Journal of Grid Computing manuscript No.
(will be inserted by the editor)

Defeating Colluding Nodes in Desktop Grid Computing Platforms

Gheorghe Cosmin Silaghi · Filipe Araujo · Luis Moura Silva · Patricio Domingues · Alvaro E. Arenas

Received: February 20, 2009 / Accepted: date

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 computing projects exert no administrative or technical control on volunteers. These can very easily output bad results, due to software or hardware glitches (resulting from over-clocking for instance), to get unfair computational credit, or simply to ruin the project. To mitigate this problem, Desktop Grid servers replicate work units and apply majority voting, typically on 2 or 3 results.

In this paper, we observe that simple majority voting is powerless against malicious volunteers that collude to attack the project. We argue that to identify this type of attack and to spot colluding nodes, each work unit needs *at least* 3 voters. In addition, we propose to post-process the voting pools in two steps. *i*) In the first step, we use a statistical approach to identify nodes that were not colluding, but submitted bad results; *ii*) then, 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.

Keywords Desktop grids · volunteer computing · collusion · replication

1 Introduction

Internet Desktop Grids [2,3] aggregate huge distributed resources over the Internet and make them available for running a growing number of applications. In 2007, BOINC [2], the most popular desktop grid (DG) platform, runs about 40 projects, aggregating more than 400,000 volunteer computers yielding on average more than 400 TeraFLOPS [1]. Numbers are even larger by now.

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 during weeks or even months [5].

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. In particular, BOINC uses replication with majority voting which can bring an error rate of about 1×10^{-5} by validating each result with only two similar responses [12]. But, as Zhao *et al*. [21] acknowledge, a potential threat comes up when workers can devise some scheme to interact, for example, with a distributed hash table.

This is an extend version of the paper that was presented as part of the PCGrid 2008 workshop at the 22nd IPDPS conference, Miami, USA, 2008

G.C. Silaghi - corresponding author
Babes-Bolyai University, Str. Theodor Mihali 58-60, 400591, Cluj-Napoca, Romania, Tel.: +40-722817582, Fax: +40-264412570
E-mail: gheorghe.silaghi@econ.ubbcluj.ro

F. Araujo · L.M. Silva
University of Coimbra, 3030-290, Coimbra, Portugal
E-mail: {filipius, luis}@dei.uc.pt

P. Domingues
Polytechnic Institute of Leiria, 2411-901, Leiria, Portugal
E-mail: patricio@estg.ipleiria.pt

A.E. Arenas
STFC, Rutherford Appleton Laboratory, Chilton, Didcot, OX11 0QX, UK
E-mail: a.e.arenas@rl.ac.uk

However, there are many signs suggesting that this kind of peer-to-peer interaction among peers will become a standard in the near future. For example, collaborative techniques are very attractive for data distribution [17,4], especially when the DG runs a parameter sweep application. More, researches advance in the direction of an entirely distributed P2P desktop grid [11]. Some of the developers of XtremWeb [3] are working on BitDew [7], which aims to provide services for management and distribution of data on DGs. While very powerful and well intentioned, all these solutions bring a side effect with them: they can help malicious workers to team up to defeat the project, thus violating the workers' independence assumption.

Further, a P2P-enhanced desktop grid might become the target for a Sybil attack [6]: an individual either creates multiple identities which appear as individual ones to the master or gets control through a virus over a large number of workers. Such a powerful individual can develop a collective malicious behavior if the platform allows for peer-to-peer interaction.

This brings new challenges to the design of a desktop grid system, 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, which is the most popular ST technique employed in the nowadays middleware. 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 tool 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 in order to conclude about the valid result. We first presented this technique in [16]. In this paper, we present a larger set of experiments, including comparisons with k-means [13], one of the most well-know clustering algorithms.

In contrast to other works on ST in DGs [15,21], 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. We describe the actual existing sabotage tolerance techniques and up-to-date conclusions about their effectiveness. We define the type of saboteurs our approach intents to cover and make a discussion about how efficient the respective sabotage strategies are. In Section 3 we present our collusion-resistant sabotage tolerance technique. We start by showing how we can statistically model the voting behavior of workers and how we can classify the workers in malicious and not malicious ones. Next, we present our global sabotage tolerance protocol. In Section 4 we present and discuss the results obtained with our sabotage tolerance protocol. Finally, 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 [12].

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 spend computing resources only for verification purposes.

2.1 Related work

In this section we present previous work regarding sabotage tolerance in desktop grids. All the methods herein reviewed assume independence between workers.

BOINC, the most popular Internet desktop grids uses *replication* with majority voting [2,15] as the sabotage tolerance 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 being the correct one. Each collected result is seen as a vote in a voting pool with $2m-1$ voters and with majority agreement being the decision criteria. With the same sabotage model, Wong [19] presents a variation of the replication with only 2 replicas, by considering the workers arranged as nodes in a graph and connected by the work units. 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* [21], where a work unit with a known result is distributed at random to workers. Workers' 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 [15] use conditional probabilities of the 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. [20] 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.* [12] performed the first study that characterizes the errors in Internet Desktop Grids. They approached only I/O errors and discussed the efficiency of the above sabotage tolerance methods. In their experiment, they observed that 35% of hosts provide at least one error, the average error rate of an erroneous host is 0.0034, yielding for a global error rate over all hosts of 0.0022. Moreover, 10% of the erroneous hosts produce more than 70% of all errors, with an average error rate of 0.0335. Additionally, distribution of errors over time seems to be non-stationary, which reduce the effectiveness of spot-checking and credibility-based systems, because they depend on high consistency of the error rates. 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 a host error rate of 1×10^{-5}. This error rate is a must, if the overall acceptable application error rate is 10^{-2}. Therefore, we will use the 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 the *colluding nodes* that make their behavior depend from the participation of other malicious nodes in the 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 the basic naive malicious node by M_1*-type*. A M_1-type worker submits bad results with a constant probability s, called *sabotage rate*. This naive sabotage model assumes that workers are independent of each other and do not communicate. Even if independent workers which do not communicate are very unlikely to submit the same erroneous result, as the sabotage tolerance literature suggests [15], from now on, we assume that all submitted erroneous results are similar, regardless whether the workers can communicate or not.

If we assume the existence of a fraction f of M_1-type saboteurs in the total population of workers, then the expected error rate $\varepsilon_{M_1}(f,s,m)$ of the majority-voting replication is given by Equation (1) [15]:

$$\varepsilon_{M_1}(f,s,m) = \sum_{j=m}^{2m-1} \binom{2m-1}{j} (fs)^j (1-fs)^{2m-1-j} \qquad (1)$$

Unlike the basic M_1-type, a colluding saboteur (further referred as M_2*-type*) 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. Thus, a M_2-type malicious worker will never sabotage without winning the decision in its voting pool. We assume that there is a complete graph connecting all the malicious nodes, such that communication between any two malicious nodes is always possible at any point in time. 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 and they act to conceal themselves from majority voting. With this assumption, colluding saboteurs will attack whenever they are sure they can win the voting decision against honest workers.

