An Alternate Multiplicity-2 Task Assignment Scheme for Distributed Computations

Doug Szajda Jason Owen Barry Lawson Arthur Charlesworth Ed Kenney
University of Richmond
Richmond, Virginia
{dszajda, wowen, blawson, acharles, ekenney2}@richmond.edu

Abstract

Many recent large-scale distributed computing applications utilize spare processor cycles of personal computers. The resulting distributed computing platforms provide computational power that previously was available only through the use of expensive supercomputers. However, distributed computations running in untrusted or unstable environments raise a number of concerns, including the potential for disrupting computations and many security issues. It is shown that the standard techniques for managing these issues, i.e. replication and/or redundancy, still do not always resolve situations where computational integrity is threatened. This paper presents a generalized strategy for applying redundancy in a manner that is tunable and provides several advantages. In addition, the improvement is achieved without an increase in the amount of computation required by participants and only a slight increase in task tracking overhead.

Keywords: distributed computation, probabilistic verification, collusion, security

1. Introduction

The advent of large-scale distributed computing platforms, consisting of many personal computers connected to the Internet, provides researchers and practitioners a new and relatively untapped source of computing power. By utilizing the spare processing cycles of these computers, the computations are inexpensive and the harnessed power can rival that of a supercomputer (when many microprocessors are involved). In a volunteer distributed setting, the computation is easily divisible into independent *tasks*, each of which can be processed by a typical personal computer in a few hours. A *participant* downloads code from the *supervisor* of the computation in order to establish an execution environment in which the supervisor can execute tasks. Each task is assigned and dispatched to a participant, and upon completion of the task significant results are returned to the supervisor. Due to the fact that computations are executed outside of the control of the supervisor, participants can intentionally or unintentionally corrupt results, or possibly attempt to claim credit for work not completed.

The common technique for securing these computations is to utilize a redundant task assignment strategy or "simple redundancy" – i.e., assigning each task to two participants and therefore at least doubling the required cost of the computation. If the two returned tasks *do not* match, this is a signal of a potential problem to the supervisor and the task can be checked manually. If the two returned tasks *do* match, it is usually assumed that the task was computed correctly. However, this is a significant weakness; in many distributed computing platforms, there are no mechanisms in place to prevent someone from obtaining multiple (even hundreds of) user names and downloading hundreds or even thousands of tasks[1]. Thus, many participants could actually be controlled by a single individual and computing tasks in a single administrative domain or even on the same machine – thus allowing for an environment where computational integrity is endangered (due to pervasive software bugs or viruses). In addition, a malicious participant could cheat if she controlled matching tasks or she could *collude* with one or more participants under her control. To this end, we define any participant who compromises the integrity of the computation, whether intentionally or unintentionally, as an *adversary.*

Our scheme is motivated by the notion that a supervisor who accepts the increased cost of computation associated with simple redundancy should receive a better return on this investment. Specifically, these same computational resources can be allocated such that an adversary is *much* more likely to be detected, and such that detection can potentially reveal the identities of

[1] For example, the Search for Extra-Terrestrial Intelligence project [4] conducted by SETI@Home has experienced days in which more than 5000 new user names were assigned, and boasts participants who have *averaged* more than 1000 tasks completed each day.

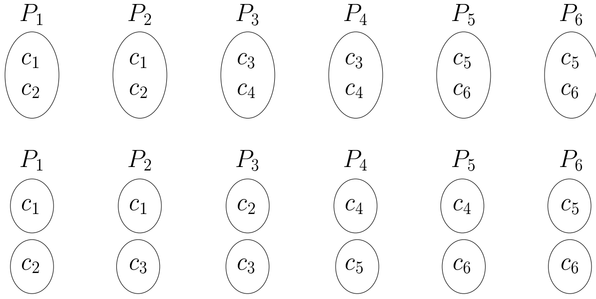

Figure 1. Alternative ways of assigning three tasks to six participants. Tasks in the top row are assigned using simple redundancy.

their colluding cohorts. In computer security, the detection of colluding adversaries is often difficult and expensive. Our mechanism requires no additional computation on the part of the participants, and only reasonable increases in resource management and bookkeeping costs for the supervisor.

