A Scheduling Algorithm for Defeating Collusion

Louis-Claude Canon, Emmanuel Jeannot, Jon Weissman

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Scheduling Algorithm for Defeating Collusion

Louis-Claude Canon — Emmanuel Jeannot — Jon Weissman

N° 7403

October 2010

Thème NUM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7403--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
BORDEAUX - SUD OUEST

A Scheduling Algorithm for Defeating Collusion

Louis-Claude Canon[*] , Emmanuel Jeannot[†] , Jon Weissman[‡]

Thème NUM — Systèmes numériques
Équipe-Projet runtime

Rapport de recherche n° 7403 — October 2010 — 27 pages

Abstract: By exploiting idle time on volunteer machines, desktop grids provide a way to execute large sets of tasks with negligible maintenance and low cost. Although desktop grids are attractive for cost-conscious projects, relying on external resources may compromise the correctness of application execution due to the well-known unreliability of nodes. In this paper, we consider the most challenging threat model: organized groups of cheaters that may collude to produce incorrect results.

By using a previously described on-line algorithm for detecting collusion and characterizing the participant behaviors, we propose a scheduling algorithm that tackles collusion. Using several real-life traces, we show that our approach minimizes redundancy while maximizing the number of correctly certified results.

Key-words: Volunteer computing systems, collusion, scheduling, result certification, BOINC

[*] Nancy University, LORIA and LaBRI – Nancy & Bordeaux, France
[†] INRIA and LaBRI – Bordeaux, France
[‡] Dept. of Computer Science and Engineering – University of Minnesota, Twin Cities – Minneapolis, USA

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération 33405 Talence Cedex
Téléphone : +33 5 40 00 69 00

Un algorithme d'ordonnancement pour déjouer la collusion

Résumé : Ce rapport présente un algorithme de placement de tâches dans les systèmes de calcul volontaires permettant de déjouer la collusion (attaque coordonnée des travailleurs volontaires). Il se base sur nos précédents travaux sur la détection de collusion. Les résultats expérimentaux utilisant des traces concrètes montrent que notre approche minimise la redondance et maximise le nombre de résultats certifiés correctement.

Mots-clés : Systèmes de calcul volontaires, collusion, ordonnancement, certification de résultats, BOINC

1 Introduction

Volunteer platforms such as desktop grids remain an attractive environment for many science and engineering applications [1, 2, 3, 4] due to their performance scaling and low cost. Recent work has shown that volunteer systems are still an order-of-magnitude cheaper than current clouds like Amazon EC-2 [5] even if volunteers are compensated for their cycles at cost. Desktop grids, however, present challenges to application deployment due to inherent volatility and dispersion of the platform: node and network failure, churn, erroneous and malicious behavior, and lack of central management. Much research has focused on how to tame this volatility under a set of assumptions. The most common of which is that failures are uncorrelated. This would account for many failure modes such as a slow or failed network link, node churn, independent node failure, or random byzantine failure due to a specific configuration problem at a node. If a result cannot be certified as correct in isolation, then collective certification techniques are needed. These techniques rely on two things: that the result space for a task is extremely large and that failures are unrelated. Such techniques include the use of precomputed answers (or quizzes) [6] or voting coupled with reputation systems [7] to "tease out" worker behavior. Replication and voting can be effective if the answer space is extremely large making the probability that two uncorrelated errors produce the same incorrect result negligibly small. With the use of reputation systems, such replication overhead can be optimized [7]. However, Internet-scale systems contain examples of correlated misbehavior and errors including botnet attacks, viruses, worms, sybil attacks, and buggy software distributions, to name a few. We use the broad term "collusion" to refer to the presence of correlated errors, either malicious or inadvertent. With correlated errors, simple majority voting with small amounts of replication may be ineffective, as colluders may contribute the same wrong answer in the majority. Therefore, new techniques are needed.

In earlier work [8], we showed how the notion of worker reputation can be extended in the context of collusive behavior. We developed techniques to identify groups of workers that tend to agree on outputs, whether colluding or not, and the likelihood that collusion may occur across groups. In this paper, we present a new collusion-resistent algorithm for on-line task scheduling and result certification that works hand-in-hand with the computation of agreement probabilities. We also present analytic results that form the basis for this new algorithm and show its on-line feasibility. A novel feature of this algorithm is that it both offers collusion avoidance (by temporally staggering same task allocation to prevent worker synchronization under malicious colluding) and does not require all task results to be known for the certification and update of probabilities. We then empirically evaluate this new algorithm using a set of real desktop grid traces under a wide variety of collusion scenarios against the standard BOINC replication algorithm that does not account for the possiblity of correlated errors. The results show that for a set of collusion scenarios, not only does our algorithm achieve significantly higher result certification accuracy or precision, and in most cases, also exhibits smaller overhead in terms of replication than BOINC.