If the fraction of M_2-type saboteurs in the total population of workers is f, each saboteur being an active one with probability s (i.e. s is the probability of a colluding saboteur to launch the coalition-formation protocol), each of them knowing all the rest of the workers, then the expected error rate is given by Equation (2).

$$\varepsilon_{M_2}(f,s,m) = \sum_{j=m}^{2m-1} \binom{2m-1}{j} \left(1-(1-s)^j\right) f^j (1-f)^{2m-1-j} \tag{2}$$

In eq. (2) we have to mention that *at least one* of the j M_2-type workers in a voting pool is sufficient to start (with probability s) the collusion formation protocol, the remaining colluding saboteurs in the voting pool participating by default to the coalition. This leads[1] to the multiplication term $1-(1-s)^j$ before f^j, which denotes the probability of finding exactly j M_2-type workers in a voting pool of size $2m-1$.

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, we denote with c 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. In this case the expected error rate is the one of Equation (3):

$$\varepsilon_{M_3}(f,c,s_1,s_2,m) = c\varepsilon_{M_1}(f,s_1,m) + (1-c)\varepsilon_{M_2}(f,s_2,m) \tag{3}$$

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 [12]), 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 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 $s_1 = s_2 = s$.

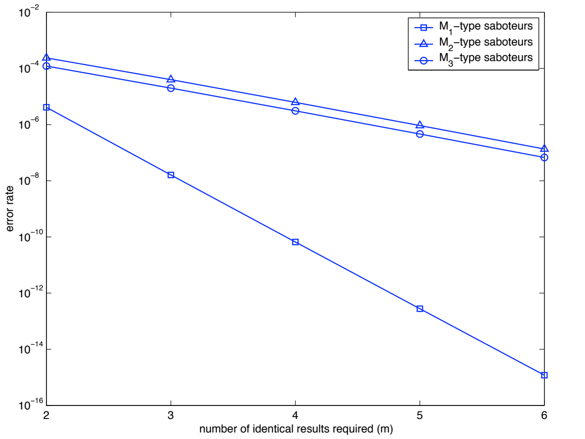


Fig. 1 Error rates comparison between various types of malicious workers against simple replication

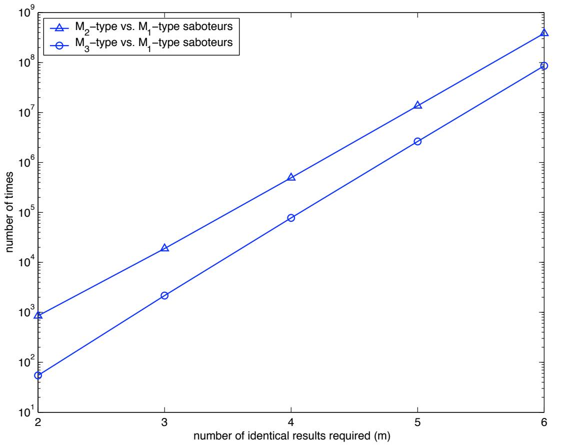


Fig. 2 Relative effectiveness comparison between various types of malicious workers against simple replication

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). Of course, the effectiveness of a naive saboteur increases with its sabotage rate s; this being the sole parameter such a saboteur can control. Figure 2 depicts the number of times colluding saboteurs of type M_2 and M_3 are more effective than the naive ones for various number of results required. The effectiveness of the saboteurs was computed for the same parameters as in figure 1. The effectiveness of the M_2-type saboteurs was computed assuming that they sabotage only when at least a coalition of size 2 is formed. The relative effectiveness of the colluding saboteurs increases exponentially with the number of results required, because the success rate of naive saboteurs is very small in the presence of higher-order replication.

[1] the overall probability that at least one worker out of j starts the collusion formation protocol, given the individual probability s is: $\binom{j}{1}s(1-s)^{j-1}+\binom{j}{2}s^2(1-s)^{j-2}+\ldots\binom{j}{j}s^j = 1-(1-s)^j$

[2] We assumed the same error rate parameters as for the top 10% erroneous hosts reported by Kondo *et al*. [12].

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. Since the actual replication works very well in the presence of M_1-type naive saboteurs, we do not intend to replace it. Instead, we complement it with a scheme targeted to spot and defeat colluding saboteurs, which are much more effective and can defeat the replication, as we have seen in Section 2.3.

3.1 Overview

Kondo *et al*. [12] demonstrated that replication with $m = 2$ is enough to cope with erroneous hosts with M_1 saboteurs. Additionally, we observe that DG projects that we are aware of, set m to be 2 at most, while some of them initially use only two replicas and ask for another one in case of conflicting responses. Therefore, in our work we fix $m = 2$. This means that the master replicates each task $2m - 1 = 3$ times. However, instead of deciding on a result as soon as the master gets a majority of 2 similar responses, it will wait and 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 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.

The master's objective is to spot out malicious workers, regardless of the sabotage model they fit in. From this point of view, a voting pool that contains contradicting votes is of interest for the master, because it contains, at least one faulty node. If the size of the voting pool is 3, this means that one loosing worker voted against two opponents. 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 saboteurs, for the same f and s, regardless their value. This makes it easier for the master to spot out naive saboteurs than colluding ones. 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 with colluding ones. Therefore, given that M_2 and M_3 nodes are not aware of our sabotage tolerance mechanism and only try to defeat majority voting, we expect a better response against M_3 than against M_2 saboteurs. Figure 3 shows the percentage of conflicting voting pools for various s, given $f = 0.1$, in pure populations with only M_1-type (respective M_2-type) saboteurs.

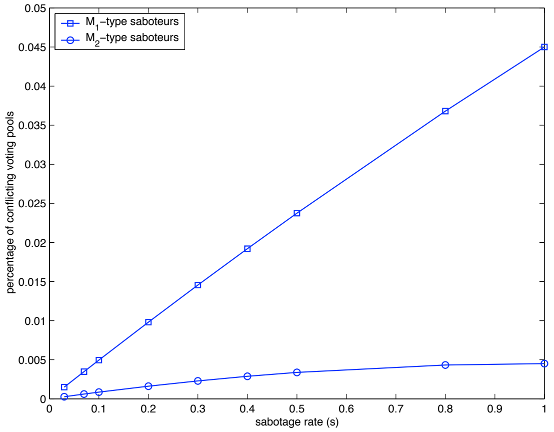


Fig. 3 Comparison of conflicting voting pools

Considering a conflicting voting pool, it would be of interest for the master to assess if the worker that is losing the decision is behaving like a naive M_1-type saboteur or not. The master can do this assessment if it possesses a theoretical model of the voting pools world and a classification tool, which we will describe in the following section.

3.2 Statistical modeling of the voting behavior

Consider a population $\mathscr{S}_P$ 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 the workers fully comply with their models during all their life. Additionally, there is an implicit assumption in the models of workers that we devised: nodes are unaware of the algorithm used by the master to spot collusion. Let the master distribute replicated tasks from a set of work units $\mathscr{S}_W$, such that, on average every worker gets N tasks.