As a simple example, consider a traveling salesperson computation involving only four cities, and assume that a participant can only compute the cost of two circuits. Let $c_1, c_2, \ldots, c_6$ denote the six non-equivalent circuits[2]. Without redundancy, this would require three participants (each computing the cost of two circuits), so simple redundancy requires six, which we denote by $P_1, P_2, \ldots P_6$. Figure 1 shows two possible ways of assigning two circuits to each participant. In the first assignment (simple redundancy), each subset is assigned to two participants, so for example P_1 and P_2 act as checks on each other's work. If they are both controlled by a single adversary, then the costs returned for circuits c_1 and c_2 are compromised since they can return identical incorrect results that the supervisor will assume is correct. Now consider the second, alternative assignment in Figure 1. Here, the work of P_1 is checked in part by P_2 and in part by P_3. Thus, in theory the supervisor can determine whether P_3 is colluding with P_1 and P_2 by checking whether the cost of c_3 has been correctly computed. Of course, an intelligent adversary familiar with this strategy would not return invalid results *every* time they received matching tasks or subtasks. Regardless, simple redundancy precludes even the possibility of efficiently identifying additional conspirators. The modified strategy provides this additional information without increasing the computational burden on any of the tasks.

Furthermore, if the supervisor intends to verify one full task worth of work in order to detect malicious activity, the two scenarios pose interesting arrangements. Under simple redundancy, the supervisor can only check either of $\{c_1, c_2\}$, $\{c_3, c_4\}$, or $\{c_5, c_6\}$ (due to the implied task granularity) to yield the possibility of detecting at most one colluding pair. Under the alternative scheme, the supervisor is free to check any two circuits from among the six in the computation. This leads to the possibility of identifying at most *two* colluding pairs. Thus, the supervisor gains twice the level of detection with the same amount of work.

The strategies presented in this paper represent a spectrum of possible task assignments. At one extreme is simple redundancy (the least expensive strategy), which provides the least protection from colluding adversaries, the smallest probability of detecting colluding adversaries, and no information about the identities of additional conspirators. At the other extreme is an assignment strategy we call *vertical partitioning*, which is of theoretical interest but does not scale to the dimensions of the typical distributed computation. Vertical partitioning is the most expensive of our strategies, but provides the greatest protection from colluding adversaries, the highest probability of detecting adversaries, and when detection occurs, the potential to efficiently identify *all* of the colluding participants when security is an issue. This is achieved through a distribution scheme in which for each pair of participants P and Q, Q checks the work of P; moreover, each part of the work of P is checked by the collective work of the participants different from P. The primary cost of vertical partitioning is task tracking overhead in the form of increased memory and increased database query times. However, the amount of computation required by the participants is unchanged. Between these two extremes lies a wide range of assignments we call *clustering*. By varying parameter values, the supervisor can choose the level of partitioning that is both suited to their specific application and provides the desired level of protection.

We note here that our strategy is not applicable to every distributed computation. In particular, we require that tasks can be divided into subtasks, a property absent from some distributed computations. Sequential computations, for which the basic model is the repeated iteration of a function on a small number of inputs, cannot be subtasked if each task is assigned only a single seed value. Protein folding (e.g. the Folding@Home project [1]) and some Mersenne Prime searches (e.g. GIMPS [2]), for example, fall under this restriction. Examples of applications for which our strategy is appropriate include DNA and protein sequence comparisons, exhaustive regression, and graphics rendering.

The remainder of the paper is organized as follows. In Section 2 we present our model of the distributed computations and platforms under consideration and introduce terminology. Section 3 presents the vertical partitioning scheme. Though impractical in its pure form, vertical partitioning provides the basis for the

[2]That is, if the four cities are A, B, C, and D, then we don't want to compute each of $ABCDA$, $BCDAB$, $CDABC$, and $DABCD$ when it suffices to compute only one of them.