2 Related work

We divide related work into correlated error scenarios, reputation systems, collusion characterization, and collusion avoidance and scheduling. The best known studies of real observed correlated errors can be found in the network literature including Sybil attacks [9], and worm propagation [10]. The problem of isolated errors has been studied in BOINC [1] which provides static replication and majority consensus. To improve performance, numerous reputation systems have been proposed to learn and characterize the statistical behavior of workers to make better scheduling decisions. Works in this area include both first-hand estimation techniques techniques such as smart and adaptive replication [7] [11], the use of quiz tasks and replication [6], the use of quizzes coupled with backtracking and voting [12], and second-hand approaches such as [13] and [14]. The problem with quizzes is to ensure that pre-computed quiz tasks cannot be detected by malicious hosts. However, none of these approaches are designed to handle general collusive behavior.

In the realm of collusion characterization, prior works include group-based agreement techniques [8], the use of checkpoints (trickle messages) for incremental checking [15], game theoretic approaches [16], graph clustering techniques [17], and detection of errors in dependent task chains [18]. These papers do not address the task allocation problem in the presence of collusion, however. Prior work in scheduling in the presence of collusion includes [19] which uses EigenTrust [13] and blacklisting. This work is limited as it requires all results be computed upfront and thus is off-line. Our approach both characterizes collusion and schedules tasks in an on-line manner based on the incremental generation of results as in actual volunteer systems (e.g. BOINC). A property of our approach is that as more results are generated, scheduling performance improves.

3 Models and Definitions

3.1 Application and Platform

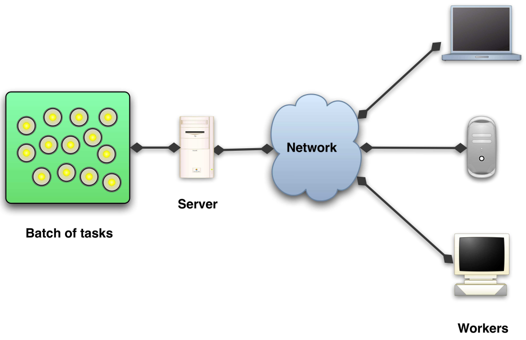

Figure 1: A Desktop Grid.

We propose the following model of a desktop grid (see Figure 1), directly inspired from BOINC [1]:

- We are given a batch of *jobs* to be executed on the platform. As we are interested in asymptotic behavior of the system, we suppose that the number of jobs is unbounded.
- We have a pool W of *workers*. Each worker $w \in W$ is able to compute all jobs in the batch. However, as workers may come and go they are not always available. This availability is defined by a set of intervals. If a worker, computing a job, leaves the system, it resumes job execution when it comes back.
- The *server* assigns each job to a set of workers and gathers the job results. For each job, we assume that there is only one correct result. Moreover, the result space is sufficiently large such that if two workers return the same incorrect result this means that they have *colluded*. Last, we assume that, like in BOINC, job assignment is pull-based: workers connect to the server and ask for a job. However, which job is assigned to given worker is left to the server and depends on its internal policy (*e.g*, redundancy, quorum) and algorithm. As some workers may take a long amount time to compute some jobs, the server assigns a timeout to each job sent to a worker. When the timeout is reached, the computation is cancelled on the given worker and the server may send the job to another worker.

3.2 Threat Model

Collusion is defined as the cooperation of multiple workers in order to send the same incorrect result. We distinguish between two types of collusion. The first is when the saboteurs voluntarily cooperate to send more than one incorrect result trying to defeat the quorum algorithm. The other case is where workers do not deliberately collude as in the case of a virus or a bug in the code executed by the workers. Moreover it is possible that a worker (a colluder or not) may simply *fail* to correctly execute the job. To model these possibilities, we consider three worker behaviors:

- a worker may *fail* independently of others and return an incorrect result. The failure probability is fixed.
- a worker may belong to the *non-colluding group*. Such a *non colluding worker* never colludes with another worker but may fail.
- a worker may belong to one or more *colluding groups*. In order to reduce the chance of being detected, members of a group act sometimes as colluders (returning the same incorrect result) and sometimes as non-colluding workers (returning the same correct result). The probability that a group decides to collude or not is fixed. Moreover, it is possible that two different groups of colluders collude together, which is called inter-collusion (*i.e.*, two workers from two different groups may send the same incorrect result with a given probability). A worker in a colluding group may also fail independently (failures are predominant over collusion), and it returns an incorrect result alone. We assume that non-colluding workers form a majority (at least relatively, if there are several colluding groups).

The set of groups (colluding and non-colluding) is denoted by G.