A voting pool $V = \{v_1, v_2, v_3\}$ is a set of three $(m = 2)$ different workers $v_i \in \mathscr{S}_P$, each of which submitting a binary vote in the pool. Consider a fixed worker $v \in V$. The number of *votes against* the worker collected in the voting pool V can be modeled as a random variable $Y_v : \{0, 1, 2\} \rightarrow \mathbb{R}$, 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*) worker's fully compliance with its model; and *iii*) the fact that workers can not influence how the master distributes them in

Table 1 Parameters describing the population structure

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

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 $Y_{v,N} : \{0,1,\dots,2N\} \to \mathbb{R}$ 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 (4):

$$(p_{v,0} + p_{v,1}X + p_{v,2}X^2)^N \tag{4}$$

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 [10].

As an example, if the random variable of a worker v is $Y_v = \{0.6, 0.2, 0.2\}$, meaning that the worker scored 0 votes against in 60% of cases, 1 votes against in 20% of cases and 2 votes against in 20% of cases, if the worker participated in 5 voting pools, then random variable $Y_{v,5}$ can be obtained by computing the polinomial $(0.6+0.2X+0.2X^2)^5$: which gives $0.0778+0.1296X+0.216X^2+0.2016X^3+0.1776X^4+0.1059X^5+0.0592X^6+0.0224X^7+0.008X^8+0.0016X^9+0.0003X^{10}$. The random variable $Y_{v,5}$ consists of the coeficients of the before-computed polynomial and should be read as follows (e.g.): the probability that after 5 voting pools the worker v registers 6 votes against is 5.92%.

The *joint distribution function* of a voter v with $Y_{v,N}$ is $F_v : \{0,1,\dots 2N\} \to \mathbb{R}$, defined as $F_v(i) = Prob(Y_{v,N} \leq i)$, $F_v(i)$ being the summation of all coefficients of the polynomial (4) up to the i rank.

In the heart of our heuristic lies a simple intuition about the distribution functions F_v: if we compare F_v for a honest node and for an M_1 malicious node there is a huge separation between both lines, because a typical honest node gets much fewer votes against than a typical M_1 node. 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 (4) using multiplications of polynomials, we got distribution function curves like the ones depicted in Figure 4. 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 4(a) we considered only naive M_1-type workers. In Figure 4(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 4(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.

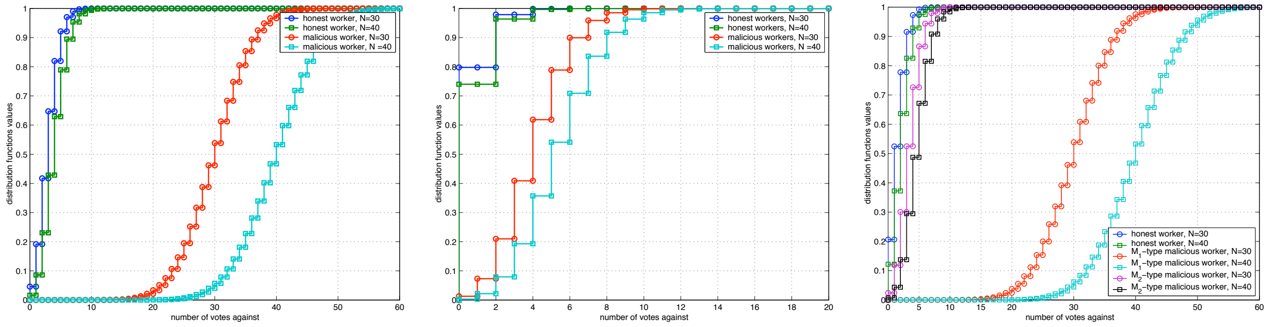
After we analyzed extensively various population structures, using the mathematical procedure explained in the above paragraph, the following important conclusions can be drawn out:

- M_1-type (naive) saboteurs always collect the biggest number of votes against, their joint distribution functions being the most-right ones in the graphic;
- for N large enough, there is a clear separation between the distribution functions F_v for the case of honest workers versus malicious workers;
- 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 biggest ones;
- 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, being placed on the left side of the distribution for the M_1-type workers.

3.3 Spotting out naive saboteurs (M_1 or M_3)

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.

[3] Given 2 random variables $Y_1 : \{x_i,\, i = 1,\dots,n_1\} \to \mathbb{R}_+$, $Y_1(x_i) = p_i$, $\sum_i p_i = 1$ and $Y_2 : \{x'_i,\, i = 1,\dots,n_2\} \to \mathbb{R}_+$, $Y_2(x'_i) = p'_i$, $\sum_i p'_i = 1$, the product $Y = Y_1Y_2$ is defined by over the space $\{x_i \wedge x'_j,\, i = 1,\dots,n_1,\, j = 1,\dots,n_2\}$ with the following expression: $Y(x_i \wedge x'_j) = p_i p'_j$.



(a) Honest and M_1-type workers

(b) Honest and M_2-type workers

(c) Honest, M_1 and M_2-type workers

Fig. 4 Theoretical distribution functions F_v for various population structures

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 4) for that worker. Applying the procedure described in Section 3.2, the master will obtain one distribution function (similar to the ones of Figure 4) for each worker.

Figure 5 depicts the distribution functions for workers participating in an experiment with a population structure with $f_1 = f_2 = 0.1, s_1 = s_2 = 0.5$ and $f_3 = 0$, for $N = 30$ and for a small number of nodes (in order to facilitate the display of the distributions on the plot). As expected from the theoretical results presented above, the distribution functions of M_1-type workers (solid lines) will agglomerate the right side of the plot, while the ones of the honest workers (circled lines) and M_2-type workers (squared lines) will stay on the centre and left side. The honest workers that were not placed in voting pools with malicious opponents have the most left-sided distributions.

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 5 the distance between their distribution functions:

$$d(v_i, v_j) = \sum_k (F_{v_i}(k) - F_{v_j}(k))^2 \qquad (5)$$

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.* [9] (Algorithm 1), to give each node a single global score (its corresponding eigenvalue).

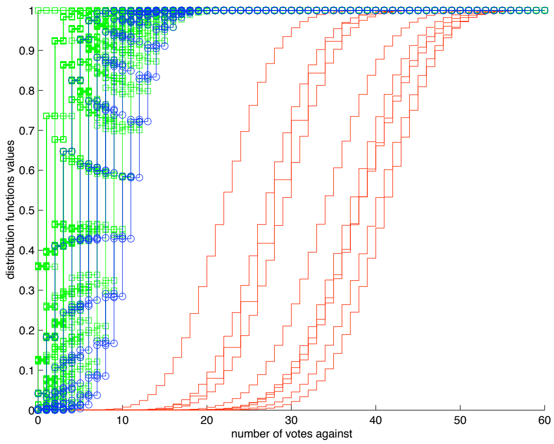


Fig. 5 Distribution functions for a population with M_1 and M_2-type saboteurs

EigenTrust algorithm aggregates private reputation values of a node for other nodes in the network in order to supply with the global reputation value for each node. Given that two nodes are assigned with similar private reputation values from the rest of the nodes in the network, EigenTrust produces global values closed each to another for those two nodes. In our case, the score produced by EigenTrust for 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, $\mathbf{t}$, if the initial matrix C is not singular. More, the global vector $\mathbf{t}$ contains only positive values with $\sum t_i = 1$.

