Exploiting Task Delays for Uncheatable Grid Computing

Wenliang Du* Michael T. Goodrich† Ankur Kasturiya‡

Abstract

We study the problem of protecting distributed grid computations from the abuses of "lazy" participants, who wish to be rewarded for tasks they receive but never actually perform. Unlike previous techniques, which rely heavily on having the computation's supervisor perform a significant amount of the work or inefficiently replicate the tasks, our approach has low computational overhead for the supervisor and the participants. We achieve this improvement by utilizing a new paradigm in uncheatable grid computing—exploiting the delays that exist between various rounds in a grid computation. In addition, the main idea of one of our utilizations of this paraidgm might at first seem counter-intuitive—we have the participants check each other's work. A naive implementation of this approach would, of course, be susceptible to collusion attacks, but we show that by, adapting efficient solutions to the parallel processor diagnosis problem, we can tolerate collusions of lazy cheaters, even if the number of such cheaters is a fraction of the total number of participants. In addition to introducing this new framework for uncheatable grid computing, we also introduce a number of new checking methods that can be used to verify if participants do all of their work.

1 Introduction

One of the great success stories of modern distributed processing is grid computing, where a supervisor distributes a large set of parallel tasks to a community of participants, who perform those tasks on behalf of the supervisor and send back the results. Examples include SETI@home [1], which claims over five million participants who have collectively performed over 1.5 billion tasks aimed at finding intelligent patterns in extraterrestrial signals, and distributed.net, which claims over 300,000 participants who have collectively searched over 68 billion blocks of cipher-text to perform brute-force inversions of cryptographic algorithms.

The participants in grid computing environments are typically rewarded with credits, points, cybercash, or some other kind of recognition. For example, SETI@home posts the pseudonyms of the top 1000 users and distributed.net lists on a daily basis the top 100 participants and teams. Unfortunately, even with modest recognitions such as these, grid computations must deal with cheating users. Indeed,

*Department of Electrical Engineering and Computer Science, Syracuse University. Email: `wedu(at)ecs.syr.edu`.

†Department of Computer Science, University of California, Irvine. Email: `goodrich(at)acm.org`.

‡Department of Electrical Engineering and Computer Science, Syracuse University. Email: `akasturi(at)mailbox.syr.edu`.

the team leader for SETI@home is quoted [9] as saying that their project spends half of their resources dealing with cheaters, who comprise roughly 1% of their users. For example, he mentioned that some users have modified the SETI@home software to make it look like they have performed more work than they actually did. The problem of dealing with such cheating users becomes even more serious, of course, when the rewards for participation become more tangible.

This paper is therefore directed at the problem of protecting grid computations from cheating users who wish to be rewarded for tasks they do not perform (or perform incompletely). This problem, known as uncheatable grid computing, has been studied recently by several researchers [5–7, 12], and we review their contributions in the next section. Roughly speaking, the previous approaches have relied either on replicating tasks, which adds a significant multiplicative overhead for the participants (SETI@home uses this technique), or on having the supervisor check computations himself, which requires the supervisor to perform a significant fraction of the tasks.

Our approach takes a radical departure from the previous approaches, in that it is based on a new paradigm, which has to date not been utilized in mechanisms for dealing with lazy cheaters—we exploit the delays that are inherantly a part of grid computing environments. Specifically, in this paper, we show how to exploit grid-computing delays as follows: (i) we propose a parallel fault diagnosis approach, which allows participants to verify each other over a series of rounds in the grid computation; (ii) we present an improved redundancy scheme to achieve a high degree of resilience against collusion.

