Defeating Colluding Nodes in Desktop Grid Computing Platforms

Gheorghe Cosmin Silaghi
Dept. of Business Information Systems
Babeș-Bolyai University, Cluj, Romania
gsilaghi@econ.ubbcluj.ro

Filipe Araujo, Luis Moura Silva
CISUC, Dept. of Informatics Engineering
University of Coimbra, Portugal
{filipius, luis}@dei.uc.pt

Patricio Domingues
School of Technology and Management
Polytechnic Institute of Leiria, Portugal
patricio@estg.ipleiria.pt

Alvaro E. Arenas
Rutherford Appleton Laboratory
STFC, Didcot, UK
a.e.arenas@rl.ac.uk

Abstract

Desktop Grid systems reached a preeminent place among the most powerful computing platforms in the planet. Unfortunately, they are extremely vulnerable to mischief, because volunteers can output bad results, for reasons ranging from faulty hardware (like over-clocked CPUs) to intentional sabotage. To mitigate this problem, Desktop Grid projects replicate work units and apply majority voting, typically on 2 or 3 results.

In this paper, we observe that this form of replication is powerless against malicious volunteers that have the intention and the (simple) means to ruin the project using some form of collusion. We argue that each work unit needs at least *3 voters and that voting pools with conflicts enable the master to spot colluding malicious nodes. Hence, we post-process the voting pools in two steps: i) we use a statistical approach to identify nodes that were not colluding, but submitted bad results; ii) we use a rather simple principle to go after malicious nodes which acted together: they might have won conflicting voting pools against nodes that were not identified in step i. We use simulation to show that our heuristic can be quite effective against colluding nodes, in scenarios where honest nodes form a majority.*

1. Introduction

Internet Desktop Grids[1] aggregate huge distributed resources over the Internet and make them available for running a growing number of applications. A major concern in such a middleware is the support for sabotage tolerance (ST). Since computations run in an open and non-trustable environment, it is necessary to protect the integrity of data and validate the computation results. Without a sabotage-detection mechanism a malicious user can potentially undermine a full computation that may have been executing for long [2].

All important ST techniques designed up-to-date for Internet DGs are based on the strong assumption that workers are independent from each other. While this assumption is fulfilled, actual sabotage tolerance techniques perform very well, supplying the required (very low) error rate for the overall computation. But, as Zhao *et al.* [11] acknowledge, a potential threat comes up when workers can devise some scheme to interact. In fact, actual developments in desktop grid middleware are enabling connection of workers in P2P networks. For example, collaborative techniques are very attractive for data distribution [8], especially when the the DG runs a parameter sweep application. Kim *et al.* [5] proposed an entirely distributed P2P desktop grid. These solutions violate the workers independence assumption, as they enable workers to communicate, thus empowering sabotage. This brings new challenges to desktop grid systems, because the master is not prepared to fight potential collective malicious behaviors, resulting from orchestrated workers.

To face the new collusion threat, this paper proposes a novel approach as a complement to the actual replication-based mechanism. With replication, the master decides about the trustworthiness of a result immediately after having collected all replicas of a work unit. Instead, in our approach, the master will postpone the decision moment in the replicated voting pools until it gathers enough information to infer the trustworthiness of the workers. We present in this paper a statistical approach to analyze together the voting pools and to infer and classify a worker as being malicious or not. Further, the master can mark a voting pool as being suspicious if a honest worker is losing the decision. On these voting pools, the master can apply further

replication to conclude about the validity of the result.

In contrast to other works on ST in DGs [7, 9, 11], we evaluate our approach considering a wider range of malicious saboteurs, including naive and colluding ones, as well as transient saboteurs which change their profile during their life.

The paper is further organized as follows. In Section 2, we present background information about desktop grids. In Section 3, we present our collusion-resistant sabotage tolerance technique. In Section 4 we present and discuss the results obtained with our sabotage tolerance protocol. Section 5 concludes the paper.

2 Background

A desktop grid system consists of a server (referred further as the *master*) which distributes work units of an application to *workers*. Workers are machines which voluntarily join the computation over the Internet. Once a work unit is completed at the worker site, the result is returned back to the master. A result error is any result returned by a worker that is not the correct value or within the correct range of values [6]. The *error rate* ϵ is defined as the *ratio* of bad results or errors among the final results accepted at the end of the computation. Thus, for a batch of N work units with error rate ϵ, the master expects to receive ϵN errors. For every application, the master employs some sabotage-tolerance mechanism for obtaining an acceptable error rate ϵ_{acc} with regard to its application. *Redundancy* is defined as the ratio of the total number of replicas assigned to workers to the actual number N of work units. Usually, redundancy is larger than 1, which means that we need computing resources only for verification purposes.