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 $\mathbf{t}$, we sort the scores of the nodes in ascending order assuming that

Algorithm 1 The simple EigenTrust algorithm [9]

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

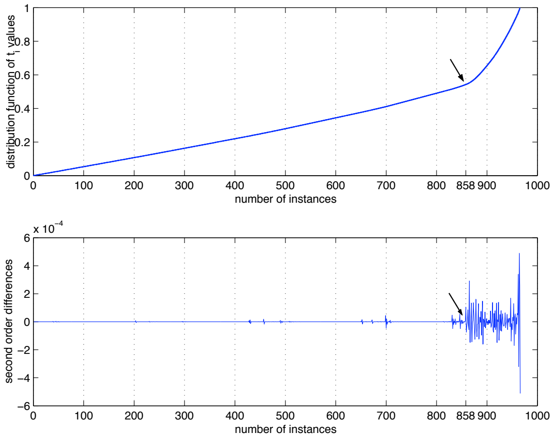


Fig. 6 Distribution function and second order differences for the **t** values

each value represents a discrete probability and we compute their corresponding distribution function.

In Figure 6 we depict a particular case for this distribution function (for a population of 1000 workers processing on average $N = 30$ work units each, with $f_1 = f_2 = 0.1$, $s_1 = s_2 = 0.5$). The distribution function registers less than 1000 t-values, because we removed from the population the workers that scored 0 votes against in all their voting pools (around 50 workers in our case).

In most of our experiments we got a clear "knee" (indicated by the arrow) in this plot, resulting from the differences between naive saboteurs and remaining population. Identifying this knee, we can classify the workers in naive malicious and not naive malicious ones. In the absence of a knee, our algorithm should just not mark any worker as naive malicious and assume that all of them are either honest or M_2-type nodes (in fact some of the nodes could be M_3 behaving mostly as M_2).

To locate the knee, when it exists, we use the second order differences[4] of the vector **t** values, as these emphasize in a clearer way the fast growth in the zone of the knee. We consider 10 consecutive values in the second differences and we compute their statistical variance. The knee shows up when these variances go above a given threshold (see Equation (6)). 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 $\theta = t_{i_{cr}}$, where i_{cr} is given by eq. (6) and classify as naive malicious all workers i such us $t_i \ge \theta$.

$$i_{cr} = \max\{i | var(\overline{t_{i-10}, t_i}) < \vartheta\} \tag{6}$$

In section 4 we will present the classification results. We should be aware that our final goal is the identification of the colluding saboteurs, while identifying with high certainty the naive saboteurs represents only an intermediary step.

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 high certainty, the master can identify malicious workers, especially those of type M_1, while keeping the classification error low - as we will see in section 4.1. 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.

Because 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. Therefore, we do not replace the actual replication-based sabotage tolerance protocol; rather we complement it with a tool to spot out situations when colluding saboteurs win against honest ones. Specifically, the master has to employ the general algorithm described in Algorithm 2.

Algorithm 2 The general sabotage tolerance algorithm

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

Up to line 4 in Algorithm 2 the master applies the classical replication. In line 5 the master selects the conflicting voting pools out of the initial replication results. Next, in

[4] Given the vector $X = \{x_1, x_2, \ldots x_n\}$, the first order difference vector ∇X is defined as the vector $\nabla X = Y = \{y_1, \ldots, y_{n-1}\}$ with $y_i = x_{i+1} - x_i$. The second order difference vector $\nabla^2 X$ is defined as $\nabla^2 X = \nabla Y$.

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. On each of these suspect voting pools, the master ask at most two new response replicas (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 **t** values in the classification procedure. At the end of this step, the master identifies those voting pools $\mathscr{S}_{V,err}$ 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 nonconflicting 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.

4 Results and discussion

4.1 Results

In this section we report the results achieved with our sabotage tolerance scheme. We considered various population structures and we let the master assign tasks such that 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, to get a statistical confidence on our results. However, for convenience, in our plots we show only the average values.

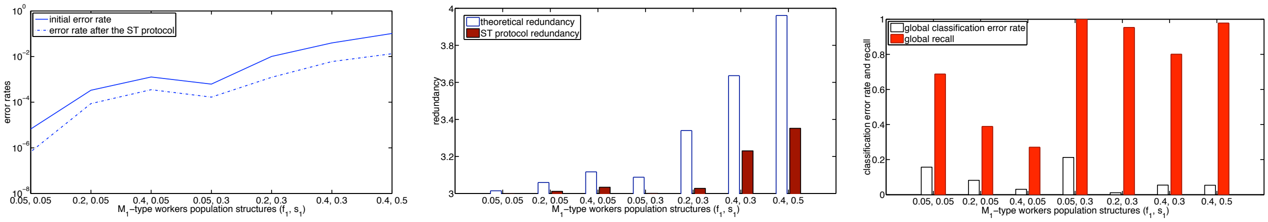
To evaluate the performance of the proposed sabotage tolerance scheme, we computed the final error rate and redundancy obtained with our scheme and we compared them against the ones obtained with the simple replication, before applying our sabotage tolerance protocol. For a cost estimation, we also compared 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.

But, we are also interested in discovering the colluding saboteurs, i.e. the ones that get defeated after applying the algorithm presented in section 3.4. We can see this task as a classification one and evaluate its performance by computing the classification error and the recall rates. As advised in the data mining field [18], the *classification error* represents the percentage of incorrectly classified examples (false positives) out of the total retrieved ones for some class. The *recall* represents the percentage of the examples of some class identified by the automated classification procedure. We should note that obtaining a low classification rate is usually achieved with a cost of a low recall. In what follows, we will also report the classification error rates and recalls for the classification tasks performed by our sabotage tolerance protocol. More precisely, we will report the global results (i.e. classification error rate and recall with respect to all saboteurs, regardless of their profile) and the results concerning only the colluding saboteurs (i.e. saboteurs belonging to M_2 and M_3 types).

First, we applied our method in pure populations with only M_1 or M_2-type workers (Figures 7 and 8). We plotted the actual ST protocol error rates (with dotted lines) against the ones obtained with simple replication (with solid lines). We can note in Figure 7a 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 7b). With only M_2-type workers (Figure 8a), 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 8b), as for every conflicting voting pool we ask two new results. This is the reason why in this case the redundancy is higher than the theoretical redundancy. But, we should note that redundancy is still lower than 4 and the percentage of the saboteurs in the population is very high.