By utilizing the new paradigm of exploiting grid delays, we are able to simultaneously achieve improved security and lower overheads for the computation supervisor. This approach matches paradigms established in program verification, for example, since verifying a computation is much more efficient than performing the computation itself. For instance, in the Hamiltonian cycle problem, verifying whether a cycle is a Hamiltonian cycle is much faster than finding a Hamiltonian cycle. For these computations, replicating tasks to defeat cheating is a waste of CPU cycles. The old paradigm for grid computation verification is that the supervisor conducts all the verification. Although each such verification might be efficient, the overall cost might be very high because the supervisor must verify the results from all computations of all the participants. To reduce the overhead for the supervisor in cases where verification costs are significant, we apply a paradigm that might at first seem counter-intuitive—we have the participants check each other's work. Nevertheless, by adapting solutions to parallel fault diagnosis, we show that we can efficiently tolerate up to a constant fraction of cheating users, even if these users collude. To implement this approach, we take advantage of a feature of grid computing that was not utilized in previous work on uncheatable grid computing: namely, that massive grid computations are performed in a pipelined fashion in a series of rounds. Thus, provided that our checking algorithms use few rounds, we can mix in the checking steps for previous tasks with the specifications of new tasks, thereby achieving low computational overheads for all parties. The challenge, of course, is coming up with schemes that use few rounds (ours use fewer than 10).

For computations whose verification is as expensive as redoing the computations, the parallel fault diagnosis approach might be less efficient than the simple redundancy approach, namely assigning multiple copies of a same task to different participants. Unfortunately, redundancy approaches are subject to collusion attacks. That is, if a group of cheating participants are assigned all the copies of the same task, they can still cheat on this task. But, here again, the new approach of utilizing delays can improve security and efficiency, for we propose a delay mechanism to significantly reduce the damage caused by collusion, thereby using pipelining to improve the efficiency of the redundancy approach.

2 Related Work

As mentioned above, there is a growing interest in solutions to uncheatable grid computing [5–7, 12], where the challenge is to construct grid computing protocols that defend against participants claiming credits for the work they have not done.

For defeating such "lazy" cheaters, Golle and Mironov proposed an interesting *ringer* scheme [6], which is restricted to problems that involve the inversion of a one-way function (IOWF). In their basic ringer scheme, during the initialization stage, the supervisor randomly selects several inputs x_i from a participant's domain D. These inputs are called *ringers*, and they are kept secret with the supervisor. The supervisor then computes *ringer images* for each x_i, that is, evaluations of the one-way function, $f(x_i)$, on these inputs. Then, in addition to assigning the grid computing tasks in D, the supervisor also assigns the ringer images to the participant. Not only does the participant need to compute f on x for all $x \in D$ and return the results of interest, she also has to return all the ringers corresponding to the pre-computed ringer images. The supervisor then verifies whether the participant has found all the ringers assigned to her or not. If yes, then the supervisor is assured with high probability that the participant has indeed conducted all the computations. Golle and Mironov augment their basic ringer scheme in several ways, including the introduction of bogus-ringers, which can prevent the participant from knowing the total number of ringers planted and magic sets, which can prevent direct images from being sent to the participant. Each of these new schemes adds a new level of security, but they are still restricted to IOWF computations.

Szada, Lawson, and Owen extend the ringer scheme to deal with other general classes of computations, including optimization and Monte Carlo simulations [12]. They propose effective ways to choose ringers for those computations. It is still unknown whether the schemes proposed in [12] can be extended further to generic computations, however, as their constructions are not generic.

Du *et al.* [5] propose a different approach to achieve uncheatable grid computing. In their scheme, the supervisor randomly selects and verifies some samples from the task domain D assigned to a participant. To prevent the participant from cheating on those samples, the participant must commit her computation results before being checked. A commitment-based sampling scheme based on Merkle tree is proposed in [5]. This approach has the advantage of being generic, but it places a significant computational burden on the supervisor in addition to his management role, since in this scheme he must redo some of each participant's tasks himself.

Finally, another approach for uncheatable grid computing is to double-check computation results by simply having other participants redo them entirely. This is the method used by SETI@home, for example, to deal with cheating users [9]. Golle and Stubblebine [7] proposed a more efficient scheme based on using probabilistic redundant execution to implement this approach. Their scheme assumes that all computations occur in a single round, however; hence, they do not take advantage of additional improvements that might be possible using pipelining.

There are also some alternative ways to defeat cheating, which have not been specifically addressed in the literature. For example, one method would be to use tamper-resistant software. In this case, a code obfuscater would be used to convert task programs to equivalent programs that are much harder to understand and reverse-engineer. Thus, it would become hard for malicious attackers to modify the program to accomplish what he wants. Nevertheless, the tamper-resistant approach is only heuristically secure, and many tamper-resistant schemes cannot withstand attacks from really determined attackers [10], including groups of colluding cheaters.