2.1 Related work

BOINC[1], the most popular Internet desktop grid middleware uses *replication* with majority voting [1, 7] as the ST mechanism. The master distributes $2m - 1$ replicas of a work unit to workers and when it collects m similar results, it accepts that result as correct. With the same model, Wong [9] presents a variation of the replication with only 2 replicas, by considering the workers arranged as nodes in a graph and connected by the workunits. This protocol allows the host to estimate without auditing the proportion of untrusted workers and how often these workers would submit incorrect results. From the redundancy point of view, a more efficient method for error detection is *spot-checking* [11], where a work unit with a known result is distributed at random to workers. Worker results are compared against the previously computed and verified result. If the result for the spotter is erroneous, then, the worker is blacklisted, in the sense that all its previously and future results are discarded. Credibility-based systems [7] use conditional probabilities of errors based on the history of host result correctness. It assumes that hosts that have computed many results with very few errors are more reliable than hosts with a history of erroneous results. This method has problems to fight against hosts that behave well for a long period of time, in order to gain credibility, and after that start to sabotage.

Yurkewych *et al.* [10] presents a study regarding the colluding behavior in commercial desktop grids. As workers receive money for their results, this study employs a game-theoretical analysis, based on the traditional tax-auditing game. They show that redundancy can eliminate the need for result auditing when collusion is prevented. Non-redundant allocation can work even with colluding scenarios, if the master is able to impose high penalties on cheating workers, given that some pre-defined positive audit rate is preserved. We differentiate from this study as in volunteer computing no monetary means can be enforced to penalize malicious workers. Kondo *et al.* [6] performed the first study that characterizes errors in Internet Desktop Grids. They approached only I/O errors and discussed the efficiency of the above sabotage tolerance methods. They concluded that simply blacklisting erroneous hosts can cost as much as 40% of the throughput, coming from hosts that produce good results in general. They also concluded that replication with majority voting is the most reliable sabotage tolerance method in order to achieve low host error rates. Therefore, we will use replication with majority voting as a starting point of our approach.

2.2 Sabotage models

To characterize erroneous hosts, we consider two models that define extreme behaviors: the first behavior is the *naive malicious*, where a node randomly commits mistakes in some work units independently of the behavior of other nodes. Note that this could possibly happen because the node is faulty, due, for instance, to malfunctioning hardware. In the other extreme, we consider *colluding nodes* that make their behavior depend from the participation of other malicious nodes in voting pools. They introduce errors only when they are sure that their sabotage can be successful, for instance, when they know that other malicious nodes are participating in the voting pool, thus forming a majority. While naive malicious nodes expose themselves to be detected and possibly black-listed in a rather easy way, the colluding voters are much more subtle and can easily pass undetected. We denote basic naive malicious nodes by M_1*-type*. An M_1-type worker submits bad results with a constant probability s, called *sabotage rate*. If we as-

[1]Where not specifically stated, the methods herein reviewed assume independence between workers.

sume the existence of a fraction f of M_1-type saboteurs in the total population of workers, then the expected error rate $\epsilon_{M_1}(f, s, m)$ of the majority-voting replication is [7]:

$$\sum_{j=m}^{2m-1} \binom{2m-1}{j} (fs)^j (1-fs)^{2m-1-j} \quad (1)$$

Unlike the basic M_1-type, a colluding saboteur (further referred as an M_2-*type* worker), has the will and the means to reach other saboteurs in order to develop malicious coalitions. In model M_2, a dishonest worker w will sabotage only if it finds enough dishonest peers to join it to defeat the honest nodes involved in the same voting pool. We assume that malicious nodes are connected in a complete graph, such that communication between any two of them is always possible. However, at this stage of our work, we impose a limit to the power of malicious nodes: they are not aware of our sabotage detection mechanisms. If the fraction of M_2-type saboteurs in the total population of workers is f and each saboteur is active with probability s (i.e. s is the probability it will launch the collusion-formation protocol), then the expected error rate $\epsilon_{M_2}(f, s, m)$ is given by Equation (2).

$$\sum_{j=m}^{2m-1} \binom{2m-1}{j} \left(1-(1-s)^j\right) f^j (1-f)^{2m-1-j} \quad (2)$$

We consider yet another type of saboteurs deemed M_3-*type*, mixed malicious, which change their behavior during their life, behaving either naive or colluding, but always performing a dishonest role. For an M_3-type saboteur, c is the *naive ratio*, which is the fraction of work units for which the worker behaves as an M_1-type saboteur with sabotage rate s_1, while for the remaining $1-c$ fraction of work units it behaves like a M_2-type saboteur with sabotage rate s_2.

