Reputation-Based Scheduling on Unreliable Distributed Infrastructures

Jason Sonnek, Mukesh Nathan, Abhishek Chandra and Jon Weissman
Department of Computer Science and Engineering
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. 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.

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 [19, 7, 8, 2, 3]. Despite the attractive features of these platforms, 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. An example of such an environment is BOINC [2], or its forerunner SETI@home [3], 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.

Task replication and voting [5] is a common technique used to deal with uncertainty in the absence of inexpensive verifiers. 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. Using a simple reputation system [18], it is possible to determine the likelihood that a given worker will return a correct and timely result with fairly high accuracy. Using these reliability ratings, the system can intelligently schedule tasks to workers such that throughput 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. 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. Through simulation of a BOINC-like distributed computing infrastructure, we compare the throughput and computational overhead of each of these techniques. Our results indicate that reputation-based scheduling can significantly improve the throughput of the system for worker populations modeling several real-world scenarios, with overhead that scales well with system size.

Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)
0-7695-2540-7/06 $20.00 © 2006 IEEE

2 Background and Related Work

Distributed Computing Infrastructures

Numerous computing infrastructures have been designed to utilize idle distributed resources. The @Home applications [3, 10] and their generalization, BOINC [2], are instances of a growing number of systems which utilize donated computing cycles to solve massive scientific problems. In contrast to BOINC, several unstructured cycle-sharing platforms have been proposed [6, 14] 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.

Dealing with Unreliability

Dealing with unreliability is a core design challenge in any distributed system. Redundant task allocation combined with voting, as used in Byzantine fault-tolerant (BFT) systems [5], is popular due to its general applicability. This approach is also used by most BOINC [2] applications to verify the results of outsourced computations. 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 [12, 9] 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. 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 [16, 20] employs pre-computed tasks called 'quizzes' that are embedded into a batch of (otherwise indistinguishable) tasks allocated to a worker. This technique requires pre-computation of certain tasks, which may be non-trivial or infeasible in many scenarios.

Reputation-Based Scheduling

Reputation systems [15] are commonly used to gauge the reliability of nodes based on past interactions. The concept of trust-aware resource management for the Grid was proposed in [4], where a technique is presented for computing trust ratings in a Grid using a weighted combination of past experience and reputation. GridEigenTrust [1] combines this trust-computation technique with the EigenTrust reputation system [13] to provide a mechanism for rating resources in a Grid. Zhao and Lo [20] propose augmenting peer-to-peer cycle sharing systems with a reputation system to reduce the degree of replication required to verify results.

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 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

Our distributed computing model consists of a central server that assigns computational tasks to a set of worker nodes. 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 [7], or they may be providing their resources in return for some incentive, such as monetary remuneration [19], credit [3, 2], or use of other nodes' resources in return [11]. 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.

3.1 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, 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.

3.2 Redundant Computation

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 [12, 9] have been proposed to verify the correctness of results, these techniques are application-specific and are not applicable to general computational scenarios. In our system model, we employ a verification technique based on *redundant computation* coupled with *voting*. Under this verification technique, if a quorum of workers agrees on a result, the server treats that result to be correct; otherwise, the task is re-scheduled. While the quorum size could be application-dependent, *majority* is typically used to determine the correct answer.

3.3 Definitions

Definition 1 Task (τ_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.

Definition 2 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.

Definition 3 Redundancy Group[1](G_i): *Redundancy group for a task τ_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. 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 4 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.

Definition 5 Likelihood-of-Correctness (λ_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 LOC λ_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.

4 Reputation-based Scheduling

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 workers.
- Using the estimated worker reliability ratings to compute the LOC of possible groups.
- Grouping workers for task assignment based on LOC estimates to maximize the throughput and success rate.

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. 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. 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).

If the workers in a group reach a quorum, the server accepts the majority answer as the "correct" result, and updates the reliability ratings accordingly. 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.
- *Pessimistic:* If a group does not achieve a quorum, all of its workers are given negative ratings.
- *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.
- *Verification-based:* This heuristic verifies results using an independent verifier. We use this heuristic is an upper bound, since it relies on perfect knowledge of the results' correctness.

[1] In the rest of the paper, we would refer to a redundancy group simply as a *group* unless required to avoid confusion.