The problem of uncheatable grid computing is close to another body of literature: the problem of dealing with malicious hosts in the study of mobile agents [14, 17]. Several practical solutions have been proposed for this related problem, which include remote auditing [10, 13], code obfuscation with timing constraints [8], computing with encrypted functions [11], replication and voting [16]. The major difference between the mobile-agent work and our grid-computing work is the threat model. The mobile-agent work assumes a general, malicious cheater—a malicious host can do whatever it takes to cheat, including spending more CPU cycles than the honest behavior. In uncheatable grid computing, however, we are presented with a different threat model, since it is irrational for a "lazy" participant in a grid computation to cheat with a cost more expensive than the honest behavior.

In addition to the methods cited above, there are cryptographic protocols, such as Private Information Retrieval (PIR) [4] and Probabilistically Checkable Proofs (PCP) [15], that can also be used to achieve uncheatable grid computing. Although such heavy machinery can provide possible theoretical uncheatable grid computing constructions, their expensive computation costs make them inappropriate choices for grid computing in practice.

3 Problem Definition

We consider a grid computing model in which *untrusted participants* are taking part. The computation is organized by a *supervisor*. Formally, such computations are defined in our model by the following elements [5]:

- **A task function $f : X \mapsto T$ defined on a finite domain X.** The goal of the computation is to evaluate f on all $x \in X$. For the purpose of distributing the computation, the supervisor partitions X into subsets. The evaluation of f on subset D is assigned to participant i.

- **A screener function S.** The screener is a function that takes as input a pair of the form $(f(x); y)$ for $x \in X$, and returns a string $s = S(f(x); y)$, where y represents the criterion. S is intended to screen for "valuable" outputs of f that are reported to the supervisor by means of the string s.

In addition, in this paper we assume the existence of the following:

- **A checker function C.** The checker is a function that takes as input a string s that was returned from the screener function applied to each element x in the domain D for some participant. It also takes as input a description of D and a certificate z that provides additional information on how the values in s should be tested (e.g., z could be the set of ringers for D). It returns a good/bad label for the input task.

If the checker function is light-weighted, it can be evaluated at the supervisor side; however, if the checker function is also computation intensive, we might also want to out-source it to the grid participants. In this paper, we have provided solutions to address both situations.

3.1 Models of Cheating

A participant can choose to cheat for a variety of reasons. We categorize the cheating using the following. We assume that each participant is given a domain $D \subset X$, and her task is to compute $f(x)$ for all $x \in D$. From now on, we use D as the domain of f for the participant.

- *Lazy Cheating Model:* In this model, the participant follows the supervisor's computations with one exception: for $x \in \check{D} \subset D$, the participant uses $\check{f}(x)$ as the result of $f(x)$. Function $\check{f}$ is usually much less expensive than function f; for instance, $\check{f}$ can be a random guess. In other words, the participant does not compute the required function f on inputs $x \in \check{D}$. The goal of the cheating participant in this model is to reduce the amount of computations, such that it can maximize its gain by "performing" more tasks during the same period of time. If the participants are getting paid, the cheating participant might be guided by the lure of money. This type of cheating behavior is a cheating on the task function f.

To maximize their gains, rational cheaters tend to use minimal cost to falsify the contributions they have never made. Their behaviors fall into the lazy cheating model. In this paper, we focus exclusively on detecting the *lazy cheating* participants.

4 Checking Grid Computations in Pipelined Rounds

In this section, we describe how to extend the parallel fault diagnosis approach of Beigel *et al.* [2, 3] to uncheatable grid computing. Throughout this section, we assume that tasks can be provided with a checker function for other tasks, which is computationally easier than performing the task itself.

In the parallel fault diagnosis problem, we are given a set of n processors, each of which is either "good" or "bad." In a single round, a processor can test another processor or be tested itself by another processor. If a good processor tests another processor, it correctly returns to a central supervisor whether the tested processor is good or bad. On the other hand, if a bad processor tests another processor, it returns an arbitrary (or even deliberately false) identification of the other processor as being good or bad. The fault diagnosis problem is to determine all the good and bad processors using a small number of parallel testing rounds, assuming that there are fewer than $n/2$ bad processors (the problem cannot be solved if there are more than this many bad processors).