Tasks			
1	2	3	4
A0	B0	C0	D0
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4
A5	B5	C5	D5
A6	B6	C6	D6

(a) $N = 4$ tasks each divided into $2N - 1 = 7$ subtasks

Participant Assignments							
1	2	3	4	5	6	7	8
A0	A0	A1	A2	A3	A4	A5	A6
A1	B0	B0	B1	B2	B3	B4	B5
A2	B1	B6	B6	C0	C1	C2	C3
A3	B2	C0	C4	C4	C5	C6	D0
A4	B3	C1	C5	D1	D1	D2	D3
A5	B4	C2	C6	D2	D4	D4	D5
A6	B5	C3	D0	D3	D5	D6	D6

(b) Assignment of subtasks to the $2N = 8$ participants

Table 1. A division of four tasks and associated assignment of subtasks

clustering scheme presented in Section 4, which explains how the ideas of vertical partitioning can be applied in practice. Finally, present our conclusions in Section 5.

2. Terminology

To introduce the language used to describe the scheme presented in this paper, consider the example structure of assignments of a hypothetical computation from Table 1. As depicted in (a), the *computation* is first divided into $N = 4$ *tasks*. Each task is then divided into $2N - 1 = 7$ *subtasks*. Then $2N = 8$ combinations, each of $2N - 1 = 7$ subtasks, are created (one for each of the $2N = 8$ participants) such that each subtask appears in exactly two combinations (the details of creating the combinations are presented in Section 3). An *assignment*, i.e., one combination of subtasks, is then presented to each participant to compute. Moreover, each term defined above is represented in Table 1 as follows:

- the *computation* corresponds to the entire table in (a);
- each *task* corresponds to a column in (a);
- each *subtask* corresponds to a single element (e.g., B_2) from a column in (a);
- each *assignment* to a participant corresponds to a column in (b).

Note that for simple redundancy a task consists of only one subtask; in this context, a task, subtask, and assignment are equivalent.

The computing platform consists of a *supervisor* — a trusted central control server or server hierarchy coordinating many (typically 10^4 to 10^7) personal computers in a "master-slave" relationship. The slave nodes, or *participants* are given work assignments by the supervisor. Because tasks in a computation are independent, communication is necessary (and allowed) only between individual participants and the supervisor. In some cases participants receive remuneration, in one of a variety of forms, for completing their associated work assignment.

3. Vertical Partitioning

Here, we assume that there are N tasks that are to be assigned to $2N$ participants such that

- Each subtask is assigned to exactly two participants.
- Each pair of participants share exactly one subtask.

Since there are $2N$ participants, each can be paired with $2N-1$ other participants, so tasks must be divided into $2N - 1$ subtasks. Moreover since the number of subtasks in the computation, $N(2N - 1)$, is the same as the number $\binom{2N}{2}$ of pairs of participants, such an assignment is always possible. An example assignment of 4 tasks to 8 participants is shown in Table 1.

There are two immediate consequences of assigning tasks in this way. First, subtasking shrinks the checkable unit of execution, which both reduces the burden of checking individual returned results, and allows the verification process to cover more of the computation. The result is that the supervisor is given finer control over which results are verified. The second is that it spreads the responsibility for verifying the work of a single participant from one other participant to *all* other participants. This distribution creates the potential for efficient identification of all colluding parties once a single colluding pair have been identified.

The beneficial effects of shrinking the size of checkable execution units should not be discounted. For many applications, the only way for the supervisor to verify a returned result is to recompute the entire task. This is expensive, and obviously cannot be done for any significant proportion of the tasks. Subtasking, however, allows the supervisor to effectively check the work of N participants for the cost of verifying a single task. The improvement in the probability of detecting malicious activity is the results of this quantization effect.

In essence, the supervisor who uses simple redundancy is locked into performing checks at the task granularity, which limits the efficacy of the checking effort.

There are other costs associated with vertical partitioning, primarily stemming from the management of subtasks. Subtasking introduces at least a factor $2N-1$ increase in the cost of maintaining any task assignment database, since tracking a subtask is every bit as expensive as tracking a full task. For large N values this will likely become prohibitive. Handling the factor $2N-1$ increase in the number of returned results will also pose difficulties, and is a problem that can not always be easily handled by adjusting the criterion by which results are deemed significant. Tuning applications so that tasks return the appropriate number of significant results is often difficult, and involves more than simply narrowing the filter, since there is always the danger of creating a filter so small that important results will be missed. Subtasking only exacerbates this tuning problem.