2.3 Discussion

Given that M_1-type saboteurs submit a rather small fraction s of bad results (with an average of 0.0034 for independent I/O errors [6]), it results that colluding saboteurs are much more destructive than independent ones. Figure 1 shows the comparison of the error rates achieved with different number of identical results required m, for $f = 0.035$ and $s = 0.0335$ in the case of both M_1-type and M_2-type saboteurs[2]. To allow for a better comparison, we used the same value of $s = 0.0335$ for all types of colluders in Figure 1. However, colluding saboteurs would be much more destructive if they always try to sabotage, i.e., if $s = 1$ (naturally, this can leave more traces of their intervention). The

[2]We assumed the same error rate parameters as for the top 10% erroneous hosts reported by Kondo *et al.* [6].

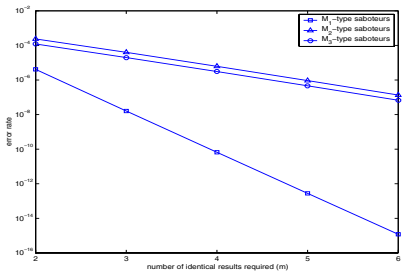


Figure 1: Error rates comparison between various types of malicious workers against simple replication

error rate of M_3-type saboteurs is something in between M_1 and M_2-types, being much closer to the latter. We considered $c = 0.5$ for an M_3-type saboteur, while keeping the same f and $s_1 = s_2 = s$.

We define the *effectiveness* of a saboteur as being the ratio between the number of times it succeeds to defeat the sabotage tolerance mechanism versus the total number of times it sabotages. While naive saboteurs succeed to defeat the master's replication-based sabotage tolerance mechanisms only in a small fraction of the attempts, a colluding saboteur will sabotage only when it is sure to win the majority voting, and therefore, its effectiveness is total (1). We should also note that besides being less destructive, naive saboteurs leave more traces behind them, making it much easier for the master to spot them out.

3 A collusion-resistant sabotage tolerance protocol

In this section we propose a collusion-resistant sabotage tolerance protocol, complementing the actual replication, which seems to be very effective against M_1-type naive saboteurs.

3.1 Overview

From results of Kondo *et al.* [6] and from the DG projects we are aware of, we fix $m = 2$, i.e., we use $2m - 1 = 3$ replicas. However, instead of deciding on a result as soon as the master gets a majority of 2 similar responses, it will postpone the decision until it gets all three results from that work unit and until it collects enough results of related workers from *different* work units. We further consider each work unit as a voting pool, where each worker is worth a vote. After it collects a number of voting information (the most it collects the better), the master will analyze the information acquired from the voting behavior and will infer which are the M_1-type naive saboteurs. The rationale for this is that, once these nodes are identified, the

f_1	proportion of M_1-type workers
s_1	sabotage rate of M_1-type workers
f_2	proportion of M_2-type workers
s_2	sabotage rate of M_2-type workers
f_3	proportion of M_3-type workers
c	naive ratio for M_3-type workers
$s_{3,1}$	sabotage rate of M_3-type workers while behaving as M_1-type
$s_{3,2}$	sabotage rate of M_3-type workers while behaving as M_2-type

Table 1: Parameters describing the population structure

remaining contradictory voting pools only contain colluding nodes of type M_2 and M_3. Then, the master will reconsider these work units and ask for further responses.

A voting pool that contains contradicting votes is of interest for the master, because it contains at least one faulty node. A valuable observation is that in the case of naive M_1-type workers, the total number of such conflicting voting pools is higher than in the case of M_2-type saboteurs, for the same f and s, regardless their value. This makes it easier for the master to spot out naive saboteurs. While a hybrid M_3 saboteur has a mixed behavior switching between being naive and colluding, this model will give us less clues than naive saboteurs, but more clues than colluding ones. Therefore, given our assumption that M_2 and M_3 nodes are not aware of our sabotage tolerance mechanism, we expect a better response against M_3 than against M_2 saboteurs.

3.2 Statistical modeling of the voting behavior

Consider a population $\mathcal{S}_P$ consisting of honest and malicious workers. Table 1 describes the meaning of each structure parameter. We impose that honest workers are in majority, i.e., $f_1 + f_2 + f_3 < 0.5$. To enable evaluation, we assume that the population structure is stable over time and that workers fully comply with their models during all their life. As we said before, we assume that nodes are unaware of the algorithm that the master uses to spot collusion. Let the master distribute replicated tasks from a set of work units $\mathcal{S}_W$, such that, on average every worker gets on average N tasks.

A voting pool $V = \{v_1, v_2, v_3\}$ is a set of three ($m = 2$) different workers $v_i \in \mathcal{S}_P$, where each worker submits a binary vote. Consider a fixed worker $v \in V$. The number of *votes against* the worker collects in the voting pool V can be modeled as a random variable $Y_v : \{0, 1, 2\} \to \mathbb{R}$, where $Y_v(i) = p_{v,i} \geq 0$ is the probability that the worker v has i votes against in the voting pool V, with $\sum_i p_{v,i} = 1$.