We can easily adapt any solution to the parallel fault diagnosis problem to the problem of checking the results from a grid computation performed by n participants. After a preprocessing round, which distributes the tasks to the n participants and receives their results, we can then simulate the parallel fault tolerance algorithm by replacing each test of a processor i by a processor j with a test of the task of participant i by a participant j using the checker function for this task. For example, the 10-round algorithm of Beigel *et al.* [2] immediately translates into a 10-round checking algorithm (after the preprocessing round that commits the results of the tasks).

There are some improvements that we can make to this approach, however, for uncheatable grid computing. First, in any round, we can allow a participant j to test another task i even if j's task is also being tested in that round (such tests are not allowed in the parallel fault diagnosis problem). This observation lets us immediately reduce the number of rounds in our simulation to 9, since the first round in the algorithm of Beigel *et al.* involves the symmetric testing of $n/2$ pairs of processors (which requires two rounds in their algorithm but only one in our simulation). Even so, this simulation algorithm is probabilistic and only guaranteed to succeed with high probability if n is very large. Thus, we would like to reduce further the number of rounds and make this algorithm more practical. To achieve these goals, let us make a simplifying assumption, which is well-motivated for grid computing but not for parallel fault diagnosis: namely, let us assume that the number of cheating participants is much less than $n/2$. The motivation for this assumption is that the pipelined nature of grid computing allows the

supervisor to prune away cheating participants as soon as they are discovered (by our testing algorithm); hence, it is unlikely for large numbers of cheaters to be in the grid (since most of them would have to be recent joiners).

To design an efficient grid computing testing scheme, we utilize the following lemma from Beigel *et al.*

Lemma 1 ([2]) *Let $G = (V, E)$ be a graph on n vertices. Let $0 < \lambda, \gamma < 1$. Suppose, for every pair of subsets A and B of V such that $A \cap B = \emptyset$, $|A| + |B| = \lambda n$, and $|A|, |B| \leq \frac{1+\gamma}{2}\lambda n$, that there are edges in E directed from A to B and B to A. Then G induces a strongly connected component of size $\gamma\lambda n$ on any subgraph with λn vertices.*

This lemma may at first seem obscure, but it is useful for proving the following theorem, which extends a theorem from the earlier work of Beigel *et al.* [3].

Theorem 1 *Let V be a set of n vertices, and let $0 < \gamma, \lambda < 1$. Let $H_d = (V, E)$ be a graph defined by the union of d independent randomly-chosen Hamiltonian paths on V (with all such cycles equally likely). Then, for all subsets W of V of λn vertices, H_d induces at least one strongly connected component on W of size greater than $\gamma\lambda n$, with probability at least*

$$1 - e^{n[(1+\lambda)\ln 2 + d(\alpha \ln \alpha + \beta \ln \beta - (1-\lambda)\ln(1-\lambda))] + O(1)},$$

where $\alpha = 1 - \frac{1-\gamma}{2}\lambda$ and $\beta = 1 - \frac{1+\gamma}{2}\lambda$.

Proof. The proof is an adaptation and correction of a proof of a weaker theorem from Beigel *et al.* [3]. By Lemma 1, it is sufficient to show that with the exponentially small probability mentioned in Theorem 1, there is a subset W of V of size λn that has a partition (A, B), with $|A|, |B| \leq \frac{1+\gamma}{2}\lambda n$, such that there is no edge from A or B or no edge from B to A. Let us consider first the probability that there is no edge from A to B (as the other case is identical). Beigel *et al.* [3] show that, for a single randomly-chosen Hamiltonian path H on V (and two disjoint subsets A and B of V), the probability that there is no edge from A to B is

$$\frac{(n-|A|)!(n-|B|)!}{n!(n-|A|-|B|)!}.$$

Thus, the probability that there is no edge from A to B or no edge from B to A in H_d is at most

