral heuristic, because it is able to discover unreliable workers quickly. However, it still has a high rating error (about 20 and 35% for bimodal and heavy-low, respectively). This is



**Figure 4:** Mean Rating Error

because the Pessimistic heuristic assigns negative ratings to *all* workers in a group which fails to achieve a majority, resulting in unfair ratings being assigned to reliable workers that happen to be in a group with less reliable workers.

For both distributions, the mean rating error of Optimistic converges to about 10% within 30-40 rounds. This heuristic performs well because it makes use of the assumption that workers are unlikely to return the same wrong answer independently. Thus, it is able to identify reliable workers, even in the absence of a majority, by matching their common results. At the same time, it is also able to penalize unreliable workers that might have prevented a majority in the first place by returning bad answers. Finally, as expected, the Verification-based heuristic, which assumes the presence of an independent verifier, has the smallest rating error, and can be considered the baseline heuristic for our purposes. In the remaining experiments, we employ the Optimistic and Verification heuristics for rating estimation.

### 5.2.1 Using Bounded History

In the preceding experiment, we assumed that the task server used all of a worker’s past history to calculate its reliability rating. While this approach has the potential to be extremely accurate, it will be slow to respond to sudden changes in a worker’s reliability. To deal with workers whose reliability is non-stationary, we introduce sliding-window versions of each of the rating estimation heuristics. The sliding-window based heuristics maintain a bounded history of information for each worker, and they rely on this window of recent worker behavior to estimate reliability. A smaller window will cause more recent behavior to have a greater impact on the worker’s rating, resulting in more adaptive ratings. However, if the window shrinks too much, the rating values may start oscillating, limiting the achievable accuracy for stable workers.

In Figure 5, we compare the accuracy of sliding-window versions of Optimistic and Verification-based heuristics to their infinite history counterparts. We use a window size of 20 rounds in these experiments. As expected, while the accuracy is not as high as in the infinite history case, the sliding-window versions of both heuristics are still fairly accurate (both converge to 15% rating error within 20 rounds). In Section 5.4, we will show that the sliding-window techniques are extremely useful for quick adaptation when reliability ratings change suddenly.

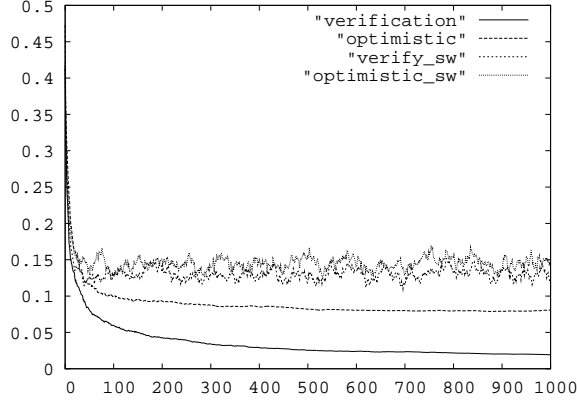


Figure 5: Rating Error Comparison: Sliding-Window vs. Infinite History

### 5.3 Reputation-Based Scheduling

We now evaluate the various reputation-based grouping algorithms described in Section 4.3. We begin by describing the metrics and parameters used in our evaluation.

#### 5.3.1 Metrics and Parameters

To evaluate the effectiveness of the grouping algorithms, we use the following metrics:

- *Throughput* ( $\rho$ ): The throughput during a round is defined as the number of tasks for which a majority was achieved during that round (i.e., the number of 'successful' tasks).

$$\rho = |T_{success}|,$$

where  $T_{success}$  is the set of successfully completed tasks during a round.

- *Mean Group Size* ( $g$ ): The mean group size for a round is the mean number of workers assigned to each task during the round.

$$g = \frac{\sum_{i=1}^{N_G} |G_i|}{N_G},$$

where  $N_G$  is the total number of groups formed during the round

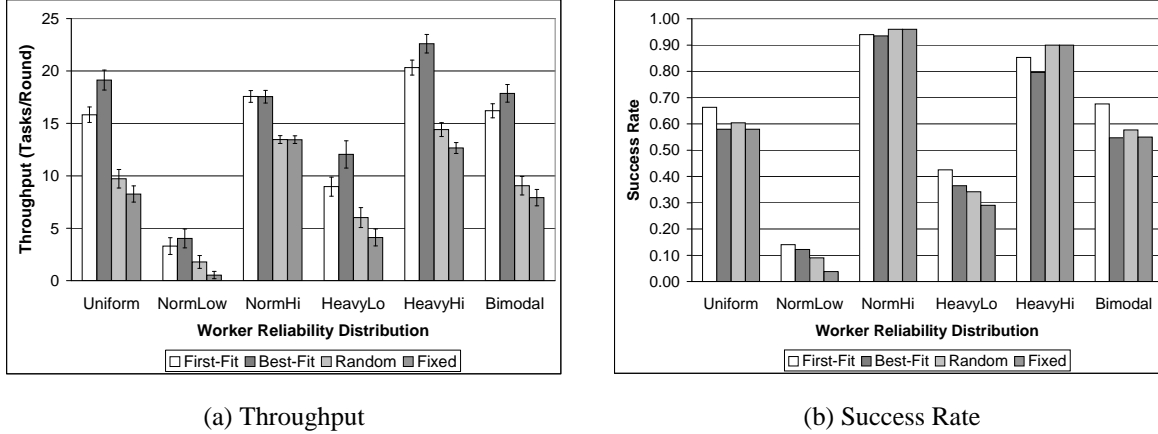
- *Success Rate* ( $s$ ): The success rate during a round is defined as the ratio of throughput to the number of tasks attempted (equal to the number of groups formed) during that round.

$$s = \frac{\rho}{N_G}.$$

To fully understand the behavior of the reputation-based schedulers, we ran an exhaustive set of simulations covering a large parameter space. Table 2 lists the parameters that we varied in our simulations and the values we used for each. We compare the four algorithms described in Section 4.3 (First-fit, Best-fit, Random-fit, and Fixed) for each parameter setting.