Due to the i) population structure stability; ii) workers compliance with their model; and iii) the fact that workers can not influence how the master distributes them in the voting pools, any two voting pools for the same worker are statistically identical and independent and thus, we can model the behavior of a worker during a sequence of N voting pools as a multinomial experiment with N trials Y_v.

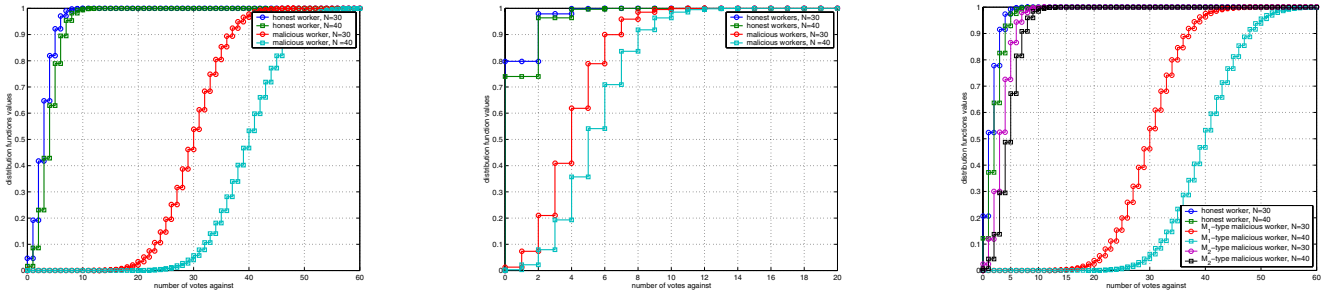
We denote by $Y_{v,N} : \{0, 1, \ldots, 2N\} \to \mathbb{R}$ the random variable defining the probabilities for the worker v to collect a given number of votes against over a total of N voting pools. From the independence between two different voting pools, we can infer that $Y_{v,N} = \prod_{t=1}^{N} Y_v = Y_v^N$. [3] In our case, as every Y_v is defined over the set $\{0, 1, 2\}$, for the sake of simplicity, the discrete values of the random variable $Y_{v,N}$ can be obtained by computing the corresponding coefficients of a polynomial like the one of Equation (3). These coefficients can be computed either by successively multiplying the polynomials (as we did) or by applying the multinomial theorem and using the trinomial coefficients [4].

$$(p_{v,0} + p_{v,1}X + p_{v,2}X^2)^N \tag{3}$$

The *joint distribution function* of a voter v with $Y_{v,N}$ is $F_v : \{0, 1, \ldots 2N\} \to \mathbb{R}$, defined as $F_v(i) = Prob(Y_{v,N} \leq i)$, $F_v(i)$ being the summation of all coefficients of the polynomial (3) up to the i rank.

For a given population structure, after determining the initial values $p_{v,0}$, $p_{v,1}$ and $p_{v,2}$ and computing the coefficients of Equation (3) using multiplications of polynomials we got distribution function curves like the ones depicted in Figure 2. The population we used to plot these curves was the following: in each of them we considered $f = 0.1$ malicious workers. Each worker has some predefined sabotage rate of 0.5 and we assigned once $N = 30$ and $N = 40$ work units per worker. First, in Figure 2(a) we considered only naive M_1-type workers. In Figure 2(b) we replaced naive M_1-type workers with colluding M_2-type workers. We can notice that for the same percentage of the malicious workers ($f = 0.1$) and the same sabotage rate ($s = 0.5$), the gap between the distribution functions for $N = 30$ and $N = 40$ increases, while the distribution function of naive workers shifts to the right. In Figure 2(c) we considered a mix of M_1 and M_2-type workers, keeping the proportion of malicious workers identical ($f_1 + f_2 = 0.1$). We can notice that the distribution function of the naive malicious is on the right side, the distribution function of the honest workers is on the left side, while the distribution function of the colluding malicious is shifted a bit on the right of the honest workers distribution.

[3] Given 2 random variables $Y_1 : \{x_i^1,\ i = \overline{1, n_1}\} \to \mathbb{R}_+,\ Y_1(x_i^1) = p_i^1$, $\sum_i p_i^1 = 1$ and $Y_2 : \{x_i^2,\ i = \overline{1, n_2}\} \to \mathbb{R}_+,\ Y_2(x_i^2) = p_i^2$, $\sum_i p_i^2 = 1$, the product $Y = Y_1 Y_2$ is defined by over the space $\{x_i^1 \wedge x_j^2,\ i = \overline{1, n_1},\ j = \overline{1, n_2}\}$ with the following expression: $Y(x_i^1 \wedge x_j^2) = p_1^i p_2^j$.