3.1 Strategy Analysis

Here, we present several quantities related to the potential for comprised integrity in a distributed computation. The derivation of these quantities are rather involved but are explicitly developed in [5]. Therein, it is shown that if an adversary controls a proportion p of the $2N$ participants in the computation, then the expected number of tasks and subtasks controlled under either strategy is the same, and is given by

$$\begin{aligned} \text{E}(\#\text{ of subtasks}) &= pN(2pN-1) \\ \text{E}(\#\text{ of tasks}) &= \frac{pN(2pN-1)}{2N-1}. \end{aligned}$$

However, it is also shown in [5] that vertical partitioning provides a benefit in terms of stability, since the variance of the number of subtasks under control of an adversary is zero. This is not the case with simple redundancy (unless the adversary controls either *all* of the participants or *none* of them), where the variance of the number of subtasks is given by

$$\text{Var}(\#\text{ subtasks}) = \frac{2pN^2(2pN-1)(1-p)(2N(1-p)-1)}{2N-3}.$$

Note that this function is symmetric about the line $p = 1/2$. So, although the means are equal for the two methods, the variances are not.

In addition, it is shown in [5] that under simple redundancy the probability that an adversary is detected by a supervisor checking m tasks is given by

$$P(\text{detecting adversary}) = 1 - \frac{(N-m)!}{\binom{2N}{2pN}} \times \sum_{k=\tau_{p,N}}^{pN} \frac{(N-k)!}{(N-k-m)!} \frac{1}{(2pN-2k)!} \frac{1}{k!} \frac{2^{2pN-2k}}{(N+k-2pN)!},$$

where $\tau_{p,N}$ is defined by $\tau_{p,N} = \max\{0, 2pN - N\}$. The vertical partitioning strategy with a supervisor checking k subtasks has a detection probability given by

$$1 - \frac{\binom{N(2N-1)-pN(2pN-1)}{k}}{\binom{N(2N-1)}{k}}.$$

To be clear on terminology, we sometimes refer to both the number of *tasks* compromised or checked in vertical partitioning and the number of *subtasks* compromised or checked in vertical partitioning. Technically the term "task" does not apply to vertical partitioning, nor does the term "subtask" apply to simple redundancy, since an adversary cannot compromise work at a subtask granularity in the latter.

3.2 Adversary Detection

Here, the probabilities of detecting malicious activity are considered assuming that the adversary will return an invalid result if and only if they have either both copies of a task (in simple redundancy) or both copies of a subtask (in vertical partitioning). [Certainly an intelligent, malicious adversary with knowledge of our strategy will not attempt to return invalid results at every opportunity presented them, but will instead likely attempt to game the system to their advantage.] Our assumption allows a tractable analysis that is valuable as a means of comparing the performance of vertical partitioning with that of simple redundancy.

We begin by considering vertical partitioning. Let the exact number of subtask pairs assigned to the adversary is $L(2L-1)$, where $L = pN$, and there are a total of $N(2N-1)$ distinct subtasks. Thus, if we verify a single randomly chosen subtask, the probability that we detect the adversary is $(L(2L-1))/(N(2N-1))$. More generally, if we instead verify k of the subtasks, then the probability of detecting the adversary is given by

$$1 - \frac{\binom{N(2N-1)-L(2L-1)}{k}}{\binom{N(2N-1)}{k}} \geq 1 - \left(1 - \frac{L(2L-1)}{N(2N-1)}\right)^k,$$

where the quantity to the right of the inequality is a lower bound for the quantity using a binomial probability calculation. For simple redundancy, and specifically the situation in which the supervisor attempts to detect cheating by verifying (i.e. computing) a single task, the probability of detecting the adversary under simple redundancy is much easier. If an adversary controls exactly k tasks, then the probability of the supervisor randomly choosing a task controlled by the adversary (from among N tasks) is $\frac{k}{N}$. Since the expected number of tasks controlled by the adversary is $L(2L-1)/(2N-1)$, the probability of catching the adversary is

$$\frac{L(2L-1)}{2N-1}\frac{1}{N} = \frac{p(2pN-1)}{2N-1}.$$

Note that this is the same as the probability derived for vertical partitioning with the supervisor checking only a single *subtask*. Thus vertical partitioning achieves in this case an equal level of protection with a factor $2N-1$ reduction in computation cost.