$$2\left(\frac{(n-|A|)!(n-|B|)!}{n!(n-|A|-|B|)!}\right)^d.$$

There are at most 2^n choices for W and at most $2^{\lambda n}$ possible ways of partitioning W into subsets A and B (actually, there are fewer, but these bounds will suffice for our purposes). Thus, the probability that there is a subset W of λn vertices that has a partition (A, B), with $|A|, |B| \leq \frac{1+\gamma}{2}\lambda n$, such that there is no edge from A or B or no edge from B to A is at most

$$2^{(1+\lambda)n+1}\left(\frac{(n-|A|)!(n-|B|)!}{n!(n-|A|-|B|)!}\right)^d.$$

This is maximized when $n - |A| = \alpha n$ and $n - |B| = \beta n$. Applying Stirling's formula, we can bound this probability by

$$e^{n[(1+\lambda)\ln 2 + d(\alpha \ln \alpha + \beta \ln \beta - (1-\lambda)\ln(1-\lambda))] + O(1)}.$$

∎

Applying this to our testing problem, we need to set the parameters so that we are guaranteed to have at least λn good testers, for using a graph H_d for sufficiently large d will guarantee with high probability that the subgraph of good testers will induce a strongly connected component of size at least $\gamma\lambda n$. For example, we have the following:

Corollary 1 *If $n \geq 20$, $\gamma = 1/2$, $\lambda = 7/8$, and $d = 8$, then any subset of $(7/8)n$ vertices induces a strongly connected subgraph of H_d of size $(7/16)n$ with probability at least*

$$1 - e^{-n}.$$

That is, it is very unlikely that we will not have a strongly connected subgraph of the required size. If we can safely assume that the number of cheating participants is at most 10% of the total (which is ten times higher than the SETI@home experience), then we can use this corollary to design the following five-round testing strategy:

1. Pair up participants and have them test each other. Discard for now any pairs that have an identified bad test (for one of them must be bad). The remaining pairs must each consist of two good participants or two bad ones.

2. Pair up pairs of participants from the first round and have them test each other with one test per participant. Discard for now any groups that have an identified bad test (for two of the four must be bad).

3. Pair up groups of participants from the previous round and have them test each other. Discard for now any super-groups that have an identified bad test (since four of the eight must be bad). Let N be the number of super-groups.

4. Note that there can be at most $n/5$ discarded nodes; hence, $N \geq (n - n/5)/8 = n/10$. Moreover, since each super-group has all good nodes or all bad nodes, the number of all bad super-groups is at most $n/80 \leq N/8$. Apply the H_8 strategy to the super-groups, using Corollary 1. This results in a strongly connected component (which must be all good nodes) of size at least $(7/16)N$, which is at least $(7/16)(n/10)8 = (7/20)n$. Moreover, there are at most $(9/16)N$ unresolved super-groups, which is at most $(9/128)n$ super-groups.

5. Split the $(7/20)n$ proven good nodes into two groups: one group of size $n/5 = (4/20)n$, which is sufficient to have each test a discarded node, and another group of size $(3/20)n$, which is sufficient to have each test a representative member of an unresolved super-group (since $3/20 = 9/60 > 9/128$).

Thus, we can test the n results from our participants in five rounds, even if 10% of them are colluding lazy cheaters. If we can further assume that at most 5% of our participants are cheaters, then we can further improve our protocol to just have three rounds (the above method would require at least 4 rounds even in this case). Our method is based on an adaptation and improvement of a protocol of Beigel *et al.* [2]. Our algorithm is also based on the following corollary to Thereom 1.

Corollary 2 *If $\gamma = 7/15$, $\lambda = 15/16$, $d = 4$, and $n \geq 20$, then any subgraph of $(15/16)n$ vertices of a random graph H_d induces a strongly connected component of size $(7/16)n$ with probability at least*

$$1 - d^{-n/4}.$$

We use this corollary to prove the following protocol works correctly with high probability.

1. Divide the input nodes into $n/4$ directed cycles of 4 nodes each, and in one round have each node test the next one in the cycle. Discard for now all the nodes in any cycle with a detected bad node (there can be at most $4n/20 = n/5$ such discarded nodes. Let $N \geq (n - n/5)/4 = n/5$ denote the remaining cycles, which we view as super-nodes, since all the remaining cycles each consist entirely of good nodes or of bad nodes.