In general, to be efficient and not detected, colluders need to synchronize among themselves. Indeed, to stay undetected, one member of a colluder group that sends an incorrect result must ensure that other members of the same group (1) are assigned the same job and (2) send the same incorrect result. In this work, we assume workers can communicate out-of-band and once a group decides to collude, every member of this group sends the same incorrect result even if this job is mapped to these workers during non overlapping time-frames.

Last, we want to emphasize that the proposed threat model is strong: a worker may belong to one or more groups, groups can cooperate, colluders may sometime send a correct result to stay undetected, colluders are not required to compute the job at the same time in order to send the same incorrect result, and none of this information is known a-priori by the server.

3.3 Independence Condition

In general, it is not possible to compute the probability that a group of workers collude given that another group is colluding. To make this computation feasible, we assume the following natural independence condition. Let H_a be the event occurring when the workers in the set a return an identical incorrect result that is different from the results returned by any worker that does not belong to a. Moreover, let $\mathcal{U}(G)$ represent all possible unions of the groups in G. Let a and b be two unions of groups. If the intersection between a and b is empty, then the events H_a and H_b are independent. Otherwise, they are disjoint (except if $a = b$, in which case they are equal). Formally:

$$\forall (a,b) \in \mathcal{U}(G)^2 \begin{cases} H_a = H_b & a = b \\ H_a \perp H_b & a \cap b = \emptyset \\ H_a \cap H_b = \emptyset & \text{otherwise} \end{cases}$$

A consequence of this assertion is that collusion in a given colluding group occurs independently of the collusion of any other colluding group. Also, inter-collusion between two colluding groups is independent of the inter-collusion of two distinct colluding groups.

3.4 Metrics

The performance of a scheduling system for desktop grids can quantified according to two main criteria. The first concerns the efficiency of the platform usage. Small degrees of replication lead to greater efficiency or throughput. The second main criteria is related to the quality of the generated outputs. When the system certifies a result, it claims that it is correct for the corresponding job. However, due to the adversity present in the environment, it can erroneously certify some results. We want to maximize the number of correctly certified results.

For assessing these criteria, we propose to measure the *overhead* and the *precision* of a system. The overhead is defined as the average duplication ratio, *i.e.*, the expected number of times each job is computed on a distinct worker. For example, a system that sends any job only once has a unitary overhead. The precision is the ratio of results correctly certified over the total number of certified results. The precision must be the closest possible to one.

3.5 Problem Definition

The input of the problem is a desktop grid (set of workers and jobs), and an inaccuracy parameter ϵ. This parameter is given by the user. A result is certified when the estimated probability of being correct is greater than $1-\epsilon$. Workers can cheat or fail according to the threat model described above. At the beginning of an exeecution, we assume that we have no a-priori knowledge about the workers.

The problem we tackle focuses on allocating jobs to workers and returning certified results for each of these jobs. The goal is to maximize the precision and minimize the overhead.

4 The Limit of the Quorum Strategy in the presence of Collusion

To motivate this work, we present a preliminary experiment to study the precision of the quorum strategy (use by BOINC) in the presence of collusion. In the quorum-based approach, a result is certified (assumed to be correct) when a given number of workers have returned the same result. A job is submitted to workers as long as the quorum is not reached. In this experiment, the settings are the same as the ones used in the SETI@Home project: minimum duplication is 4, required quorum is 3, maximum duplication is 10.

We have 6 scenarios. Each with the same workload trace, availability trace and platform trace (docking@home and SETI@Home) as described in Section 8. The number of collusion groups (with inter-collusion between distinct groups of collusion) increases from 0 to 5 by an unitary increment. Honest workers always form a majority (absolute for the first two, relative for the others: 100%, 60%, 50%, 40%, 40%, 25%). The simulations cover 500,000 jobs and there are 2,000 workers. Last, the probability of collusion is varying for each case.

Results show that the precision (percentage of results correctly certified, Fig. 2), can be as low as 70% when using only the quorum in case of large adversity. Hence, a solution based only on quorum is largely insufficient when facing organized cheaters.

The goal of the remainder of this paper is to propose a solution to overcome this problem.

5 Overview of the Architecture

The architectural framework is based on three components (see Figure 3):

- The *characterization system* takes as input the observed behavior of the workers. For a given job, it updates the worker profiles when they send the same result. In some cases, it may also need to know which result is the correct one. Based on these observations, the characterization system clusters workers and estimates the composition of the colluding groups (if any) and the non-colluding group (always assuming that it is the largest one). Sufficient replication is performed to guarantee this within a threshold. On the quantitative side, it estimates the probability that a given subset of workers have colluded and returned an incorrect result. To account for the imprecision of the system, the characterization component