Regarding the classification tasks for the M_1-type pure populations (figure 7c), our protocol correctly retrieves almost all naive saboteurs with an acceptable low recall, if they sabotage consistently (i.e. $s_1 > 0.3$). Also, the recall is low if there are a considerable number of saboteurs. The worse results are obtained for the cases with $f_1 = 0.05$ or $s_1 = 0.05$, i.e. there are very few saboteurs or they do not reveal out their profile. In this case, classification rates are bigger (for example in the case $f_1 = 0.05$ or $s_1 = 0.3$) because the classification procedure is a statistical one and records some errors by classifying honest workers as M_1-type malicious ones. But, this situation does not affect globally our ST protocol because it introduces only very few additional auditing and redundancy. We can see in figure 7a that also in this case, a 10-times gain in ST protocol error rate is obtained. We should note that this population setup evaluates the performance of the initial classification procedure described in section 3.3, which spots out (with a good certainty) the naive saboteurs.

Regarding the classification tasks for the M_2-type pure populations (figure 8c), we should note that our algorithm

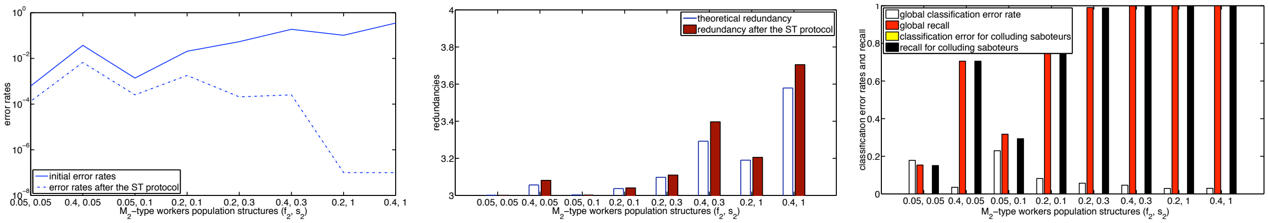


(a) Error rates

(b) Redundancies

(c) Classification error rates and recalls

Fig. 7 Results obtained for a population structure with only honest and M_1-type nodes



(a) Error rates

(b) Redundancies

(c) Classification error rates and recalls

Fig. 8 Results obtained for a population structure with only honest and M_2-type nodes

recalls almost all saboteurs if they are in an acceptable proportion ($f_2 > 0.05$) and they reveal their identity ($s_2 > 0.3$). The bigger global classification error rates for the cases when $f_2 = 0.05$ are associated with low recall. This means that step 6 of the algorithm 2 spots out very few naive saboteurs, letting (as expected) the forthcoming replication on suspect voting pools to spot out colluding workers with the price of the increased redundancy. We should also note that the classification error rate for colluding saboteurs is always 0 (figure 8c), because in M_2-type pure populations there are no other saboteurs to be classified as colluding malicious.

Next, we considered mixed population structures with naive and colluding workers against honest ones (figures 9, 10 and 11). Specifically, 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.

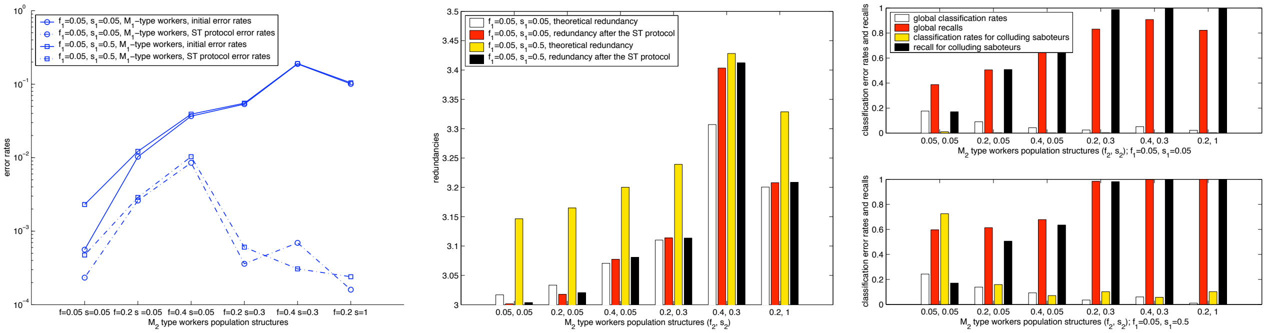
We can notice (figures 9a and 10a) that if the naive workers do not overwhelm the colluding ones (the cases when $f_1 = 0.05$ and $f_1 = 0.2$), then the ST protocol is very effective in spotting out the collusion, especially on the cases when the colluding workers are well defined (the sabotage rate is big enough). For the case when $f_1 = 0.4$ (figure 11a), 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 (figures 9b, 10b and 11b) is in between the pure population cases. In majority of cases, the redundancy is small, being very significantly below the theoretical one. We can notice that for the case when colluding workers are in a large proportion ($f_2 = 0.4$ - figure 9b), if the naive workers shows their profile by a big sabotage rate ($s = 0.5$), the redundancy is lower than the case of only pure M_2-type workers, as the naive saboteurs are spotted out by the procedure described in section 3.3.

In what regards the classification, the ST protocol correctly identifies most of the colluding saboteurs. We should note that big classification error rates for the colluding saboteurs are reported for the cases when naive saboteurs overwhelm the colluding ones ($f_1 > f_2$). But, this is achieved concurrently with a very low global classification error (see figures 10c and 11c). This means that our ST protocol also identifies naive saboteurs that happened to vote together and classified them as colluding.

As we discussed previously, a M_3-type worker is a hybrid one. We again considered various proportions of M_3-type saboteurs ($f_3 = 0.05$, $f_3 = 0.2$ and $f_3 = 0.4$) and naive rates ($c = 0.1$, $c = 0.5$ and $c = 0.9$) over a full combination of sabotage rates $s_{3,1}$ and $s_{3,2}$ (figures 12, 13 and 14).

The ST protocol results for populations consisting on M_3-type workers ((figures 12a, 13a and 14a)) 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. The redundancies show up a similar pattern as explained before. Again, our protocol is much effective when the M_3 saboteurs behave mostly as colluding ones ($c < 0.5$) and have a well defined colluding profile ($s_{3,2} \geq 0.3$). But, also, in the rest of cases, we obtain at least a 10 times improvement comparing with the basic replication. The results of the classification procedures are also very good, the algorithm identifying almost all colluding saboteurs.