2. There can be at most $(n/20)/4 = n/80$ such all-bad cycles, that is, at most $5N/80 = N/16$ bad super-nodes. Apply the tests dictated by a random H_4 in one round, viewing the N super-nodes as the nodes in H_4. Note that the number of bad super-nodes is small enough for us to apply Corollary 2 to show that, with high probability, there will be a strongly connected component of $(7/16)N$ super-nodes (which must necessarily be all good). That is, we will have at most $(9/16)N \leq (9/16)n/4 = (9/64)n$ super-nodes whose classification may be in doubt.

3. Note that at this point there are at least $(7/16)4N \geq (7/4)n/5 = (7/20)n$ identified good nodes. Divide these nodes into two groups: one group of size $n/5 = (4/20)n$ nodes, which are used in one round to test the previously-discarded nodes, and another group of $(3/20)n = (9/60)n$ nodes, each of which can test a single super-node in one round (since $9/60 > 9/64$).

Thus, we can test all the n tasks in three rounds if there are at most 5% bad participants (which is five times the SETI@home experience), assuming we have an effective checker function. We discuss some possible checker functions below.

5 A Redundancy Scheme with Delays

In this section, we describe how to use pipelining to improve the efficiency of redundancy-based uncheatable grid computing schemes.

5.1 The Naive Redundancy Approach

The naive redundancy approach is to replicate these n tasks w times. Then the supervisor randomly distributes and assigns these $w \cdot n$ tasks to participants[1]. Golle and Stubblebine studied this naive redundancy approach in [7]. We briefly review their results of the naive redundancy approach. The goal of this paper is to improve this approach and further reduce the number of corrupted tasks.

Assume there are m participants and k of them are colluding. We say that a task is *corrupted* if all the copies of this task is assigned to the colluding group. Let X be a random variable representing the number of corrupted tasks. Let $p = \frac{k}{m}$, and p is the probability that a copy of task is assigned to these k colluding participants. Therefore p^w is the probability that all the w copies of a task is assigned to the

[1] It is better to make sure that none participant is assigned more than one copies of the same task. However, with such a constraint, the analysis becomes much more complicated. Therefore, we will not include this constraint in our analysis.

colluding participants. Hence, the probability ($Pr(X = a)$) that exactly a tasks are assigned to these colluding participants is the following:

$$Pr(X = a) = \binom{n}{a} p^{w \cdot a}(1 - p^w)^{n-a}. \tag{1}$$

Since $Pr(X = a)$ is the binomial distribution with the parameter n and p^w, the expected value is

$$\text{Expected Damage} = np^w = n(\frac{k}{m})^w.$$

To simplify the discussion, we let $w = 2$ in the rest of discussion, i.e., each task will be conducted by two different participants. Our results can be generalized to $w \geq 2$.

5.2 The Scheme with Delays

To reduce the damage, the supervisor does not assign both copies of a task at the same time, he delays the assignment of the second copies until some participants have submitted their tasks. The task assignment is divided into the following two stages:

1. During the first stage the first copies of the tasks are assigned to the participants. The colluding group cannot corrupt any task during this stage, because only one copy of all the tasks is assigned, and the participants do not have any common tasks. If a group of colluders decide to cheat on some task, they can succeed only if the second copy of the same task are also assigned to them. The probability of succeeding in cheating on x tasks is p^x. Since p is usually small, a large x makes the probability approaching zero.

2. In the second stage the supervisor assigns the second copies of tasks to only those participants who have completed the assigned tasks and who have already sent back the results. The supervisor does not assign the second set of tasks as soon as the participant sends the results. The supervisor waits for d participants to complete their first set of tasks and then assigns the second set of tasks to these d participants. This d parameter is known as the delay parameter. After the first delay, i.e., the supervisor has assigned the first set of the second copy of the tasks to the first group of d participants, and the supervisor again waits for next group of d participants to submit their results.

Thus, to reduce the cheating damage, the supervisor can increase the delay parameter. However, the bigger the d is, the more time participants needs to spend on waiting. That is, there is a tradeoff. We study the relationship between d and the damage of cheating below.

5.3 Analysis

We can analyze the probability of cheating of the above mentioned scheme. We assume that the total number of the participants are m; the number of participants colluding in these m participants is k; each participant is assigned same amount of tasks. We also assume that attackers will only cheat if they see both copies of a task, i.e., they will not gamble.