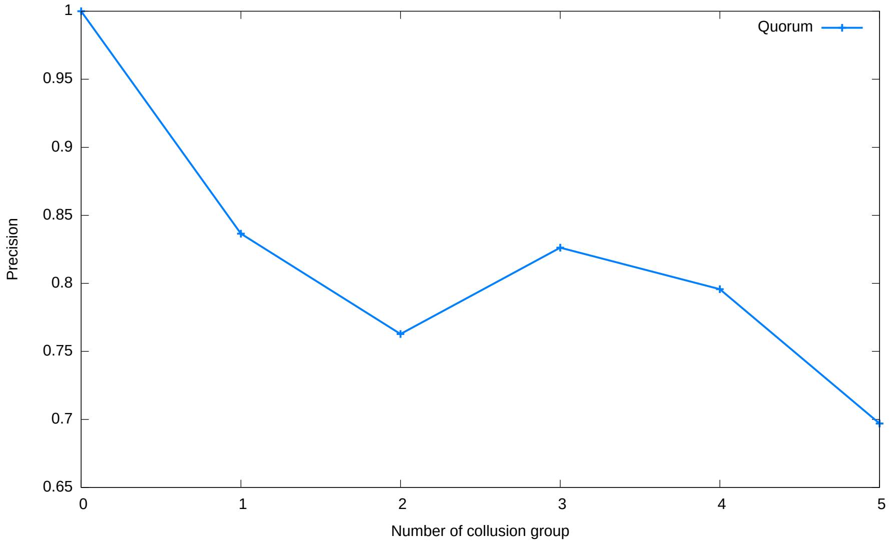

Figure 2: Precision of the Quorum-based approached when varying the number of collusion group

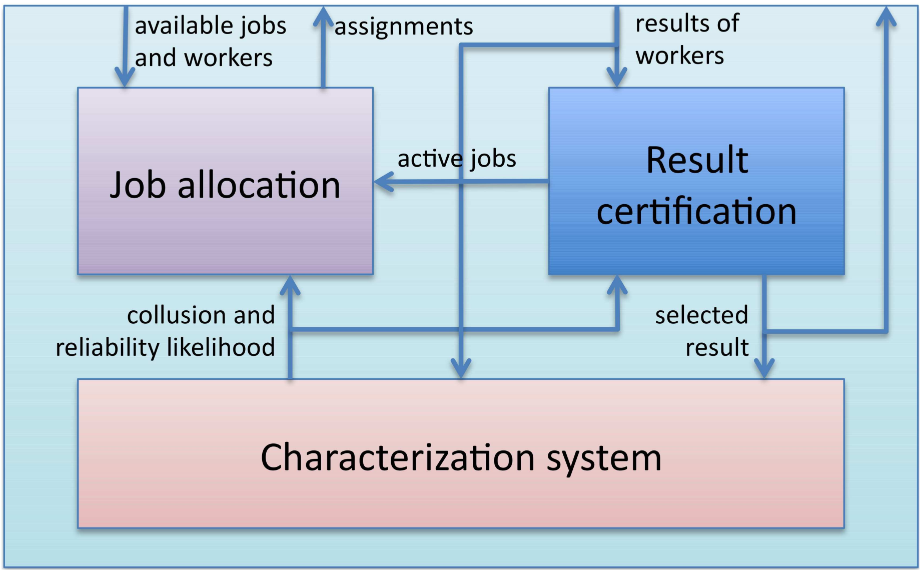

Figure 3: Architecture of the proposed solution

must quantify the inaccuracy of the output result. To do so, the characterization system computes the probability of collusion as a *random variable* determining the average probability and its variance. Hence, the larger the variance, the more inaccurate the expected probability. We use both values (mean and variance) for the result certification as shown in Section 7.

In our previous work [8], we have proposed a characterization system and assessed its performance. We have proposed two algorithms, one based on collusion (correlated bad behavior) representation, and one based on agreement (general agreement between nodes whether good or bad). Both approaches are incremental: they start by putting each worker in different collusion groups and, based on observed behaviour, cluster workers together. Hence, as shown in the previous work, the precision of the system improves with time as more and more observations are made. In this paper, we use the agreement representation because it is a simpler mechanism than the collusion representation and it does not need to know the correct result to output the requested probability and variance (see [8] for the details).

- The *result certification* component is used to decide among all the (possibly different) results returned by the workers for a given job which one is the correct one. The estimated correct result is called the *certified* result. More precisely, based on returned results and information given by the characterization system, it estimates the probability that a given result is correct. How this probability is estimated is described in Section 6. When no result can be certified (because the confidence is too low) the job state is set to be *active* and new results are computed until the confidence threshold is reached.

- The *job allocation* component decides which job is allocated to a given worker. How this mapping is computed is another contribution of this paper and is described in Section 7.

6 Result Correctness Probability

Upon job completion, a worker returns a result to the server. All the results that correspond to the same job are then considered for certification. If the correct result is present, the objective is to certify it. Otherwise, no result should be certified.