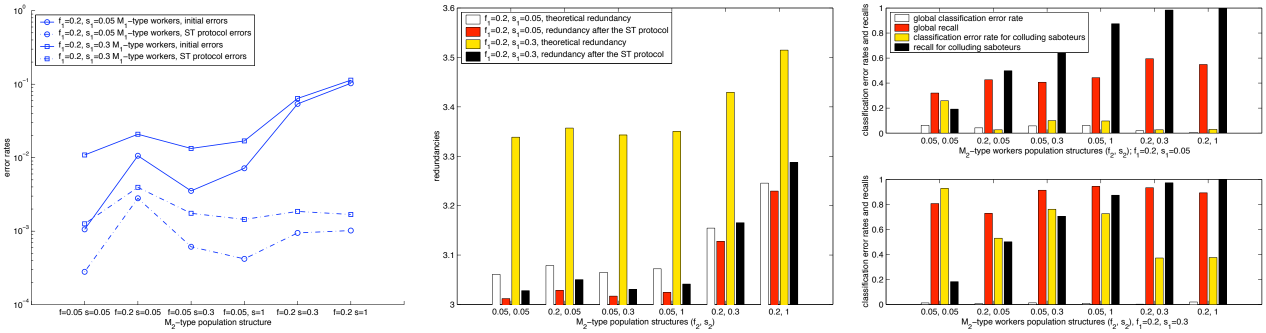


(a) Error rates

(b) Redundancies

(c) Classification error rates and recalls

Fig. 9 Results obtained for a population structure with honest, M_1 and M_2-type nodes, M_1-type workers being in a small ($f_1 = 0.05$) proportion

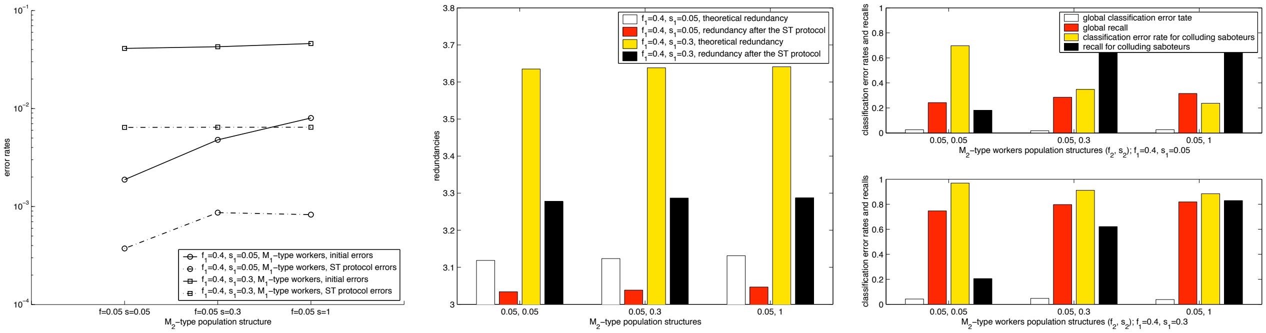


(a) Error rates

(b) Redundancies

(c) Classification error rates and recalls

Fig. 10 Results obtained for a population structure with honest, M_1 and M_2-type nodes, M_1-type workers being in a medium ($f_1 = 0.2$) proportion



(a) Error rates

(b) Redundancies

(c) Classification error rates and recalls

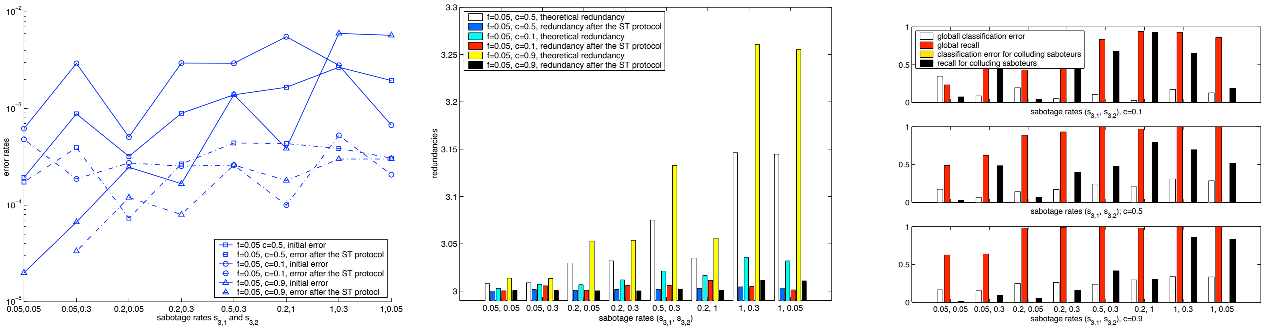
Fig. 11 Results obtained for a population structure with honest, M_1 and M_2-type nodes, M_1 -type workers being in a large ($f_1 = 0.4$) proportion

Mixing M_3-type workers with pure M_1 and M_2 ones does not change the above results.

4.2 Discussion

In the Section 4.1 we have drawn out the following conclusions:

- we succeed to keep the error rate in the acceptable limit of 10^{-4} for the most majority of cases
- 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
- in all cases, we get at least 10 times improvement comparing with the simple replication, without a meaningful increase of the redundancy. Even in the worst case (with

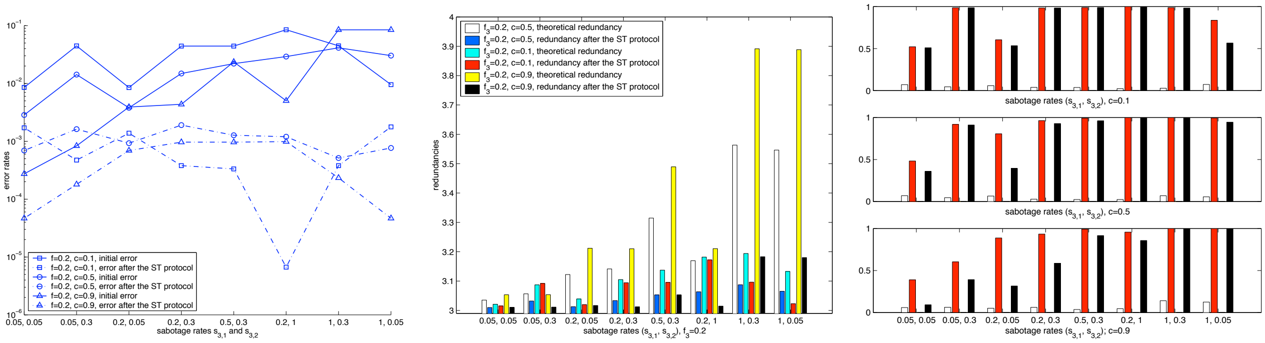


(a) Error rates

(b) Redundancies

(c) Classification error rates and recalls

Fig. 12 Results obtained for a population structure with M_3-type nodes and various naive rates, M_3-type workers being in a small ($f_3 = 0.05$) proportion