Parameter	Values
Worker reliability distribution	see Table 1
Target LOC ( $\lambda_{target}$ )	Success rate of Fixed
Worker Pool Size	100, 1000
Minimum Group Size ( $R_{min}$ )	3
Maximum Group Size ( $R_{max}$ )	3, 5, 7, 9

**Table 2:** Simulation Parameters



**Figure 6:** Algorithm Comparison

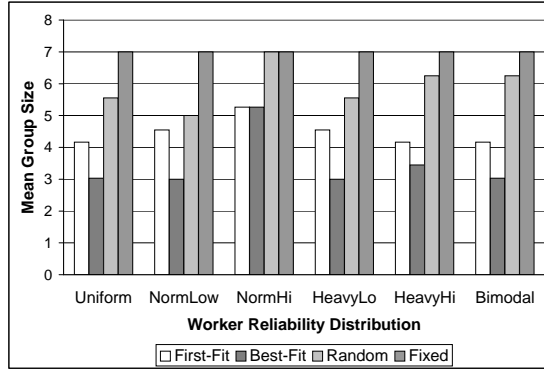
For a given distribution and  $R_{max}$ , we set  $\lambda_{target}$  equal to the success rate of the Fixed algorithm for the same parameter values. This ensures that the success rate of the various algorithms will be approximately the same, facilitating a comparison between our proposed algorithms and the baseline Fixed algorithm. Due to space constraints, we will present a subset of the results here, including descriptions of the most interesting findings.

### 5.3.2 Comparing Scheduling Algorithms

In our first experiment, we measured the mean throughput, group size, and success rate for the different grouping algorithms using a pool of 100 workers. We compared the algorithms for each of the worker reliability distributions described in Table 1 using the Optimistic rating technique.

In Figure 6(a), we present the throughput results for an  $R_{max}$  value of 7 workers. The First-fit and Best-fit algorithms improve on the throughput of Fixed by 25-250%, depending on the worker reliability distribution. The Random-fit algorithm, while not performing as well as First-fit and Best-fit, still outperforms Fixed by about 20-50%.

Figure 6(b) plots the success rate results for the experiment. Since we set  $\lambda_{target}$  equal to the success rate achieved by the Fixed algorithm, we would expect that the mean success rate for the other algorithms to be similar. The success rate of Random-fit and Best-fit is nearly identical to that of Fixed—the minor differences are due to the use of the lower-bound LOC function combined with approximate worker reliability measures. First-fit deviates significantly for most of the distributions due to its greedy group formation policy. Recall from Section 4.3 that First-fit attempts to form groups starting with the most reliable workers, and working down to the least reliable workers. For the distributions



**Figure 7:** Mean Group Size

	$R_{max} = 9$	$R_{max} = 7$	$R_{max} = 5$	$R_{max} = 3$
First-fit	$18.21 \pm 1.29$	$20.33 \pm 1.42$	$22.37 \pm 1.55$	$26.13 \pm 1.86$
Best-fit	$20.56 \pm 1.65$	$22.61 \pm 1.77$	$24.03 \pm 1.74$	$26.01 \pm 1.86$
Random-fit	$11.06 \pm 1.05$	$14.41 \pm 1.32$	$18.95 \pm 1.60$	$26.78 \pm 2.08$
Fixed-size	$10.20 \pm 0.83$	$12.65 \pm 1.05$	$17.32 \pm 1.48$	$26.84 \pm 2.13$

**Table 3:** Effect of decreasing  $R_{max}$  on throughput (Heavy-High distribution)

which have a relatively low average reliability, First-fit will create many groups which exceed  $\lambda_{target}$  when it is forming groups using the higher reliability workers at the head of the worker list. Conversely, for distributions which have a high average reliability, First-fit will have no choice but to form several groups which do not meet the target when it reaches the unreliable workers towards the end of the list. Overall, these results indicate that reputation-based scheduling algorithms significantly increase the average throughput for all of the reliability distributions, while maintaining a high success rate.

Figure 7 shows the mean group-size results for the above experiment. Both First-fit and Best-fit are able to form substantially smaller groups satisfying the target LOC requirement. As a result, these algorithms attempt significantly more tasks in each round, resulting in the substantially higher throughputs shown in Figure 6.

In Table 3, we present the throughput results for varying maximum group sizes using the Heavy-High distribution. As the  $R_{max}$  parameter is reduced, the gap between the Fixed algorithm and the reputation-based algorithms starts to narrow, since it becomes harder to form smaller groups that meet  $\lambda_{target}$ . In particular, if we set  $R_{min} = R_{max}$ , then all of the scheduling algorithms are essentially the same. In this case, all the algorithms form groups of size 3, and the groups formed by each algorithm have approximately the same likelihood of reaching a majority, causing them to have nearly the same throughput.