(a) Honest and M_1-type workers (b) Honest and M_2-type workers (c) Honest, M_1 and M_2-type workers

Figure 2: Theoretical distribution functions F_v for various population structures

After we analyzed extensively various population structures, using the mathematical procedure explained in the above paragraph, the following important conclusions can be drawn out: (i) M_1-type (naive) saboteurs always collects the biggest number of votes against, their joint distribution functions being the most-right ones in the graphic; (ii) for N large enough, there is a clear separation between the distribution functions F_v for the case of honest workers versus malicious workers; (iii) the honest workers have the distribution functions on the left side of the graphic, the distances between a honest worker distribution and a naive (M_1) malicious one being the bigger ones; (iv) as expected, the distribution function for an M_3-type worker, not shown on the plots due to space consideration, will lay down between distribution functions of M_1 and M_2 workers.

3.3 Spotting out naive saboteurs

Based on the theoretical conclusions drawn out in Section 3.2, we now propose a method for spotting out saboteurs that behave permanently or intermittently as naive M_1-type ones. This includes M_3-type workers. Suppose that the master distributes a batch of work units, such that each worker takes place in an average of N voting pools. For some particular worker i, the number of voting pools is N_i and the master can count the number of times c_0, c_1, c_2, the worker registered 0, 1, and 2 votes against, among its work units. These figures, divided by N_i give the practical (sampled) probabilities p_0, p_1, p_2 (as used in Equation 3) for that worker. Applying the procedure described in Section 3.2, the master will obtain one distribution function (similar to the ones of Figure 2) for each worker.

For two voters $v_i \neq v_j$ with the distribution functions F_{v_i} and F_{v_j} computed after considering all voting pools they took place in, we define in Equation 4 the distance between their distribution functions:

$$d(v_i, v_j) = \sum_k (F_{v_i}(k) - F_{v_j}(k))^2 \qquad (4)$$

Now, consider the symmetrical matrix $D = (d_{i,j})$ of size $n \times n$, where its elements are defined as the distances $d_{i,j} = d(v_i, v_j)$. A row i of this matrix shows how statistically different is the behavior of worker v_i from the rest of workers in the population. The matrix D can be normalized to a matrix C to make the values of each row sum 1, by dividing each row by its own sum.

According to the theoretical findings (Section 3.2), the distances between naive-behaving saboteurs and the majority of the population should be large. Having in matrix C a measure of distance between any pair of nodes, we can use the EigenTrust algorithm of Kamvar *et al.* [3] (Algorithm 1), to give each node a single global score (its corresponding eigenvalue). The score of each node tells us how likely is that node to be dishonest. Kamvar *et al.* proved that the algorithm will converge to some global scores vector, $\vec{t}$, if the initial matrix C is not singular. More, the global vector $\vec{t}$ contains only positive values with $\sum t_i = 1$.

Algorithm 1 The simple EigenTrust algorithm [3]

Input data:
$C = (c_{i,j})$ a matrix of size $n \times n$, with $\sum_j c_{i,j} = 1$
some small error ϵ
$\vec{t}^0 = (t_i^{(0)})$, with $t_i^{(0)} = \frac{1}{n}$, for every $1 \leq i \leq n$
repeat
  $\vec{t}^{(k+1)} \leftarrow C^T \vec{t}^{(k)}$
  $\delta \leftarrow \|\vec{t}^{(k+1)} - \vec{t}^{(k)}\|$
until $\delta < \epsilon$

To avoid obtaining singular matrices, we remove from C the rows and columns for workers that scored only 0 votes against in all their voting pools. After we compute $\vec{t}$, we sort the scores of the nodes in ascending order considering that each value represents a discrete probability and we compute their corresponding distribution function. In Figure 3 we depict a particular case for this distribution function (for a population of 1000 workers processing on average $N = 30$ work units each, with $f_1 = f_2 = 0.1$, $s_1 = s_2 = 0.5$). In

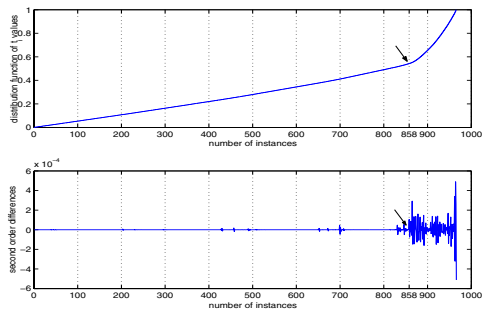


Figure 3: Distribution function and second order differences for the $\vec{t}$ values

most of our experiments we got a clear inflection in this plot (indicated by the arrow), resulting from the differences between naive saboteurs and remaining population. In the absence of an inflection, the algorithm assumes that all workers are either honest or M_2-type (in fact some of the workers could be M_3 behaving mostly as M_2).