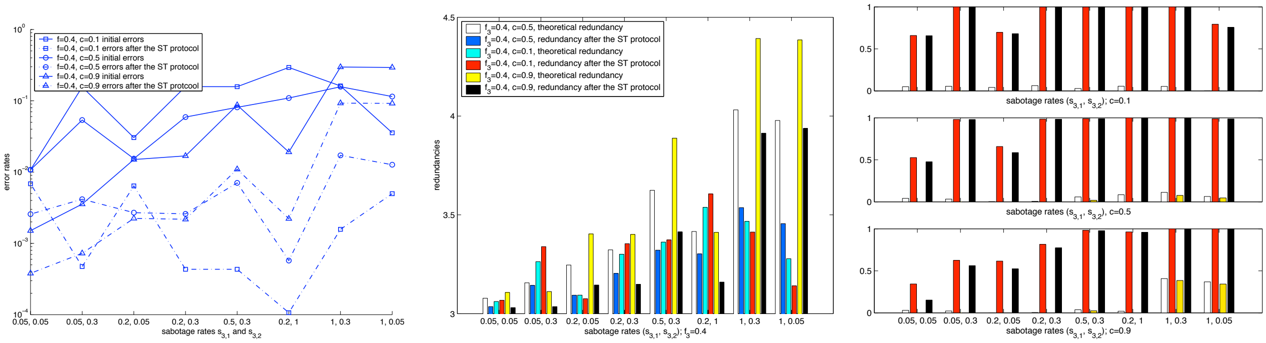


(a) Error rates

(b) Redundancies

(c) Classification error rates and recalls

Fig. 13 Results obtained for a population structure with M_3-type nodes and various naive rates, M_3-type workers being in an average ($f_3 = 0.2$) proportion



(a) Error rates

(b) Redundancies

(c) Classification error rates and recalls

Fig. 14 Results obtained for a population structure with M_3-type nodes and various naive rates, M_3-type workers being in a large ($f_3 = 0.4$) proportion

a large number of very effective colluding saboteurs), the redundancy remains below an entire additional replication per work unit.

- the classification procedure is effective, spotting out most of the colluding workers with a low classification error rate.

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). Nevertheless, the actual findings in DGs make us think that such a scenario has a very low probability of occurring in practice. In fact, our selection of N was balanced between the good results it yields in most cases and the need to keep it within the realistic assumptions of a DG environment.

When presenting the sabotage models in section 2.2 we assumed that the malicious nodes are not aware of our sabotage detection mechanism and they act to conceal themselves from majority voting. Now, let suppose that some 'very' malicious M_2-type nodes are aware of our ST protocol and they will sabotage only when *all* nodes in the voting pools are also M_2-type ones. If *every* M_2-type node in the population is exhibiting such a behavior, these nodes will score no votes against and will be never discovered. This behavior happens to be at the limit of the behavior presented in the paragraph before, with M_2-type nodes sabotaging very infrequently - as the probability the master assigns three M_2-type nodes in a voting pool is very low (0.064 if $f_2 = 0.4$). Although, for the moment have no solution against this situation, we believe that this situation is very unlikely to occur and even if it would take place, the sabotage rate would remain low (around 10^{-3} as before).

We also think that colluding M_2-type nodes aware of our ST protocol and waiting for showing themselves up to the auditing phases of the protocol (steps 8 and 9 from algorithm 2) might not undermine the power of our algorithm. Suppose that M_2-type nodes do not sabotage in the initial round of voting. Therefore, conflicting voting pools will involve only M_1-type nodes which will be spotted out by the classification algorithm. In this case, the number of remaining suspect voting pools will be very low (almost zero) and steps 8 and 9 of the algorithm will almost be avoided. This case is similar with the one that only M_1-type nodes live in the population, besides the honest ones. More, if M_2-types nodes sabotaging in the initial phase are co-existing together with M_2-type nodes that sabotage only in the auditing phases, as the ST protocol selects for auditing honest nodes, it will be very unlikely that two or three such ST protocol-aware M_2-types nodes to be selected for auditing.

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 [15]. 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). If we want a very small classification error rate for this algorithm, the recall may be lower and conversely, for a large recall we should accept a larger classification error rate. The larger the classification recalls are, the lower will be the redundancy of the ST protocol and the number of errors in our ST protocol. On the other hand the lower the classification error is, the lower gets the ST protocol error rate. In any cases, 100% recall is not achievable by any possbile classification scheme.

Regarding the computational effort, the matrix multiplication algorithm is the most costly part. Kamvar *et al.* in [9] give a computational analysis for this cost. To compute the initial probability estimates $p_{v,i}$, our algorithm scans once the total voting pool results. To compute the joint distribution functions F_v, the algorithm performs for each worker at most N^2 multiplication and addition operations. To compute the matrix C, the algorithm performs a summation for each pairs of workers (quadratic complexity).

Thus, although the computation is somewhat heavy, the master has to perform only scalar operations with quadratic complexity and this can be run off-line.

Although our algorithm is an off-line one, we do not state that the master has to wait until the whole pool of client requests have been processed. The master simply has to set a proper value for N (like 30 in our experiments) and run the enhanced ST protocol once an average number of N results have been collected per worker. As we mentioned previously, bigger this number is, better performances are obtained. Practically, DG projects with average-to-short task length which usually employ $m = 2$ replication can afford values of N from 20 to 30, which is enough for providing a good protection against collusion. Such values of N are equivalent to running the enhanced ST protocol at every 1-2 weeks, for projects like the ones supported in Einstein@Home or World Community Grid.

4.3 Classification alternatives

At the heart of the heuristic presented in section 3 resides the initial classification procedure that identifies with a good reliability the naive M_1-type workers. We agreed upon the presented statistical approach after we tested several alternatives, offered by the machine learning field, presented in this section. First, we should note that we have an unsupervised learning task with the goal of identifying the naive saboteurs. As presented in section 3.2, nodes belonging to different types are characterized by different number of votes against collected during their life in the system. Thus, the number of votes against represents the essential information to profile a node.

Literature suggests to apply clustering algorithms like ROCK [8], applied to cluster the US Congressmen after their voting behavior during some period of time. If we consider each node represented as a categorical vector of votes per voting pools, our volunteer computing setup differs from the US Congress setup because a node votes only on very few work units from the total number of work units generated in the system. In the US Congress each congressman votes in the majority of open issues. Thus, the distance measure defined by [8] is not applicable in our case and we got very unstable results with this approach.

For clustering, machine learning suggests k-means [13] as a simple heuristic to unsupervised cluster individuals. We tested various representation of our nodes. First, we employed the $P \times P$ matrix of votes against, where P is the number of workers, each node having allocated a row in this matrix. Because this matrix is very sparse, the euclidean distance used by k-means does not have a strong-defined meaning. Thus, k-means succeeds to separate the naive workers only when they sabotage strongly, i.e. $s_1 > 0.5$, which is not acceptable for a desktop grid environment. If each node is represented by its distribution function F_{v_i}, we got somehow better results. But, in both cases, k-means does not guarantee the convergence. For cases with naive workers showing up very infrequently ($s_1 < 0.1$), the classification performance of k-means is very poor. We got the best of k-means for our problem if we represented each worker by the normalized values of the total number of votes against and the total number of voting pools where the node get defeated. In this case, k-means succeeds to extract out a good number of naive workers (high recall), but on the cost of big number of false positives (high error rate). Figure 15 shows the classification results, with k-means compared with our statistical approach for several population structures.