Our procedure is to compute the probability that each result is correct given: the results that were received, the workers that computed them and the estimated groups and probabilities of collusion. Given these values, the result with the probability the closest to 1 is selected. However, if no probability is close to 1 (*i.e.*, larger than a given threshold), then no result is certified and the job is resubmitted to another worker.

For a given job, the workers are clustered according to their returned result: all workers that return exactly the same result for this job are put in the same *team*. We denote P the set of teams for a specific job (each worker $w \in W$ is not necessarily present in any of the teams as a job is computed by a subset of workers only). The event E_a occurs when all the workers in set a collude to return the same incorrect result. The event $\overline{E_a}$ occurs when the workers in set a return the correct result. A specific configuration P of teams is denoted as F_P. Therefore, the probability that the result returned by team i is correct given the configuration of team P is $\Pr[\overline{E_i}|F_P]$.

The computation of this value requires the exact value of the probability that any subset of workers collude (*i.e.*, $\forall a \subseteq W, \Pr[E_a]$) as well as the decomposition of workers into groups (*i.e.* G). However, the characterization system provides only an estimation of these values. The derivation that follows assumes that these probabilities are exact. Additionally, we need the exact composition of the groups (colluding and non-colluding). Again, we assume that the estimation given by the characterization system is accurate. The impact of this approximation is assessed in Section 8.

Using the independence condition described in Section 3.3, it is possible to compute the exact value of $\Pr[\overline{E_i}|F_P]$. However, this evaluation is exponential. Hence, we propose two approximations. The first is always performed to ensure a tractable, yet precise evaluation. The second is used when the number of teams reaches a given limit in order to speed-up computation. These approximations come in addition to the estimations of any probability $\Pr[E_a]$.

We describe the initial derivation of $\Pr[\overline{E_i}|F_P]$ in Lemma 1. It leads to a reformulation that depends only on an event $I_{Q,P}$ that defines an intersection of distinct collusion events. The probability of this event is further bounded in Lemma 2 (the first approximation is then made). To allow for an effective computation, Lemma 3 describes how an union of collusion events can be calculated, using the second approximation when necessary. See Table 1 for a description of our notation.

Let $I_{Q\,P}$ be the event that occurs when each of the teams in Q colludes but no inter-collusion occurs between any pair of teams in P. Let $h(Q)$ be the groups having at least one worker in one of the teams of Q.

$$I_{Q\,P} = \bigcap_{i \in Q} \left(E_i \cap \bigcap_{g \in h(P \setminus \{i\})} \overline{E_{i \cup g}} \right)$$

Lemma 1. *The probability that a given team i gives the correct result given a specific configuration P of teams is given by:*

$$\Pr[\overline{E_i}|F_P] = \frac{\Pr[I_{P\setminus\{i\}\,P}] - \Pr[I_{P\,P}]}{\Pr[I_{P\,P}] + \sum_{i\in P}\left(\Pr[I_{P\setminus\{i\}\,P}] - \Pr[I_{P\,P}]\right)}$$

Proof. We use the definition of the conditional probability to derive the probability $\Pr[\overline{E_i}|F_P]$:

$$\Pr[\overline{E_i}|F_P] = \frac{\Pr[F_P \cap \overline{E_i}]}{\Pr[F_P]}$$

By definition of F_P:

$$\begin{aligned} F_P &= F_P \cap \left(\bigcap_{i\in P} E_i \cup \bigcup_{i \in P} \overline{E_i} \right) \\ &= F_P \cap \bigcap_{i \in P} E_i \cup \bigcup_{i\in P} F_P \cap \overline{E_i} \end{aligned}$$

As these events are disjoints, the probability can be expressed as:

$$\Pr[F_P] = \Pr\left[F_P \cap \bigcap_{i\in P} E_i \right] + \sum_{i \in P} \Pr\left[F_P \cap \overline{E_i}\right]$$

W	Set of workers
w	a worker $w \in W$
ϵ	inaccuracy parameter (user given)
a and b	Subsets of workers: $(a, b) \subseteq W^2$
G	Groups of workers: $G \subset \mathcal{P}(W)$ (G is a partition of W)
g	Group of workers: $g \in G$
$\mathcal{U}(G)$	All the possible union of the groups in G: $\mathcal{U}(G) = \bigcup_{U \in \mathcal{P}(G)} \left\{ \bigcup_{g \in U} g \right\}$
P	Partition of the workers according to the results
Q	Sub-partition of the workers: $Q \subseteq P$
$h(Q)$	Groups having at least one worker in one of the teams of Q: $h(a) \subseteq G$
i and j	Team of workers (they return the same result) : $(i, j) \in P^2$
$\dot{h}(i)$	The workers belonging to the groups $h(\{i\})$: $\dot{h}(i) = \bigcup_{g \in h(\{i\})} g$ ($\dot{h}(i) \subseteq W$))
E_a	Event “all the workers in the set a return an identical incorrect result for a given job”
H_a	Event “all the workers in set a return an identical incorrect result that is different from the results returned by any worker that does not belong to a”: $H_a = E_a \cap \bigcap_{g \in G \setminus h(\{a\})} \overline{E_{a \cup g}}$
F_P	Event “the workers produces results such that they correspond to the specific configuration P of teams”
$I_{Q\,P}$	Event "all the teams in Q collude and no inter-collusion occurs between any pair of teams in P"