In this situation, the reputation-based scheduling algorithms can differentiate themselves by either increasing  $\lambda_{target}$  (which will increase the success rate, but may lower the throughput), or decreasing  $\lambda_{target}$  (which will have the opposite effect). This clearly illustrates the importance of selecting a target LOC which matches the goals of the underlying system.

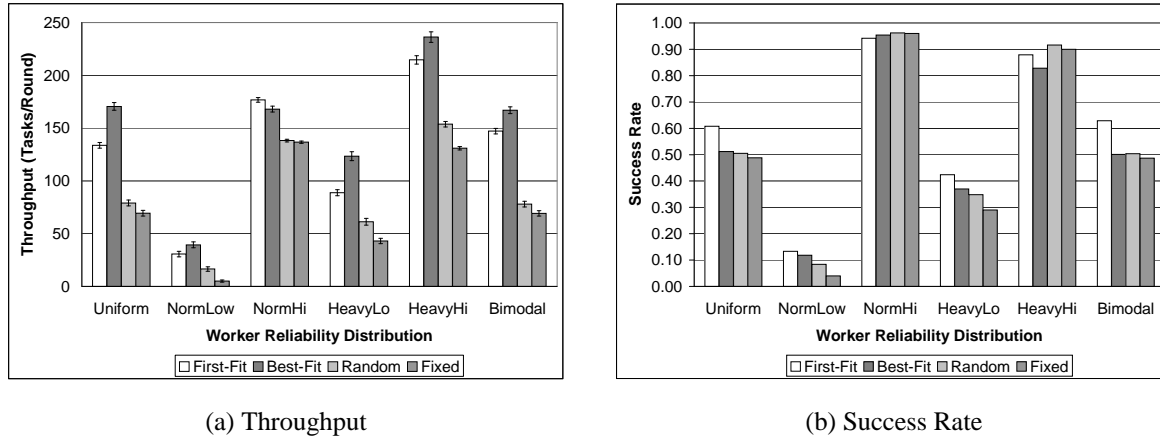


Figure 8: Mean Throughput and Success Rate, Network Size=1000

### 5.3.3 Effect of Scale

In our second experiment, we use the same parameter settings as the previous experiment, but increase the network size from 100 to 1000 workers. Figure 8 shows the throughput and success-rate results for this experiment. Scaling the size of the network up to 1000 workers causes a proportional increase in the throughput, without affecting success rate much. Clearly, increasing the size of the network has no impact on the effectiveness of these techniques. We will consider the overhead associated with the different scheduling algorithms in Section 5.6.

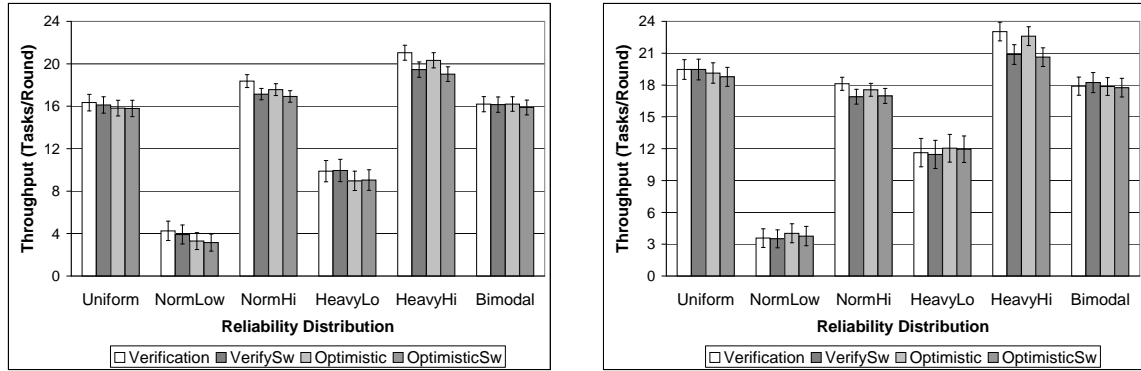
### 5.3.4 Effect of Bounded History

To measure the impact of less accurate reliability ratings, we repeated the above experiments using the Verification-based reliability estimation heuristic, along with the sliding-window versions of Verification-based and Optimistic. In Figures 9(a) and (b), we compare the throughput achieved using First-fit and Best-fit, respectively, for each of the reliability estimation heuristics. In this experiment, the size of the worker pool was 100, and  $R_{max} = 7$ . As expected, the Verification-based rating estimation heuristic nearly always has the highest throughput. The average throughput for the Optimistic heuristic is 2-10% lower in most cases, and the two sliding-window rating techniques suffer reduced throughput between 2% and 25%.

It is interesting to note that Best-fit has higher throughput when using the less accurate rating techniques in some cases. We hypothesize that this is due to the complex grouping function employed by Best-fit. Small changes in the ratings may cause Best-fit to form significantly different groups. In some cases, the groups formed for ratings which are less accurate on average may actually achieve slightly higher throughput, assuming that the ratings used are not extremely inaccurate.