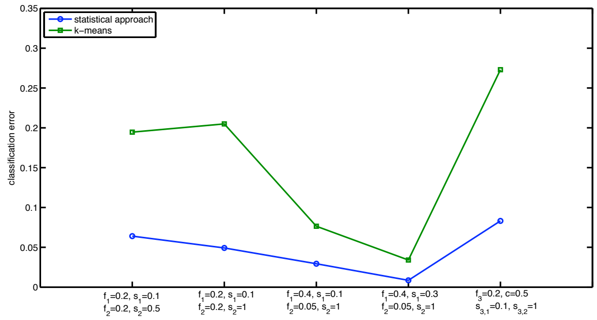


Fig. 15 Classification error rates for k-means clustering procedure against the statistical approach for various population structures

Retrieving a high number of false positive naive workers strongly influence the performance of the auditing procedure of algorithm 2. Thus, there is a higher possibility to select a malicious node to audit a suspect voting pool and less voting pools will be considered suspicious, as the number of nodes classified as honest is lower. Figure 16 shows global comparative results, plotting together the error rate of the ST protocol when performing the classification with k-means and with our statistical approach.

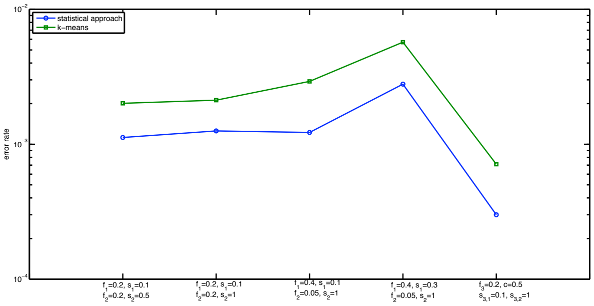


Fig. 16 Error rates of the ST protocol when using k-means clustering procedure against the statistical approach for various population structures

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 simplify 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 the nodes are not aware of the algorithm 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.

We believe that our procedure to spot out various malicious behaviors in voting environments can be extended to accommodate more than three voters in a voting pool. For example, e-commerce environments like E-Bay use opinions to create a profile of various actors: buyers and sellers. A scheme similar with the one described in this paper might be employed to spot out which responders cheat when submitting opinions. Another application would be quorum systems in Byzantine faulty environments, where replicated machines need to come to a consensus about various items like read/write operations. Research on this topic focuses on devising protocols to accommodate as many faulty machines as possible, but they do not tackle the problem of identifying those faulty machines that exhibit a correlated behavior over time [14]. Thus, we think that our approach is worth for consideration at a higher conceptual level and, with a proper generalization, might have a larger applicability in computer science.

Acknowledgements The authors are grateful to the constructive comments and useful feedback they received from all the anonymous reviewers of this paper. This work was funded by the European Commission IST FP6 Network-of-Excellence CoreGRID, project No.004265. G.C. Silaghi was also supported by the Romanian National Authority for Scientific Research under the project IDEI_573.

References

1. Anderson, D.P., McLeod, J.: Local scheduling for volunteer computing. In: 21th Intl. Parallel and Distributed Processing Symposium (IPDPS 2007), Proceedings, 2007, USA, pp. 1–8. IEEE Computer Society (2007)
2. Anderson, D.P.: BOINC: A system for public-resource computing and storage. In: 5th Intl Workshop on Grid Computing (GRID 2004), 2004, USA, Proceedings, pp. 4–10. IEEE Computer Society (2004)
3. Cappello, F., Djilali, S., Fedak, G., Hérault, T., Magniette, F., Néri, V., Lodygensky, O.: Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. Future Generation Comp. Syst. **21**(3), 417–437 (2005)
4. Costa, F., Silva, L., Fedak, G., Kelley, I.: Optimizing data distribution in desktop grid platforms. Parallel Processing Letters **18**(3), 391–410 (2008)
5. Domingues, P., Sousa, B., Silva, L.M.: Sabotage-tolerance and trust management in desktop grid computing. Future Generation Comp. Syst. **23**(7), 904–912 (2007)
6. Douceur, J.R.: The sybil attack. In: Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers, *Lecture Notes in Computer Science*, vol. 2429, pp. 251–260. Springer (2002)
7. Fedak, G., He, H., Cappello, F.: Bitdew: a programmable environment for large-scale data management and distribution. In: SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, pp. 1–12. IEEE Press (2008)
8. Guha, S., Rastogi, R., Shim, K.: Rock: A robust clustering algorithm for categorical attributes. Data Engineering, International Conference on (1999)
9. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: WWW '03: Proc. of the 12th intl. conf. on WWW, pp. 640–651. ACM Press (2003)
10. Karlin, S., Taylor, H.M.: A First Course in Stochastic Processes, Second Edition. Academic Press (1975)
11. Kim, J.S., Nam, B., Marsh, M.A., Keleher, P.J., Bhattacharjee, B., Richardson, D., Wellnitz, D., Sussman, A.: 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)
12. Kondo, D., Araujo, F., Malecot, P., Domingues, P., Silva, L.M., Fedak, G., Cappello, F.: Characterizing result errors in internet desktop grids. In: Euro-Par 2007, Parallel Processing, 13th Intl Euro-Par Conf, France, 2007, Proceedings, *LNCS*, vol. 4641, pp. 361–371. Springer (2007)
13. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: L.M.L. Cam, J. Neyman (eds.) Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297. University of California Press (1967)
14. Malkhi, D., Reiter, M.K.: Byzantine quorum systems. Distributed Computing **11**(4), 203–213 (1998)
15. Sarmenta, L.F.G.: Sabotage-tolerance mechanisms for volunteer computing systems. Future Generation Comp. Syst. **18**(4), 561–572 (2002)
16. Silaghi, G.C., Domingues, P., Araujo, F., Silva, L., Arenas, A.: Defeating colluding nodes in desktop grid computing platforms. Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on pp. 1–8 (2008)
17. Wei, B., Fedak, G., Cappello, F.: Collaborative data distribution with bittorrent for computational desktop grids. In: 4th Intl Symposium on Parallel and Distributed Computing (ISPDC 2005), 2005, France, pp. 250–257. IEEE Computer Society (2005)
18. Witten, I.E., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques (Second Edition). Morgan Kaufmann (2005)
19. Wong, S.: An authentication protocol in web-computing. In: 20th Intl. Parallel and Distributed Processing Symp. (IPDPS 2006), Proceedings, 2006, Greece. IEEE (2006)
20. Yurkewych, M., Levine, B.N., Rosenberg, A.L.: On the cost-ineffectiveness of redundancy in commercial p2p computing. In: 12th ACM Conference on Computer and Communications Security (CCS '05), Proceedings, pp. 280–288. ACM (2005)
21. Zhao, S., Lo, V.M., GauthierDickey, C.: Result verification and trust-based scheduling in peer-to-peer grids. In: 5th IEEE Intl Conf. on Peer-to-Peer Computing (P2P 2005), 2005, Germany, pp. 31–38. IEEE Computer Society (2005)