To locate the zone of the inflection, we use the second order differences of $\vec{t}$ values, as these emphasize in a clearer way the fast growth in that zone. We consider 10 consecutive values in the second differences and we compute their statistical variance. The inflection shows up when these variances go above a given threshold (Equation (5)). In our experiments we set $\vartheta = 10^{-8}$. We also tried other thresholds with a difference of up to 3 orders of magnitude and we noticed no sensible difference. Thus, we can consider $\theta = t_{i_{cr}}$ and classify as naive malicious all workers i such us $t_i \geq \theta$.

$$i_{cr} = \max\{i | var(\overline{t_{i-10}, t_i}) < \vartheta\} \quad (5)$$

3.4 A general sabotage tolerance protocol

The theoretical modeling using multinomial experiments presented in Section 3.2 allowed us to define the classification procedure presented in Section 3.3. With a good certainty, the master can identify malicious workers, especially those of type M_1, while keeping the classification error low. A low classification error means that a small number of false positive workers, which are in fact honest workers, are reported by the classification scheme. In this section we go further and define our general sabotage tolerance protocol.

Since actual replication is effective to defeat naive saboteurs, our protocol identifies those cases where a worker that is not classified as (naive) malicious is defeated, and asks further replication on those voting pools. Specifically, the master has to employ the general algorithm described in Algorithm 2.

Up to line 4 in Algorithm 2, the master applies classical replication. In line 5 the master selects the conflicting voting pools out of the initial replication results. Next, in line 6, the master applies the classification algorithm of Section 3.3 and obtains a list of malicious workers. In line 7, the master selects among the conflicting voting pools those where another worker not a malicious one is defeated. Each suspect voting pool is further audited (line 8) by putting the tasks on honest workers. The honest workers are selected from the ones that recorded zero votes against or from the ones that registered the smallest $\vec{t}$ values in the classification procedure. At the end of this step, the master identifies those voting pools $\mathcal{S}_{V,err}$ where the initial result was reverted. From these voting pools, the master identifies the colluding workers (line 9). Next, in step 10, the master traces back all non-conflicting voting pools where three malicious workers where initially assigned. On each of these voting pools, the master invalidates the initial quorum and asks a new 3-times replication with honest workers as above. In the end, the master accepts the results of each voting pool with a majority voting.

Algorithm 2 The general sabotage tolerance algorithm

1: **Input data:**
2: $\mathcal{S}_W$: the set of work units, $\mathcal{S}_P$: the set of workers
3: **Begin**
4: $\mathcal{S}_V \leftarrow$ *Distribute_tasks*($\mathcal{S}_W$, $\mathcal{S}_P$, 3);
5: $\mathcal{S}_{V,conflicting} \leftarrow$ Select conflicting pools from $\mathcal{S}_V$
6: $\mathcal{S}_{Mal} \leftarrow$ Identify malicious workers
7: $\mathcal{S}_{V,suspect} \leftarrow$ Select suspect pools from $\mathcal{S}_{V,conflicting}$
8: $\mathcal{S}_{V,err} \leftarrow$ Ask 2 more responses on pools from $\mathcal{S}_{V,suspect}$
9: $\mathcal{S}_{M_2 \cup M_3} \leftarrow$ Identify colluding workers from $\mathcal{S}_{V,err}$
10: $\mathcal{S}_{V,suspect1} \leftarrow$ Identify voting pools with consensus of only colluding workers
11: Ask a new voting pool on every $V \in \mathcal{S}_{V,suspect1}$
12: Accept the results for all $V \in \mathcal{S}_V$ by majority voting
13: **End**

4 Results and discussion

4.1 Results

In our analysis, we considered various population structures. Each worker gets on average $N = 30$ tasks (i.e. 10000 work units for a population of 1000 workers). For each population structure we have run 100 experiments (we show only the average values). We computed the final error rate and redundancy obtained with our scheme and we compared them with the one obtained with the simple replication, before applying our sabotage tolerance protocol. We compared the actual redundancy of our scheme against a "theoretical" redundancy that would be obtained if the sabotage tolerance protocol would ask for another task replica

on each voting pool with conflicting responses. This theoretical redundancy is an optimistic value because it is still not enough for establishing the correct result of a conflicting voting pool.

We can note in Figure 4a that with only M_1-type naive workers our sabotage tolerance protocol works pretty well, increasing the effectiveness of the replication by at least 10 times and avoiding the verification of each conflicting voting pool (Figure 4b). With only M_2-type workers (Figure 4c), the sabotage tolerance protocol works in its full power if the workers are sabotaging with rates greater than 0.3, i.e., $s_2 \geq 0.3$. For smaller values of s_2, like 0.05, results of our algorithm are not so good, but even in this case, when simple replication is effective, our protocol succeeds to improve error rate by about 10 times. We can note that defeating all colluding saboteurs (the cases with big sabotage rates) is done with the cost of a bigger redundancy (Figure 4d), as for every conflicting voting pool we ask two new results. But, we should note that redundancy is still lower than 4 and the percentage of the saboteurs in the population is very high.