## 5.4 Dealing with Non-Stationary Workers

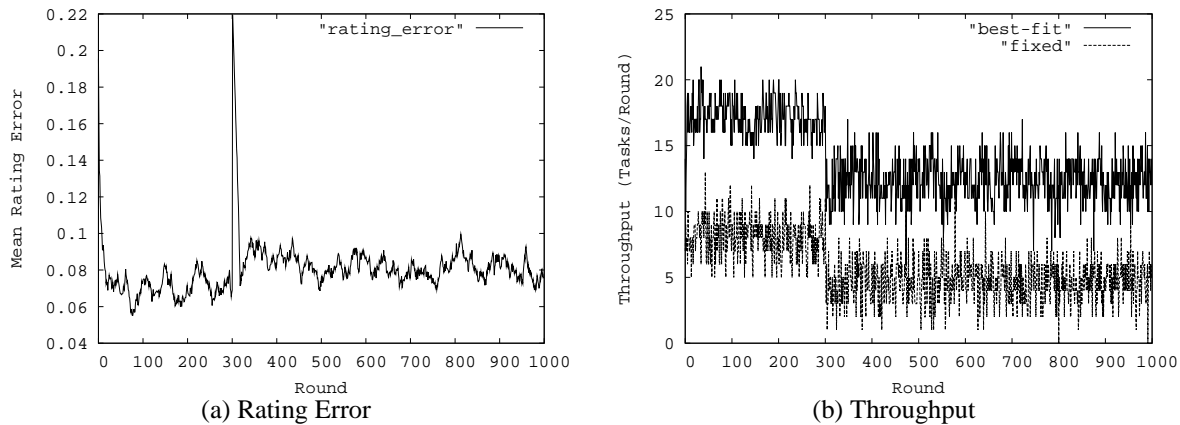
In the previous set of experiments, each worker was assigned a static reliability value (internal probability of returning a correct response) for the duration of the simulation based on the collective reliability distribution. Real-world workers



(a) First-fit

(b) Best-fit

**Figure 9:** Throughput for various reliability rating heuristics, window size=20



(a) Rating Error

(b) Throughput

**Figure 10:** Large-scale Blackout: Performance of Best-fit and Fixed algorithms coupled with window-based Optimistic rating estimation heuristic.

are likely to exhibit non-stationary behavior, i.e., their reliability will vary with time. We consider two different scenarios in which the workers’ reliability changes dynamically: one emulating a large-scale blackout, and the other modeling a large-scale recovery. Both of these scenarios are characterized by large number of workers suddenly changing their reliability behavior. In each scenario, we evaluated the effect of non-stationary workers on throughput and success rate. We also analyzed the impact of changes in worker reliability on the rating error. In both experiments, we used the sliding-window Optimistic rating heuristic.

### 5.4.1 Large-Scale Blackout

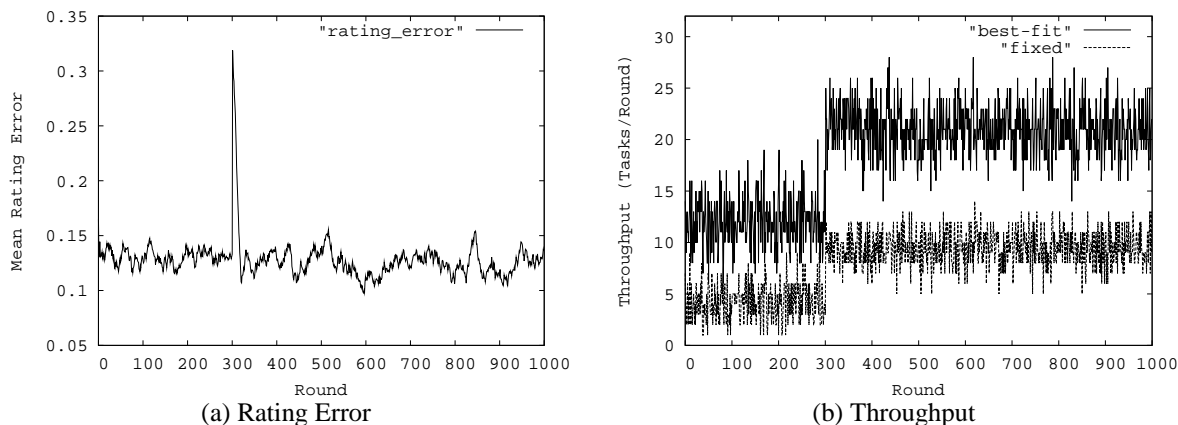
We first consider a large-scale worker blackout scenario. Such a scenario could correspond to a real-world event such as a network partitioning, a large organization crash, or a major virus, which may suddenly compromise the reliability of a large number of workers. To emulate such an event, we modified the simulation so that 30% of the workers transitioned from a highly-trusted normal-high distribution to an unreliable heavy-low distribution after round 300.

Figures 10(a) and (b) show the rating error and throughput, respectively, for a simulation using the Best-fit algorithm with  $R_{max} = 7$ . In Figure 10(a), we can clearly see a massive spike in the rating error after round 300. However, the system quickly adapts to the change, and the rating error stabilizes near its previous levels within a few rounds. This indicates that the sliding-window rating estimation technique is effective in dealing with sudden changes in worker reliability ratings.