4.2 Computing the LOC

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 $G = \{w_1, \ldots, w_{2k+1}\}$ consisting of workers $w_i, i = 1 \ldots 2k + 1$[2]. Let r_i be the reliability rating of a worker $w_i, i = 1 \ldots 2k + 1$, at a given point in time. Then, the LOC λ of the group G is given by:

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

where $\epsilon = \{\epsilon_1, \ldots, \epsilon_{2k+1}\}$ is a vector of responses from the workers in the group, with 1 representing a correct response, and 0 representing an incorrect response. 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. We assume 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.

The complexity of computing the λ value can be shown to be $O(2^{2k})$, which is infeasible for most practical purposes. To reduce the cost of computing λ values for multiple groups, we use a lower bound λ^{lb} for λ based on Jensen's inequality that has an $O(n^2)$ computation complexity [17].

4.2.1 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 λ_{target} such that if $\lambda \geq \lambda_{target}$, then we conclude that G is an acceptable group. We refer to λ_{target} as the *target LOC*. Since λ_{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.

Choosing a proper value for λ_{target} is critical to maximizing the benefit derived from the system. If λ_{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 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.

We now present several reputation-based scheduling algorithms that form groups G_i from the pool of available worker nodes such that λ_i of each G_i satisfies λ_{target}.

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.

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 λ_{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 a sufficient number of reliable workers.

[2] We consider odd-sized groups to avoid ambiguity in defining majority for even-sized groups.

4.3.3 Best-Fit

The Best-fit algorithm attempts to form groups G_i such that λ_i is as close as possible to λ_{target}. The Best-fit algorithm searches the space of available workers and groupings to find a G_i that exceeds λ_{target} by the smallest possible margin. If no group of size R_{max} or smaller meets λ_{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 λ_{target}. As a result, tasks are not overprovisioned with more reliable resources than necessary, and well-balanced groups are formed.

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 λ_i meets λ_{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.

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. Finally, we briefly consider the overhead introduced by the proposed scheduling algorithms.

5.1 Evaluation Methodology

Our evaluation is based on a simulator loosely modeled around the BOINC [2] distributed computing infrastructure, which consists of a task server and some number of worker machines. 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 reforms groups for task allocation in the next round. In the results shown here, we ran our simulations for a total of 1000 rounds each.

Name	Distribution	Real-world Scenario
Uniform	Uniform	General environment
Heavy-tail-high	1-Pareto(1, 0.1)	Most workers reliable
Heavy-tail-low	Pareto(1, 0.2)	Most workers unreliable
Normal-high	N(0.9, 0.05)	Reliable environment
Bimodal	N(0.2,0.1) + N(0.8,0.1)	50% (un)reliable workers
Normal-low	N(0.3, 0.1)	Hostile environment

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 in Table 1, 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

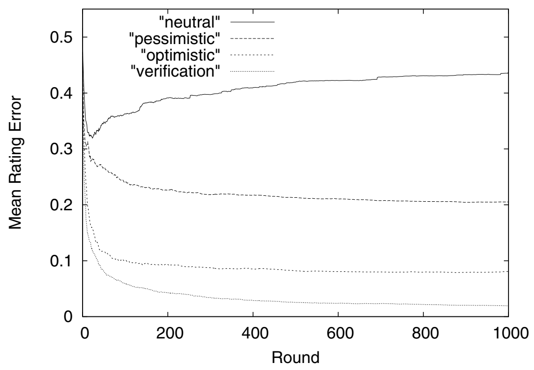

Figure 1. Mean Rating Error - Bimodal Distribution

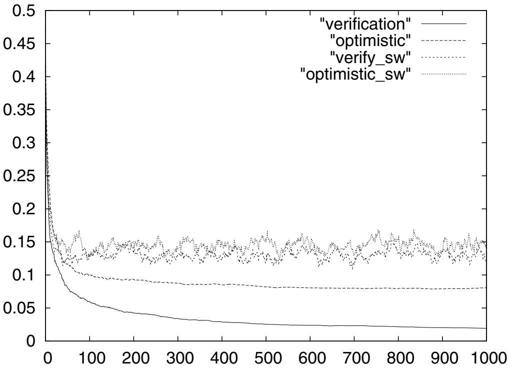

Figure 2. Rating Error Comparison: Sliding-Window vs. Infinite History

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 $|actual\_rating - estimated\_rating|$.

Figure 1 shows the results obtained for simulations using the bimodal distribution. 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. 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%). This is 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.

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. 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. 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 2, 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).