Table 1: List of notations

Each of these events can be reformulated:

$$F_P \cap \bigcap_{i \in P} E_i = \bigcap_{i \in P} \left(E_i \cap \bigcap_{g \in h(P \setminus \{i\})} \overline{E_{i \cup g}} \right)$$

$$F_P \cap \overline{E_i} = \overline{E_i} \cap \bigcap_{j \in P \setminus \{i\}} \left(E_j \cap \bigcap_{g \in h(P \setminus \{j\})} \overline{E_{j \cup g}} \right)$$

To compute the last term, we proceed as follows:

$$\begin{aligned}
\Pr[F_P \cap \overline{E_i}] &= \Pr \left[\overline{E_i} \cap \bigcap_{j \in P \setminus \{i\}} \left(E_j \cap \bigcap_{g \in h(P \setminus \{j\})} \overline{E_{j \cup g}} \right) \right] \\
&= \Pr \left[\bigcap_{j \in P \setminus \{i\}} \left(E_j \cap \bigcap_{g \in h(P \setminus \{j\})} \overline{E_{j \cup g}} \right) \right] \\
&\quad - \Pr \left[\bigcap_{j \in P} \left(E_j \cap \bigcap_{g \in h(P \setminus \{j\})} \overline{E_{j \cup g}} \right) \right]
\end{aligned}$$

Moreover, by definition of $I_{Q\,P}$ we have,

$$I_{Q\,P} = \bigcap_{i \in Q} \left(E_i \cap \bigcap_{g \in h(P \setminus \{i\})} \overline{E_{i \cup g}} \right)$$

Hence, by identifying all the terms, we obtain the result. □

The following lemma proposes a bound of $\Pr\left[I_{Q\,P}\right]$ that is used to compute an approximation of this term.

Lemma 2. *For a given configuration P of teams and a subset of teams $Q \subseteq P$, we have:* $\Pr\left[I_{Q\,P}\right] \leq \prod_{i \in Q} \left(\Pr[E_i] - \Pr\left[\bigcup_{g \in h(P \setminus \{i\})} E_{i \cup g} \right] \right)$

Proof. Let $\dot{h}(i)$ denote the workers belonging to the groups $h(\{i\})$.

The main result is given by the following derivation:

$$\begin{aligned}
\Pr\left[I_{Q\,P}\right] &= \Pr \left[\bigcap_{i \in Q} \left(E_i \cap \bigcap_{g \in h(P \setminus \{i\})} \overline{E_{i \cup g}} \right) \right] \\
&\leq \prod_{i \in Q} \Pr \left[E_i \cap \bigcap_{g \in h(P \setminus \{i\})} \overline{E_{i \cup g}} \right] \\
&\leq \prod_{i \in Q} \left(\Pr[E_i] - \Pr \left[E_i \cap \bigcup_{g \in h(P \setminus \{i\})} E_{i \cup g} \right] \right) \\
&\leq \prod_{i \in Q} \left(\Pr[E_i] - \Pr \left[\bigcup_{g \in h(P \setminus \{i\})} E_{i \cup g} \right] \right)
\end{aligned}$$

At the second line, we need to prove that:

$$\Pr\left[E_i \cap \bigcap_{g\in h(P\backslash\{i\})} \overline{E_{i\cup g}} \,\middle|\, E_{i'} \cap \bigcap_{g\in h(P\backslash\{i\})} \overline{E_{i'\cup j}}\right] \\ \leq \Pr\left[E_i \cap \bigcap_{g\in h(P\backslash\{i\})} \overline{E_{i\cup g}}\right]$$

By definition, we can formulate:

$$E_i \cap \bigcap_{g\in h(P\backslash\{i\})} \overline{E_{i\cup j}} = \bigcup_{a\in\mathcal{U}(G\backslash h(P))} H_{\dot{h}(i)\cup a}$$

As the events in the union (right term) are disjoints, we need to show that for all $(i, i' \neq i) \in P^2$ and for all $(a, a') \in \mathcal{U}(G\backslash h(P))^2$:

$$\Pr[H_{\dot{h}(i)\cup a} | H_{\dot{h}(i')\cup a'}] \leq \Pr[H_{\dot{h}(i)\cup a}]$$