Figure 10(b) is particularly interesting. Both the Best-fit and Fixed algorithms suffer a considerable dip in throughput after the blackout. We would expect this from the Fixed algorithm, since it just blindly assigns workers to groups, and does not take into account the worker’s reliability ratings. Since the overall reliability of the environment decreased, the throughput of the Fixed algorithm decreased.

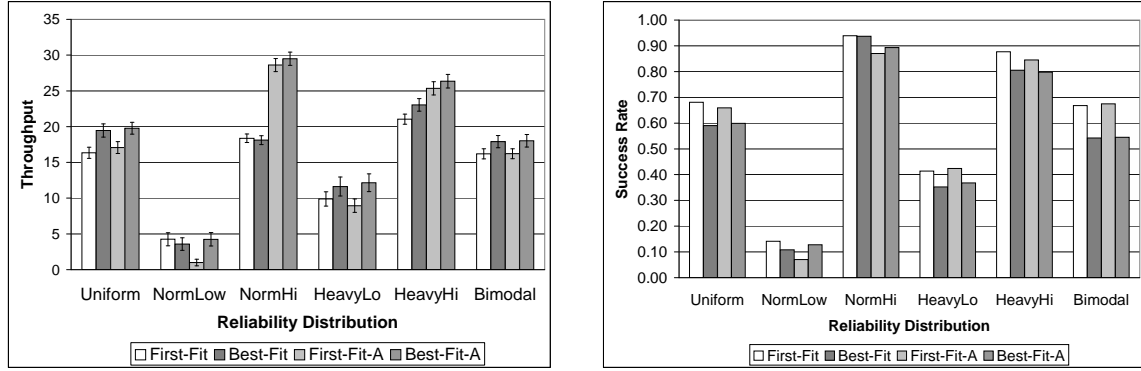
Ideally, the Best-fit algorithm would adapt to the new environment and maintain a relatively high throughput. However, the reputation-based algorithm exhibited the same behavior as the baseline algorithm in this experiment. The reason for this seemingly disappointing behavior is that Best-fit is still operating under a target LOC that was tailored to the higher reliability environment. Since the system has fewer trusted workers at its disposal, it is not able to form many groups that meet this high  $\lambda_{target}$ . Thus, it ends up creating a number of very large groups in an attempt to meet unrealistic success rate requirements. If the system could have adapted  $\lambda_{target}$  to match the new environment, it may have been possible to maintain a higher throughput. We will revisit this scenario using adaptive techniques in Section 5.5.

### 5.4.2 Large-Scale Recovery



**Figure 11:** Large-scale Recovery: Performance of Best-Fit and Fixed heuristics coupled with window-based Optimistic rating estimation heuristic.

This scenario is the opposite of the blackout scenario considered above. Here, we emulate a scenario where we go from a less-trusted environment (heavy-low) to a more-trusted environment (heavy-high), e.g., when connection to a network partition is re-established, a site crash is recovered, or a virus is cleaned up from large number of infected nodes. Figure 11 shows the results for the Best-Fit and Fixed heuristics again coupled with the window-based Optimistic rating heuristic. The rating-error plot (Figure 11(a)) is very similar to that for the large-scale blackout case,



(a) Throughput

(b) Success Rate

**Figure 12:** Comparison of First-fit and Best-fit algorithms with static and adaptive  $\lambda_{target}$  values.

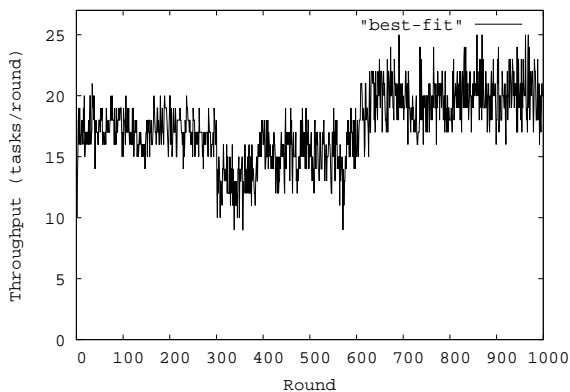
and the rating values stabilized quickly after the large-scale change. As can be seen from Figure 11(b), the Best-Fit heuristic was able to easily meet the target LOC requirements. In this case, it would have been possible for the system to require a higher success rate without negatively impacting throughput. The throughput plot clearly shows that the Best-fit heuristic benefited more from the increase in the proportion of reliable workers than did the Fixed heuristic.

## 5.5 Adaptive LOC

There are several drawbacks associated with using a static value for the target LOC parameter  $\lambda_{target}$ . First, it is difficult to select an appropriate value without prior knowledge of the expected worker population. Second, the worker reliability distribution may vary with time due to node churn and other events which affect a node's reliability. To maximize the benefit derived from the system, it would be desirable to dynamically adapt  $\lambda_{target}$  to meet current conditions. The task server is ideally suited to select an appropriate  $\lambda_{target}$ , since it constantly updates the reliability ratings of the workers and monitors the performance of the system.

Neither throughput nor success rate alone is a sufficient metric for determining an optimal value of  $\lambda_{target}$ . As described in Section 4.3, there is a trade-off between the two values, and there are many realistic scenarios in which one is preferable to the other. Thus, determining an optimal value requires us to consider throughput and success rate simultaneously. Such an optimization is an instance of a *multi-objective optimization (MO)* problem. A common approach to solving an MO problem is to use techniques such as Goal Programming [10, 35] or Multilevel Programming [8, 35] that reduce the multiple objectives to a single objective, and then employ standard Linear Programming techniques to obtain a solution.