In Figures 4e, 4f and 4g we considered that the naive workers are in a small, medium or large proportions ($f_1 = 0.05$, $f_1 = 0.2$ and $f_1 = 0.4$) and we varied the structure parameters regarding the colluding workers. We can notice that if the naive workers do not overwhelm the colluding ones ($f_1 = 0.05$ and $f_1 = 0.2$) then the ST protocol is very effective in spotting out the collusion, especially on the cases when the colluding workers are well defined (the sabotage rate is big enough). For the case when $f_1 = 0.4$, colluding workers have only a very small influence on the overall and we got a situation similar with a pure M_1 population. Still, we get 10 times improvement in the error rate. In this mixed case, the redundancy is in between the pure population cases. These redundancy plots were omitted to to space constraints.

As we discussed previously, an M_3-type worker is a hybrid one. Therefore, the results for populations consisting on M_3-type workers do not differ too much from the one presented up to now, being similar with the case of mixed M_1 and M_2 populations. We omitted these figures to conserve space. Mixing M_3-type workers with pure M_1 and M_2 ones does not change the results.

4.2 Discussion

Results of Section 4.1 showed that (i) we succeed to keep the error rate in the acceptable limit of 10^{-4} for the most majority of cases; (ii) if the malicious workers reveal their colluding profile with high consistency (some sabotage rate $s_2 \geq 0.3$), our sabotage tolerance heuristic spots them successfully, even if the number of saboteurs is large; (iii) in all cases, we get at least 10 times improvement over simple replication, without a meaningful increase of redundancy. Even in the worst case (with a large number of very effective colluding saboteurs), the redundancy remains below an entire additional replication per work unit.

From the experiments it appears that the most difficult situation for our sabotage tolerance approach occurs when there are many colluding saboteurs (e.g. $f_2 = 0.4$) and when they sabotage very infrequently ($s_2 = 0.05$). Here, our protocol succeeds to lower the error rate, but it still remains around 10^{-3}. If possible, a solution might be to increase the number of voting pools per worker (N).

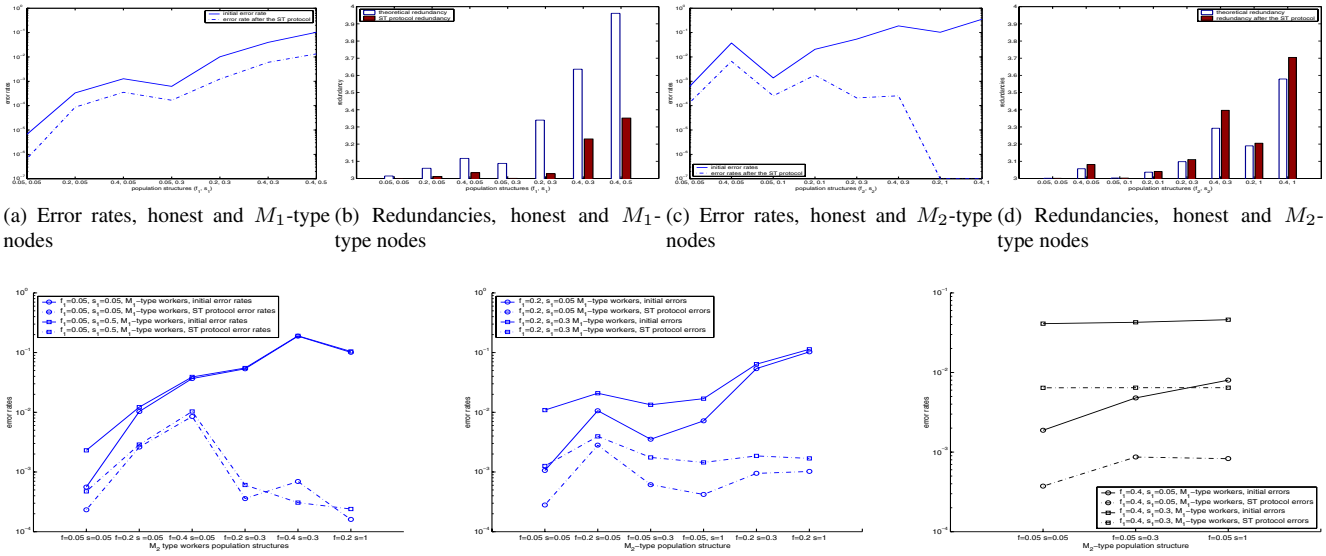
Another difficulty of the ST protocol occurs when the number of naive malicious workers is large ($f_1 = 0.4$). The effectiveness of our ST protocol is closely related to the weakness of replication in these situations, as shown in [7]. In fact, although we succeed to get improvements, to increase performance one might need to increase m. Another issue with our heuristic concerns the fact that we do not eliminate completely all erroneous results. This results from a number of facts. First, we select "honest" workers to verify the suspicious results. Although we have a very good confidence that our selected workers are honest, we can not eliminate the possibility of selecting malicious workers instead. This situation can happen with higher probability if the number of saboteurs is very big. Second, the classification algorithm of Section 3.3 is tuned for a compromise between error classification (false positives) and recall (total number of real positives identified). Simultaneously improving both is very hard or even impossible to achieve.