To generalize our simple redundancy analysis by considering the probability of detecting a cheater when the supervisor verifies m full tasks rather than just one, the probability in this case is given by

$$1 - \frac{(N-m)!}{\binom{2N}{2L}} \times \sum_{k=\tau_{L,N}}^{L} \frac{(N-k)!}{(N-k-m)!} \frac{1}{(2L-2k)!} \frac{1}{k!} \frac{2^{2L-2k}}{(N+k-2L)!}.$$

4 Clusters: Practical Vertical Partitioning

Real distributed computing can consist of millions of tasks distributed to millions of participants. Vertical partitioning is not practical at these orders of magnitude. Using this strategy in practice thus requires breaking the computation into several *clusters*, each of which consists of a reasonable sized number of work units for a given application, with each cluster employing a vertical partitioning strategy. To keep our notation as consistent with the previous sections as possible, we will assume for this section that the entire computation consists of M work units. These are to be distributed to $2M$ participants, and the M work units are to be divided into C clusters each containing N work units. Thus, $C = M/N$. The N work units in each cluster are to be distributed to $2N$ participants according to the vertical partitioning strategy.

Clustering has several advantages over pure vertical partitioning. First, unlike vertical partitioning, an adversary controlling multiple participants is no longer guaranteed to possess duplicates of any particular subtask, since tasks in different clusters are disjoint. In addition, the relatively low N values as compared to vertical partitioning lead to decreased task tracking overhead, and makes the strategy practical for real computations. Most importantly, the notion of clustering provides the supervisor of a computation with significant flexibility — by varying the parameters N (or equivalently C), the entire spectrum from simple redundancy (clustering with $N = 1$ and $C = M$) to vertical partitioning ($N = M$ and $C = 1$) can be covered.

4.1 Analysis of Clustering

We examine here how the introduction of clustering affects several of the probabilistic quantities previously considered. The derivations for the exact expressions of these probabilities are again given in [5]. To consider the expected number of tasks under the control of the adversary if the computation uses a clustering scheme (once again assuming that the adversary controls proportion p of the $2M$ participants in the computation, for a total of $2pM$ participants), let $\{C_1, C_2, \ldots, C_C\}$ denote the clusters. Vectors are used to describe specific assignments of participants to the adversary, with $\mathbf{v} = (k_1, k_2, \ldots, k_C)$ denoting the event that the adversary has been assigned exactly k_i participants in cluster C_i. Such an assignment must of course always satisfy $\sum_{i=1}^{C} k_i = 2pM$, where $2pM$ must be an integer (so p is constrained). From [5], the expected number of subtasks under control by an adversary controlling $2pM$ participants is given by

$$E(\#\text{ of tasks}) = \frac{1}{2N-1} \frac{1}{\binom{2M}{2pM}} \times \sum_{\mathbf{v}=(k_1,\ldots,k_C)\in\mathcal{E}} \prod_{i=1}^{C} \binom{2N}{k_i} \sum_{j=1}^{C} \binom{k_i}{2}.$$

There are several special (but non-trivial) cases of input parameters (e.g. $p = (2pM-1)/(2M-1)$) for which one can easily calculate this quantity. In each case, the value is identical to the values obtained for both pure vertical partitioning and simple redundancy. Moreover, we have computed several values for nontrivial parameter settings, and again in each case the expected number of tasks matches the values obtained for both simple redundancy and vertical partitioning. We thus conjecture, but have not yet been able to prove, that this expected value is equal to the expected values for simple and vertical partitioning regardless of the value of C.