We use a similar approach to determine the  $\lambda_{target}$  value adaptively. In our approach, we define a new metric  $M(s, \rho) = s \cdot \rho$ , where  $s$  and  $\rho$  represent the success rate and throughput respectively. We then use a hill-climbing algorithm to adapt the target LOC  $\lambda_{target}$  based on measurements of the current value of  $M$ . In Figure 12 we compare the throughput and success rate achieved with a static value for  $\lambda_{target}$  to that achieved with our adaptive algorithm. In this experiment, the parameter settings were the same as those described in Section 5.3.



**Figure 13:** Large-Scale Blackout: Effect of Adaptive  $\lambda_{target}$

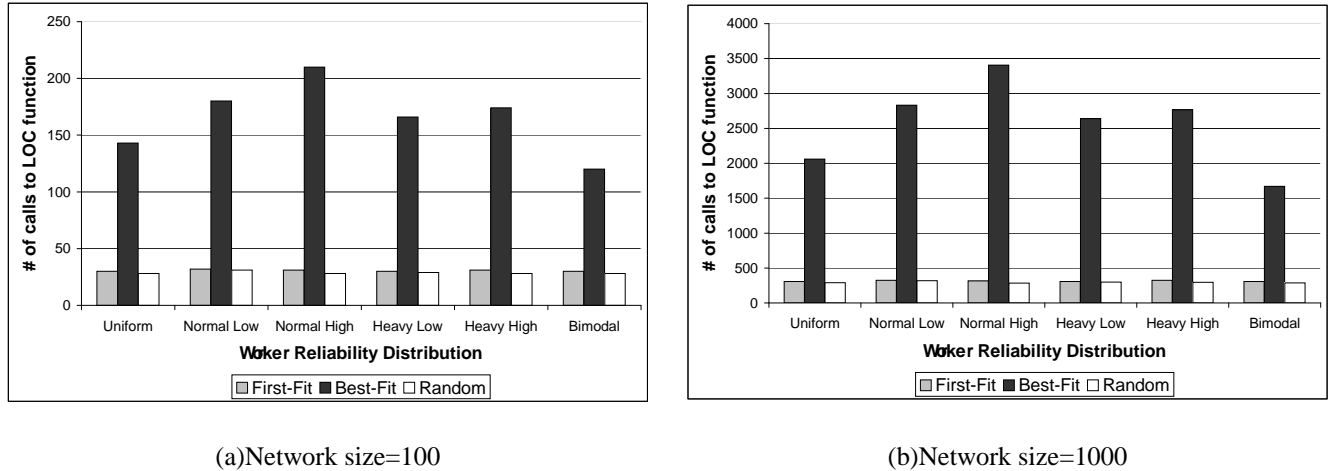
Despite the simple optimization function used for adaptation, the Adaptive Best-fit algorithm improves throughput by about 5-10% in most cases, while maintaining a comparable success rate. For the Heavy-high and Normal-high distributions, the adaptive algorithm posts more impressive gains of 20% and 65%, respectively. The reason for these large increases is that the success rate achieved by the Fixed algorithm for these extremely reliable environments is quite high. As a result, when attempting to match this success rate, Best-fit also has to form large groups. By reducing the required success rate slightly (2-3%), the Adaptive Best-fit algorithm is able to form much smaller groups, and thus to achieve much higher throughput. This clearly demonstrates that our simple optimization function is fairly effective at balancing throughput and success rate.

The results for the First-fit algorithm are not as good when using the adaptive  $\lambda_{target}$ . It still posts modest improvements in throughput for several of the distributions, but the throughput is actually lower for the distributions with a low mean reliability. We hypothesize that First-fit is not able to take advantage of the adaptive LOC due to the fact that it has less freedom over how it groups workers.

We now revisit the large-scale blackout scenario described in Section 5.4. In Figure 13, we demonstrate the impact of an adaptive  $\lambda_{target}$  during a large-scale blackout using the Best-fit algorithm. As in the non-adaptive case, the throughput drops drastically in round 300, immediately after 30% of the workers become unreliable. However, in this case the system immediately starts compensating for the drop in reliability by reducing  $\lambda_{target}$ . Approximately 100 rounds later, the average throughput is near pre-blackout levels. Near round 500, the system tries to raise  $\lambda_{target}$  in an attempt to improve the success rate. As a result, we see the average throughput drop between rounds 550 and 600. The system automatically corrects for this mistake and starts to lower  $\lambda_{target}$  again. It stabilizes near round 700 at a higher throughput (but considerably lower success-rate - 80% vs. 96%) than it was achieving before the blackout. This experiment clearly demonstrates the value of dynamically updating  $\lambda_{target}$  based on current conditions in the system.

## 5.6 Overhead

In this section, we compare the overhead of the grouping algorithms in terms of the number of invocations of the LOC function. The time spent computing the LOC for a given group is the primary component of the overhead incurred by the reputation-based schedulers, so we can compare the algorithms in a system-independent manner.