If $(\dot{h}(i) \cup a) \cap (\dot{h}(i') \cup a') = \emptyset$, then the independence conditions ensure the equality. Otherwise, $\Pr[H_{\dot{h}(i)\cup a} | H_{\dot{h}(i')\cup a'}] = 0$. □

When the events $E_i \cap \bigcap_{g\in h(P\backslash\{i\})} \overline{E_{i\cup g}}$ are independent, from Lemma 2 we have: $\Pr\left[I_{Q\,P}\right] = \prod_{i\in Q}\left(\Pr[E_i] - \Pr\left[\bigcup_{g\in h(P\backslash\{i\})} E_{i\cup g}\right]\right)$ With a small number of colluding groups or inter-colluding groups, this assumption is valid and the computation is exact. Hence, we use the bound of Lemma 2 as an approximation of the probability.

The following lemma allows the computation of a union of events $E_{i\cup j}$.

Lemma 3. *For all sets of groups Q and for all teams $i \in P$,*

$$\Pr\left[\bigcup_{g\in Q} E_{i\cup g}\right] = \begin{cases} 0 & \text{if } Q = \emptyset \\ \Pr[E_{i\cup g}] + \Pr\left[\bigcup_{g'\in Q\backslash\{g\}} E_{i\cup g}\right] & \\ - \Pr\left[\bigcup_{g'\in Q\backslash\{g\}} E_{i\cup g\cup g'}\right] & \text{otherwise} \end{cases}$$

Proof. We use the following basic result to handle the combinatoric:

$$E_{a\cup b\cup c} = E_{a\cup b} \cap E_{a\cup c}$$

By considering that $Q = \{g\} \cup Q\backslash\{g\}$, by adding each term of the union and by removing the intersection (the intersection $E_{i\cup g} \cap \bigcup_{g'\in Q\backslash\{g\}} E_{i\cup g'}$ can be simplified as $\bigcup_{g'\in Q\backslash\{g\}} (E_{i\cup g} \cap E_{i\cup g'})$ and much further as $\bigcup_{g'\in Q\backslash\{g\}} E_{i\cup g\cup g'}$), we have:

$$\Pr\left[\bigcup_{g\in Q} E_{i\cup g}\right] = \Pr[E_{i\cup g}] + \Pr\left[\bigcup_{g'\in Q\backslash\{g\}} E_{i\cup g'}\right] \\ - \Pr\left[\bigcup_{g'\in Q\backslash\{g\}} E_{i\cup g\cup g'}\right]$$

□

As the evaluation of this union is exponential in the number of teams, we can realize a second order approximation when the number of groups in Q is above some limit. In this case, we discard the intersection term (third term).

As shown in section 5, the characterization system is assumed to provide the probability of collusion of any subset of workers, this probability being computed as a random variable. Hence, for any set of workers a, the characterization system outputs the average of $\Pr[E_a]$ and its variance. Based on the above lemmas, we can compute an approximation of the probability that a result is correct assuming that the expected probabilities output by the characterization system are close to exact. A question that arises is how does the variance propagate? In Lemma 3, for example, only addition is used. In this case, the variance of a sum of random variables is the sum of the variances of the random variables. A similar rule exists for multiplication between random variables (in Lemma 2). However, there is no general rule for division (in Lemma 1). In this case, the mean and the standard deviation of the numerator are both divided by the mean of the denominator (its variance is then discarded).

7 Job Allocation and Result Certification

We now describe how jobs are allocated to workers. The algorithm is shown in Alg. 1. Each job can have 4 states: *available* when it has never been assigned to a worker (A_v represents the set of available jobs); *processing* when one worker is computing this job (P represents the set of jobs in the processing state); *active* when some results have been received but no result has yet been certified and no worker is computing it (A_c represents the set of active jobs); *terminated* when a result has been certified for this job (T represents the set of terminated jobs).

Workers ask the server for jobs (pull-based mechanism). The server assigns to each worker one active job or, if no job is active, an available job. Also, each assigned job to a given worker is different (no job is computed twice by the same worker).

The job is then processed by the worker and a result is returned to the server. The characterization system is informed of the result that has been computed by this worker and for this job. This information may be used in order to improve the estimation of the group compositions and their collusion probabilities.

We also cluster workers into teams (set of workers that have computed the same results of this job). As the proposed formulas in Section 6 do not take into account individual failures, a team is created if at least two workers have returned the same result (as the result space is large, these workers do not return the same erroneous result by happenstance). A consequence of discarding single results is that any certified result must have been computed at least twice. The minimum overhead of our method can achieve is therefore 2.