From [5], the probability of detecting an adversary who controls a single subtask, given that the supervisor verifies m subtasks, is given by

$$\sum_{\mathbf{v}\in\mathcal{E}} P(\mathbf{v}) \left(1 - \frac{\binom{M(2N-1)-T(\mathbf{v})}{m}}{\binom{M(2N-1)}{m}}\right),$$

where

$$P(\mathbf{v}) = \frac{1}{\binom{2M}{2pM}} \prod_{i=1}^{C} \binom{2N}{k_i}.$$

Figure 2 illustrates how this probability (with a few sample C values) compares with those obtained for simple redundancy and vertical partitioning. In this graph, we assume that the adversary has been assigned exactly two participants and in a manner most favorable for disruption. That is, in the data for simple redundancy, we assume that the adversary has been assigned identical work units and in clustering we assume that the two assigned participants are in the same cluster. In each case we assume that one equivalent task is verified by the supervisor, so in pure vertical partitioning, $2M-1$ subtasks are verified, while in clustering $2N-1$

Simple Redundancy ($M = 4$, $N = 1$, $C = 4$)

Tasks			
1	2	3	4
A0	B0	C0	D0

Participant Assignments							
1	2	3	4	5	6	7	8
A0	A0	B0	B0	C0	C0	D0	D0

Clustering ($M = 4$, $N = 2$, $C = 2$)

Tasks			
1	2	3	4
A0	B0	C0	D0
A1	B1	C1	D1
A2	B2	C2	D2

Participant Assignments							
1	2	3	4	5	6	7	8
A0	A0	A1	A2	C0	C0	C1	C2
A1	B0	B0	B1	C1	D0	D0	D1
A2	B1	B2	B2	C2	D1	D2	D2

Vertical Partitioning ($M = 4$, $N = 4$, $C = 1$)

Tasks			
1	2	3	4
A0	B0	C0	D0
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4
A5	B5	C5	D5
A6	B6	C6	D6

Participant Assignments							
1	2	3	4	5	6	7	8
A0	A0	A1	A2	A3	A4	A5	A6
A1	B0	B0	B1	B2	B3	B4	B5
A2	B1	B6	B6	C0	C1	C2	C3
A3	B2	C0	C4	C4	C5	C6	D0
A4	B3	C1	C5	D1	D1	D2	D3
A5	B4	C2	C6	D2	D4	D4	D5
A6	B5	C3	D0	D3	D5	D6	D6

Table 2. Task division and assignment for simple redundancy, clustering, and vertical partitioning. Assignments shown for $M = 4$ participants. The clustering example uses $C = 2$.

Quantity	Simple Red. (S)	Clusters (C)	Vertical Partitioning (V)	Summary
# rows in matrix	1	$2N - 1$	$2M - 1$	$S \leq C \leq V$
P(adversary cheats)	$\frac{1}{2M-1}$	$\frac{2N-1}{2M-1}$	1	$S \leq C \leq V$
# tasks compromised	1	$\frac{1}{2N-1}$	$\frac{1}{2M-1}$	$S \geq C \geq V$
Expected # tasks compromised	$\frac{pM(2pM-1)}{2M-1}$	$\frac{pM(2pM-1)}{2M-1}$?	$\frac{pM(2pM-1)}{2M-1}$	$S \stackrel{?}{=} C \stackrel{?}{=} V$
P(adv. detected, 1 task checked)	$\frac{p(2pM-1)}{2M-1}$		$1 - \frac{\binom{M(2M-1)-pM(2pM-1)}{2M-1}}{\binom{M(2M-1)}{2M-1}}$	$S \leq C \leq V$

Table 3. Comparing simple redundancy, clustering, and pure vertical partitioning, with M tasks in the computation, and N tasks in each cluster. The detection probability for clusters is omitted due to space considerations.

subtasks are verified. As expected, the probability for clustering falls between that for vertical partitioning and that for clustering, with the curve approaching simple redundancy as C increases, and pure vertical partitioning as C decreases. Some of our findings are summarized in Table 3.