Theorem 2 *If the attackers can only get the second copies after at least d of them have submitted their first copies, the expected damage for the collusion of k participants is the following:*

$$\textit{Expected Damage} = \frac{k(k-d)n}{2m^2} \tag{2}$$

Proof. During the first stage, the probability of a task being assigned to the colluding participants is $p_1 = \frac{k}{m}$, as we have a group of k colluders among the m participants.

Suppose we have total t delay periods, $D_1, \ldots, D_t$, in the complete assignment of the second copy of all the tasks, i.e. $k = t * d$. The second copy of a particular task could be assigned at each of these t delay periods, with the same probability $\frac{1}{t}$.

Assume that the second copy of a task U is selected to be assigned in the delay periods D_i. Let $P(D_i)$ represents the probability that U can be corrupted given that the first copy of U is already assigned to the colluding group and the second copy is assigned at the delay period D_i. In the naive scheme, once both copies are assigned to the colluders, this task is corrupted. However, in our scheme, the first copy of task U might have already been finished because it might belong to those $i \cdot d$ participants who have already submitted their results. In this situation, the colluders cannot cheat on U based on our non-gambling assumption[2]. Therefore, U must belong to those un-submitted tasks that are assigned to the colluding group, its probability is

$$P(D_i) = \frac{k - i \cdot d}{m}.$$

Since the second copy of task U can be assigned at any delay period D_i, for $i = 1, \ldots, t$ with the same probability, the total probability that U can be corrupted given that the first copy of U is already assigned to the colluding group is

$$\begin{aligned} p_2 &= \frac{1}{t}(P(D_1) + \ldots + P(D_t)) \\ &= \frac{1}{t}(\frac{k-d}{m} + \ldots + \frac{k-td}{m}). \\ &= \frac{k-d}{2m}. \end{aligned}$$

The probability for a particular event to get corrupted would then be the following:

$$p = p_1 * p_2 = \frac{k(k-d)}{2m^2}.$$

Therefore, the expected damage is

$$\text{Expected Damage} = np = \frac{nk(k-d)}{2m^2}.$$

∎

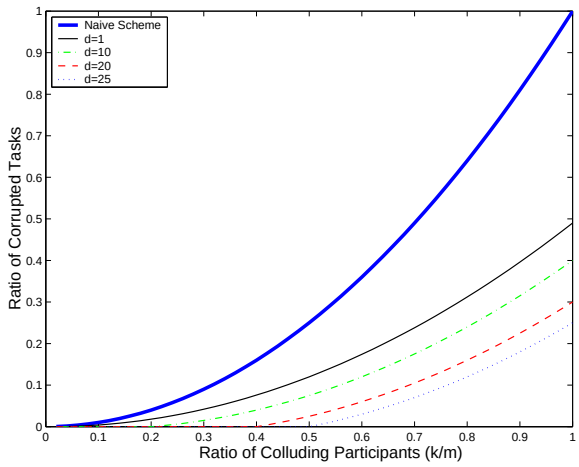

Figure 1. Expected Damage ($n = 1000$ and $m = 50$)

We compare the results of our scheme with the naive redundancy scheme in Figure 1. Our scheme improves the naive redundancy scheme. Even with $d = 1$, the expected damage is cut to half compared to the naive scheme. This is because we are decreasing the number of colluders who can corrupt tasks after each delay.

From the figure, we know the larger the value of d, the fewer the number of tasks that can be corrupted. However, in practice, making the delay exactly d is not easy, because the supervisor cannot distinguish who is a cheater and who is not. For the supervisor, it conducts a new round of task assignment once it receives accomplished tasks from d participants. If these d participants are all honest, then this assignment is safe. However, it is possible that only one of the participants is cheater; this means, to the cheaters, the effective delay period is just 1, not d. Therefore, to maximize their damage (i.e., forcing the effective delay to be 1), the cheaters can adopt a slow strategy, hoping the honest participants can contribute to the other $d - 1$.