Regarding the computational effort, the matrix multiplication algorithm is the most costly part (the analysis is in Kamvar *et al.* in [3]). However this can be done off-line.

5 Conclusion

In this paper, we presented an algorithm that targets colluding nodes in desktop grid systems. We argued that simple majority voting is powerless against colluding behavior and we observed that any DG system needs at least 3 replicas to defeat such nodes. Then, we proposed an algorithm that uses off-line processing on a moderately large set of voting pools to spot malicious nodes, before accepting *any* computation from volunteers. To evaluate our approach we used three types of nodes, ranging from naive (M_1) to colluding (M_2) ones, including nodes with commuting behavior (M_3). Our experimental results show that our statistical approach identifies well the nodes acting in a naive way, leaving only the colluding (M_2) nodes undetected. Then, we go after M_2 nodes on a voting-pool-by-voting pool basis, whenever we find conflicting results. We succeed to keep the overall error rate low, even in the presence of smart colluding nodes.

As future work, we intend to further improve and sim-



(a) Error rates, honest and M_1-type nodes (b) Redundancies, honest and M_1-type nodes (c) Error rates, honest and M_2-type nodes (d) Redundancies, honest and M_2-type nodes

(e) Error rates, honest M_1 and M_2-type nodes (f) Error rates, honest M_1 and M_2-type nodes (g) Error rates, honest M_1 and M_2-type nodes

Figure 4: Results obtained for various population structures

plify our mechanisms for finding M_1 and M_3 nodes. Additionally, we believe that one important limitation of our work results from the assumption that nodes are not be aware of the algorithm that the master uses to spot them. Although the M_3 model tries to conceal its pattern of behavior, it falls short on that attempt and we still do not have some formal proof or some evidence showing that no model can defeat our sabotage tolerance scheme.

Acknowledgments

This work was funded by the EC IST FP6 project CoreGRID — project No.004265. G.C. Silaghi was also supported by the Romanian National Authority for Scientific Research under the project IDEI_573. The authors would also like to thank the anonymous referees for comments and suggestions.

References

[1] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *5th Intl Workshop on Grid Computing (GRID 2004), 2004, USA, Proceedings*, pages 4–10. IEEE Computer Society, 2004.

[2] P. Domingues, B. Sousa, and L. M. Silva. Sabotage-tolerance and trust management in desktop grid computing. *Future Generation Comp. Syst.*, 23(7):904–912, 2007.

[3] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proc. of the 12th intl. conf. on WWW*, pages 640–651. ACM Press, 2003.

[4] S. Karlin and H. M. Taylor. *A First Course in Stochastic Processes, Second Edition*. Academic Press, 1975.

[5] J.-S. Kim, B. Nam, M. A. Marsh, P. J. Keleher, B. Bhattacharjee, D. Richardson, D. Wellnitz, and A. Sussman. Creating a robust desktop grid using peer-to-peer services. In *21th Intl. Parallel and Distributed Processing Symp. (IPDPS 2007), Proceedings, 2007, USA*. IEEE, 2007.

[6] D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. M. Silva, G. Fedak, and F. Cappello. Characterizing result errors in internet desktop grids. In *Euro-Par 2007, Parallel Processing, 13th Intl Euro-Par Conf, France, 2007, Proceedings*, volume 4641 of *LNCS*, pages 361–371. Springer, 2007.

[7] L. F. G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Comp. Syst.*, 18(4):561–572, 2002.

[8] B. Wei, G. Fedak, and F. Cappello. Collaborative data distribution with bittorrent for computational desktop grids. In *4th Intl Symposium on Parallel and Distributed Computing (ISPDC 2005), 2005, France*, pages 250–257. IEEE Computer Society, 2005.

[9] S. Wong. An authentication protocol in web-computing. In *20th IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS 2006), Proceedings, 2006, Greece*, 2006.

[10] M. Yurkewych, B. N. Levine, and A. L. Rosenberg. On the cost-ineffectiveness of redundancy in commercial p2p computing. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 280–288. ACM, 2005.

[11] S. Zhao, V. M. Lo, and C. GauthierDickey. Result verification and trust-based scheduling in peer-to-peer grids. In *5th IEEE Intl Conf. on Peer-to-Peer Computing (P2P 2005)*, pages 31–38, Germany, 2005.