**Figure 14:** Overhead (Number of calls to LOC function)

	uniform	normal	heavy-tail	bimodal
First-fit	0.0010 ± 0.0016	0.0000 ± 0.0002	0.0014 ± 0.0052	0.0012 ± 0.0052
Best-fit	0.0784 ± 0.0587	0.0015 ± 0.0023	0.0210 ± 0.0123	0.0910 ± 0.0755
Random-fit	0.3532 ± 0.2325	0.0025 ± 0.0033	0.0561 ± 0.0442	0.3328 ± 0.2252

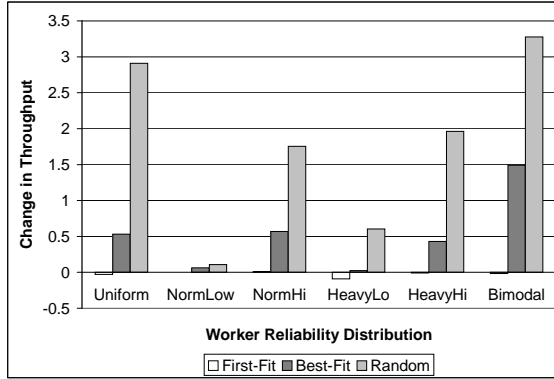
**Table 4:**  $\lambda$  vs.  $\lambda^{lb}$ : Average error in practice

Figure 14 shows the average number of calls to the LOC function during a single round for network sizes of 100 and 1000. The First-fit and Random-fit algorithms form groups in a sequential fashion. Each grouping considers at most a constant number of worker pairs, and the number of groups is linear in the number of workers, so the number of calls to the LOC function scales linearly with the size of the network. The Best-fit algorithm is more expensive because it tries to form the best possible groups by using a binary search of the available workers. Thus, the number of calls to the LOC function is  $O(n \log n)$ , where  $n$  is the size of the network.

## 5.7 Accuracy of $\lambda^{lb}$

In Section 4.2, we presented a function (Equation 2) to compute  $\lambda^{lb}$ , which is a lower-bound on the likelihood that the group  $G$  will return a majority of correct answers. In this section, we will analyze the impact of using an approximation function on the effectiveness of the system. The lower bound function corresponds to the GM in the AM-GM inequality, a special case of Jensen’s inequality. The GM is commonly accepted as an acceptable lower bound for the AM, since it is widely believed that a tighter lower-bound will be too computationally expensive. For functions that comply to this inequality, the difference between the AM (the actual LOC function in our case), and the GM increases as the spread of the values increases. Therefore, as the disparity between the low and high reliability ratings within a group increases, the lower bound diverges more and more from the actual  $\lambda$  value.

We empirically computed the difference between  $\lambda$  and  $\lambda^{lb}$  for each of the reputation-based algorithms. The results are illustrated in Table 4. As expected, we found that the error for Random-fit, which may end up grouping workers



**Figure 15:** Change in throughput when using  $\lambda$  vs.  $\lambda^{lb}$

with very dissimilar ratings, was quite high (0.05-0.3). In contrast, the First-fit algorithm, which groups similarly rated workers, had negligible error ( $\leq 0.001$ ). The error for Best-fit fell in the middle but is still relatively small (0.002-0.09).

To quantify the effect of these errors on the effectiveness of the reputation-based algorithms, we repeated the experiment from Section 5.3.2 using the actual LOC function instead of the lower-bound. The throughput results are shown in Figure 15.

As expected, these results show a correlation between the error experienced by an algorithm and the benefit associated with using  $\lambda$  instead of  $\lambda^{lb}$ . In addition, the benefit is the highest for the distributions where the error was the worst (uniform, bimodal). Based on these measurements, we can conclude that Best-fit could improve its throughput by up to 10% by using a more accurate value for  $\lambda$ , and Random-fit could gain up to 25% over the values presented earlier.

## 6 Conclusions and Future Work

In this paper, we have presented a design and analysis of techniques to handle the inherent unreliability of nodes in large-scale donation-based distributed infrastructures, such as P2P and Grid systems. We proposed a reputation-based scheduling model to achieve efficient task allocation in such an unreliable environment. Our reputation system represents the underlying reliability of system nodes as a statistical quantity that is estimated based on the prior performance and behavior of the nodes. Our scheduling algorithms use the estimated reliability ratings to form redundancy groups that achieve higher throughput while maintaining desired success rates of task completion. In addition, we present a technique for adaptively adjusting scheduling parameters to match the underlying reliability distribution, which can be used to control the system's response to non-stationary node reliability. We evaluate our algorithms using a simulator based on the BOINC [4] distributed computing infrastructure. In our simulation, we varied the underlying reliability distribution of the worker reliability values to emulate several real-world scenarios. Our simulation results indicate that reputation-based scheduling can significantly improve the throughput of the system (by as much as 25-250%)

for worker populations modeling several real-world scenarios, including non-stationary behavior, with overhead that scales well with system size.

In the future, we plan to investigate optimization techniques for our adaptive scheduling algorithm that can be tuned to different performance metrics. We also intend to implement our techniques in a real testbed (e.g., one using BOINC) and to use real workload traces to evaluate the efficacy and overhead of our algorithms under real-world deployment.

## Acknowledgment

We would like to thank Arindam Banerjee for providing helpful pointers on the lower-bound function and its analysis.