An important question for a supervisor considering the use of our strategy is the optimum cluster size. The optimum cluster size will also depend in part on application characteristics, including factors such as the total number of tasks appropriate for a computation, the structure of tasks, the inherent verifiability of tasks, and especially resilience to incorrect results. When an adversary receives multiple copies of the same task in a computation where results are checked via simple redundancy, an entire task is compromised, and this compromise will likely evade detection. The relatively large variance associated with simple redundancy means there is a nontrivial probability that an adversary could be assigned several pairs of matching tasks. At the same time, with clustering an adversary controlling multiple participants is guaranteed to have matching subtasks, but the damage they can inflict at one time is limited to the size of a subtasks. Applications such as graphics rendering, Monte Carlo simulations, and genetic algorithms that can tolerate small amounts

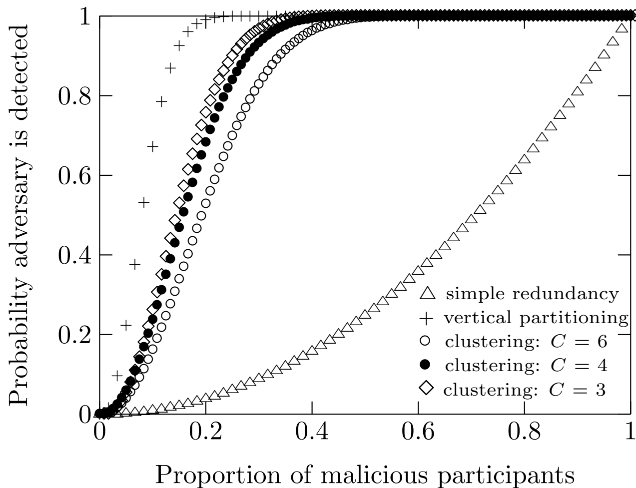

Figure 2. Comparing detection probabilities under the assumption that the adversary returns a bad result for any task or subtask she controls, for $M = 60$ tasks and various cluster sizes.

of incorrect results but may be badly damaged by the loss of an entire task (or more) will benefit from the stability provided by the clustering strategy.

4.2 Augmenting our strategy with other schemes

Our clustering strategy is extensible in the sense that it can be used in conjunction with some recently proposed security measures. The ringer schemes of Golle and Mironov [3], and Szajda, et al. [6] can both improve the probability of detecting malicious activity and at the same time decrease some of the costs associated with our strategy. The ringer strategy involves seeding tasks with precomputed results in such a manner that the identities of seeds are difficult for the adversary to determine and that the values are guaranteed to be returned as significant by an honest participant. Like the present scheme, the ringer strategy provides the supervisor with improved verification capabilities. Even when a participant returns all ringers in a task, however, there is no guarantee that the remainder of the task has been executed correctly, or even that it has been executed at all—a malicious participant may be able to identify the ringers, or simply get lucky. When used in conjunction with clustering, a ringer strategy further increases the number of tasks that can be verified which corresponds to a higher probability of detecting an adversary, malicious or not.

5. Conclusions

We have presented a novel means of applying redundancy to distributed computations. This technique modifies redundancy by assigning work to participants in a manner that increases the number of participants that check the work of a single participant. We presented analysis that showed that the expected number of tasks compromised is the same as for simple redundancy, but that the effect of our assignment scheme is an increase in the ability to detect colluding adversaries, as well as the potential to identify all colluding co-conspirators once one colluding pair has been identified. These enhancements are achieved without increasing the computational burden on the participants, and with acceptable increases in task tracking overhead imposed on the supervisor. Moreover our strategy provides greater stability in the potential for collusion for an adversary with a given proportion p of participants. We further showed that our general strategy is tunable, and that by varying input parameters, the supervisor of a computation can choose a level of protection that covers a spectrum from simple redundancy at one extreme to vertical partitioning at the other.

References

[1] The Folding@home Project. Stanford University. http://www.stanford.edu/group/pandegroup/cosm/.

[2] The Great Internet Mersenne Prime Search. http://www.mersenne.org/prime.htm.

[3] P. Golle and I. Mironov. Uncheatable distributed computations. In *Proceedings of the RSA Conference 2001, Cryptographers' Track*, pages 425–441, San Francisco, CA, 2001. Springer.

[4] The Search for Extraterrestrial Intelligence project. University of California, Berkeley. http://setiathome.berkeley.edu/.

[5] D. Szajda, A. Charlesworth, B. Lawson, J. Owen, and E. Kenney. Collusion resistant redundancy for distributed metacomputations. Technical Report TR-05-02, Richmond, VA, 2005.

[6] D. Szajda, B. Lawson, and J. Owen. Hardening functions for large-scale distributed computations. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 216–224, Berkeley, CA, May 2003.