5.3 Reputation-Based Scheduling

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

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

To fully understand the behavior of the reputation-based schedulers, we ran an exhaustive set of simulations covering a large parameter space. Due to space constraints, we will present a subset of the results here. A full analysis can be found in [17].

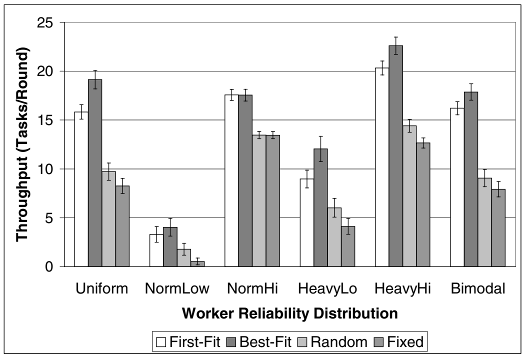

Figure 3. Algorithm Comparison - Throughput

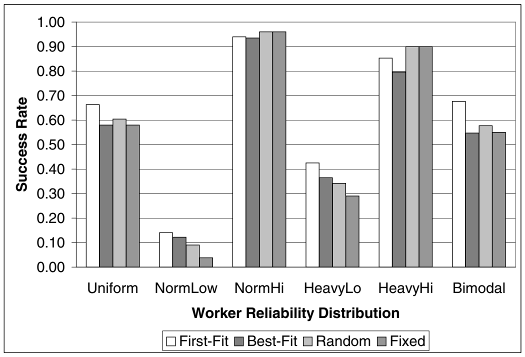

Figure 4. Algorithm Comparison - Success Rate

For a given distribution , we set λ_{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.

5.3.1 Comparing Scheduling Algorithms

In our first experiment, we measured the mean throughput, 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 3, we present the throughput results for a maximum group size 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%.

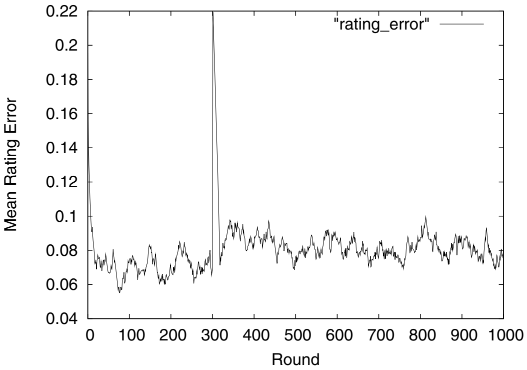

Figure 5. Large-scale Blackout: Rating error using window-based Optimistic rating heuristic

Figure 4 plots the success rate results for the experiment. Since we set λ_{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, which causes First-fit to overprovision reliable clients to the first several groups. 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.

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 are likely to exhibit non-stationary behavior, i.e., their reliability will vary with time.

We consider a large-scale worker blackout scenario that 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 300 rounds.

Figure 5 shows the rating error for a simulation using the Best-fit algorithm. In Figure 5, 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.

5.5 Overhead

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.

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.

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. 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. The 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 , with overhead that scales well with system size.

References

[1] B. Alunkal, I. Veljkovic, G. von Laszewski, and K. Amin. Reputation-Based Grid Resource Selection. In *Workshop on Adaptive Grid Middleware*, 2003.

[2] D. Anderson. BOINC: A System for Public-Resource Compting and Storage. In *Proceedings of the 5th ACM/IEEE International Workshop on Grid Computing*, 2004.

[3] 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.

[4] F. Azzedin and M. Maheswaran. Integrating Trust into Grid Resource Management Systems. In *Proceedings of the International Conference of Parallel Processing*, 2002.

[5] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of Symposium on Operating Systems Design and Implementation*, Feb. 1999.

[6] 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.

[7] 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.

[8] Condor project. http://www.cs.wisc.edu/condor/.

[9] 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.

[10] Folding@home distributing computing project. http://folding.stanford.edu.

[11] 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.

[12] P. Golle and I. Mironov. Uncheatable Distributed Computations. In *Proceedings of CT-RSA*, Apr. 2001.

[13] 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.

[14] 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.

[15] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara. Reputation Systems. *Communications of the ACM*, 43(12):45–48, 2000.

[16] 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.

[17] 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.

[18] 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.

[19] Sun grid. http://www.sun.com/service/sungrid/.

[20] 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*, Sept. 2005.