However, the economic gain of the above strategy is undesirable. A participant's economic gain is decided by the number of tasks it can perform for the supervisor. If in order to cheat, a participant lowers its task processing rate, the amount of economic gains from the same supervisor is reduced. This might contradict to the motivation of the cheating. Furthermore, the supervisor can rank the participants based on their response time. Such ranking can be used for assigning tasks and payment. Thus, it is disadvantageous for participants to intentionally wait for longer period of time.

In many situations, the number of tasks n is significantly large, and it takes long time to even finish assigning the first copy of a task. Therefore, once a participant finishes its assigned tasks, it will be assigned new tasks (still the first copy), rather than the second copy of the old tasks. This way, the delay factor d can be so large that the dishonest participants lose the incentives to cheat.

6 Conclusion and Future Work

We have shown how to use pipelining to allow participants in a grid computation to check each other's work, even in the presence of arbitrary collusion among lazy cheaters (provided the number of cheaters

[2]Before the second copy of U is assigned, cheating on U is risky, because the second copy might not be assigned to this group.

is not too high). We have also shown how to use pipelining to improve redundancy-based uncheatable grid computing schemes.

There are many directions for future work. For example, it would be interesting to design learning models and probabilistic weights to score participants on their likelihood of cheating and then tailor uncheatable grid computing schemes to these scores. Such an approach may help us improve collusion resistance further. Likewise, this paper is just a first step at showing how to exploit delays in grid computations to improve security and efficiency. We would anticipate that there are many new ideas that could be combined with this paradigm to achieve additional results in uncheatable grid computing specifically and in other computer security applications more broadly.

References

[1] SETI@Home: The Search for Extraterrestrial Intelligence Project. University of California, Berkeley. Available: http://setiathome.berkeley.edu/.

[2] R. Beigel, W. Hurwood, and N. Kahale. Fault diagnosis in a flash. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 571–580, 1995.

[3] R. Beigel, G. Margulis, and D. A. Spielman. Fault diagnosis in a small constant number of parallel testing rounds. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 21–29, 1993.

[4] C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. *Lecture Notes in Computer Science*, 1592:402–414, 1999.

[5] W. Du, J. Jia, M. Mangal, and M. Murugesan. Uncheatable grid computing. In *The 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 4–11, Tokyo, Japan, March 23–26 2004.

[6] P. Golle and I. Mironov. Uncheatable distributed computations. *Lecture Notes in Computer Science*, 2020:425–440, 2001.

[7] P. Golle and S. Stubblebine. Secure distributed computing in a commercial environment. In P. Syverson, editor, *Proceedings of Financial Crypto 2001*, volume 2339 of *Lecture Notes in Computer Science*, pages 289–304. Springer-Verlag, 2001.

[8] F. Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. *Mobile Agents and Security, Lecture Notes in Computer Science,Springer-Verlag*, 1419:92–113, 1998.

[9] L. Kahney. Cheaters bow to peer pressure. *Wired Magazine*, Feb. 15, 2001.

[10] F. Monrose, P. Wykoff, and A. D. Rubin. Distributed execution with remote audit. In *Proceedings of ISOC Symposium on Network and Distributed System Security*, pages 103–113, February 1999.

[11] T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts, springer-verlag. *Lecture Notes in Computer Science*, 1419:44–60, 1998.

[12] D. Szajda, B. Lawson, and J. Owen. Hardening functions for large scale distributed computations. *IEEE Symposium on Security and Privacy*, 2003.

[13] G. Vigna. Protecting mobile agents through tracing. In *Proceedings of the 3rd Workshop on Mobile Object Systems*, June 1997.

[14] G. Vigna, editor. volume 1419 of *Lecture Notes in Computer Science*. Springer, 1998.

[15] R. O. W. Aiello, S. Bhatt and S. Rajagopalan. Fast verification of any remote procedure call: short witness-indistinguishable one-round proofs for np. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 463–474, July 2000.

[16] F. S. Y. Minsky, R. van Renesse and S. D. Stoller. Cryptographic support for fault-tolerant distributed computing. In *Proceedings of Seventh ACM SIGOPS European Workshop,System Support for Worldwide Applications*, pages 109–114, Connemara, Ireland, September 1996.

[17] B. S. Yee. A sanctuary for mobile agents. In *Secure Internet Programming*, pages 261–273, 1999.