## References

- [1] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, 2001.
- [2] B. Alunkal, I. Veljkovic, G. von Laszewski, and K. Amin. Reputation-Based Grid Resource Selection. In *Workshop on Adaptive Grid Middleware*, 2003.
- [3] K. Anagnostakis and M. Greenwald. Exchange-Based Incentive Mechanisms for Peer-to-Peer File Sharing. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, 2004.
- [4] D.P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Proceedings of the 5th ACM/IEEE International Workshop on Grid Computing*, 2004.
- [5] D.P. 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.
- [6] A. Awan, R. Ferreira, S. Jagannathan, and A. Grama. Unstructured Peer-to-Peer Networks for Sharing Processor Cycles. *Journal of Parallel Computing*, 2005.
- [7] F. Azzedin and M. Maheswaran. Integrating Trust into Grid Resource Management Systems. In *Proceedings of the International Conference of Parallel Processing*, 2002.
- [8] W. Candler and R. Norton. Multilevel programming and development policy. Technical Report. Technical Report 258, World Bank Development Research Center, Washington D.C., 1977.
- [9] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of Symposium on Operating Systems Design and Implementation*, February 1999.
- [10] A. Charnes and W.W. Cooper. Goal programming and multiple objective optimization - part I. *European Journal of Operational Research*, 1:39–54, 1977.
- [11] A. Chien, S. Pakin, M. Lauria, M. Buchanan, K. Hane, L. Giannini, and J. Prusakova. Entropia: Architecture and performance of an enterprise desktop Grid system. *Journal of Parallel and Distributed Computing*, 63(5):597–610, 2003.
- [12] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, July 2003.
- [13] B. Chun, Y. Fu, and A. Vahdat. Bootstrapping a Distributed Computational Economy with Peer-to-Peer Bartering. In *Proceedings of First Workshop on Economics of Peer-to-Peer Systems*, 2003.
- [14] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, New York, 1991.
- [15] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In *Proceedings of Ninth ACM Conf. on Computer and Communications Security*, pages 207–216, November 2002.
- [16] Folding@home distributing computing project. <http://folding.stanford.edu>.
- [17] W. Du, J. Jia, M. Mangal, and M. Murugesan. Uncheatable Grid Computing. In *24th IEEE International Conference on Distributed Computing Systems*, pages 4–11, March 2004.
- [18] I. Foster and C. Kesselman, editors. *Grid2: Blueprint for a New Computing Infrastructure*. Morgan Kaufman, CA, USA, 2004.
- [19] Y. Fu, J. Chase, B. Chun, S. Schwab, and A. Vahdat. SHARP: An Architecture for Secure Resource Peering. In *Proceedings of the Nineteenth ACM Symposium on Operating System Principles*, 2003.

- [20] P. Golle and I. Mironov. Uncheatable Distributed Computations. In *Proceedings of CT-RSA*, April 2001.
- [21] P. Golle and Stuart G. Stubblebine. Secure Distributed Computing in a Commercial Environment. In *Proceedings of the 5th International Conference on Financial Cryptography*, pages 289–304, February 2002.
- [22] Sun Grid. <http://www.sun.com/service/sungrid/>.
- [23] R. Gupta and A. Somani. CompuP2P: An Architecture for Sharing of Compute Power in Peer-to-Peer Networks with Selfish Nodes. In *Proceedings of Second Workshop on Economics of Peer-to-Peer Systems*, 2004.
- [24] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.
- [25] S. Lee, R. Sherwood, and B. Bhattacharjee. Cooperative Peer Groups in NICE. In *Proceedings of IEEE Infocomm*, 2003.
- [26] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster Computing on the Fly: P2P Scheduling of Idle Cycles in the Internet. In *Proceedings of the IEEE Fourth International Conference on Peer-to-Peer Systems*, 2004.
- [27] D. Molnar. The SETI@Home problem. *ACM Crossroads*, September 2000.
- [28] C.K. Prahalad and V. Ramaswamy. *The Future of Competition: Co-Creating Unique Value with Customers*, chapter 2, page 26. Harvard Business School Press, Boston, Mass, 2004.
- [29] Condor project. <http://www.cs.wisc.edu/condor/>.
- [30] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara. Reputation Systems. *Communications of the ACM*, 43(12):45–48, 2000.
- [31] H. Rheingold. You got the Power. *Wired Magazine*, 8(8), August 2000. Also available at <http://www.wired.com/wired/archive/8.08/comcomp.html>.
- [32] L. F. G. Sarmenta. Sabotage-Tolerance Mechanisms for Volunteer Computing Systems. In *Proceedings of ACM/IEEE International Symposium on Cluster Computing and the Grid*, 2001.
- [33] P. Shread. Gateway offers Computing on Demand. [http://www.gridcomputingplanet.com/news/article.php/3281\\_1555061](http://www.gridcomputingplanet.com/news/article.php/3281_1555061), 2002.
- [34] J. Sonnek and J. Weissman. A Quantitative Comparison of Reputation Systems in the Grid. In *Proceedings of the Sixth ACM/IEEE International Workshop on Grid Computing*, 2005.
- [35] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. Wiley, New York, 1986.
- [36] S. Zhao and V. Lo. Result Verification and Trust-based Scheduling in Open Peer-to-Peer Cycle Sharing Systems. In *IEEE Fifth International Conference on Peer-to-Peer Systems*, September 2005.