Based on these estimations (structure and probability) of the teams, the probability p and the variance v of this probability (as it is a random variable) is computed by the characterization system (see Section 6). The `certify` function returns the result r for which $p - \sqrt{v}$ is the greatest. We use the inaccuracy parameter ϵ, given by the user, to certify the result. If this value is greater than the threshold $1 - \epsilon$ (typically 0.95), the result r is certified: we assigned this result to j and the state of j becomes terminated. As the probability p is difficult to estimate, its value may be inconsistent in some cases (greater than one with

Algorithm 1: Job allocation and result certification

```
Input: a desktop grid with a set of jobs and $\epsilon$ the inaccuracy parameter.
move all jobs to $A_v$ // All jobs are set as available
$A_c \leftarrow \emptyset$ // No job is active
$P \leftarrow \emptyset$ // No job is being processed
$T \leftarrow \emptyset$ // No job is terminated
while Worker $w$ is contacting the server do
    if $w$ pulls a job then // The worker is asking for a job
        if $A_c \neq \emptyset$ then // look for an active job
            select $j$ in $A_c$ such that $w$ has never computed $j$
        else // select an available job
            select $j$ in $A_v$
        assign $j$ to $w$
        move $j$ to $P$
    if $w$ returns a result for job $j$ then // The worker is returning a
    result
        update estimation of collusion between workers
        update teams of workers for $j$
        $w$ asks for a new job
        $(p, v, r) \leftarrow$certify($j$)
        if $p - \sqrt{v} > 1 - \epsilon$ then
            move $j$ to $T$ // $j$ is terminated
            set $r$ to $j$ // result $r$ is associated to job $j$
        else
            move $j$ to $A_c$ // $j$ is still active and requires at
            least on more result to be certified
```

a large variance). This happens when a sub-result is divided by a value close to zero. To prevent this situation, we add another criterion: variance v must be greater than some threshold (in practice, a large value like 0.1 is sufficiently discriminative). If $p - \sqrt{v}$ is under the threshold then no result is certified and the state of the job becomes active (it will be assigned to a new worker later).

An important consequence of the job assignment mechanism is that a job is never executed by more than one worker at a time. This prevents colluders to decide, out-of-band, to collude when they know that the same job has been submitted to several members of the colluding group. With our mechanism, a collusion decision must be made when a job is submitted to a worker with the risk that only this worker of the group is assigned to this job and hence, that it is alone to send an incorrect result. If a colluder waits to see if his/her cohorts have the same task, its job will time-out.

Last, jobs are randomly selected from the active pool. We could use either FIFO or a more elaborate strategy that considers which worker is asking for a job. In this work, we have implemented only the random policy because in a given simulation, we have observed that there was at most one active job in more than 90% of the cases and, in 8% of the cases, only two active jobs. Moreover, as the jobs from the active pool are always chosen with priority vs.

jobs from the available pool, this ensures that a job entering the active pool will necessarily becomes terminated provided that each execution is timed-out. Therefore, with Algorithm 1, if we consider the time during which a job stays active without being processed, these strategies are equivalent.

8 Empirical Validation

We compare our method called CAA (Collusion Aware Algorithm) with the same quorum-based algorithm used in BOINC. Namely, each job is first assigned to 4 workers. Whenever a quorum of 3 is achieved for a given job (3 workers agreeing on an identical result), no more workers are assigned to this job. Moreover, a job cannot be assigned to more than 10 workers. These settings corresponds to SETI@Home.

For all the simulations, we have fix $\epsilon = 0.95$ for our approach.

8.1 Settings

For assessing the performance of our method, we simulate a platform based on a several traces from the *Failure Trace Archive* (FTA [21]) that correspond to real parallel and distributed platforms. Specifically, the SETI@Home traces provide availability periods of a set of machines over two years. Moreover, these traces gives the speed of each machine in number of floating operations per second. As these traces concerns more than 220,000 workers, we limit the number of machines that are considered. By default, we use 2,000 workers.

The simulated workload is also based on a trace. Michela Taufer provided us a workload trace of the docking@home volunteer project. Each job is characterized by a estimated number of floating operations to perform. The duration of the execution of one job on a given machine is then determined by dividing the cost by the speed. The provided workload contains about 150 thousands jobs. This workload is replicated until one million jobs have been computed. A simulation ends either due to this limit or because the end of the availability trace file is reached. In more than 90% of the performed simulations, at least 500,000 jobs are computed. As such, the overhead and the precision are measured on the first 500,000 jobs only. It allows a fair comparison of the methods.

Machines that undergo long periods of inactivity, that are slow or that disappear from the network may cause some jobs to stay in the system indefinitely. Therefore, a timeout of 10 days is associated to each execution.

We utilize a set of parameters that specifies the adversity of a platform relatively to our threat model. Every worker is supposed to return the correct result for any job by default, except if it fails or if it colludes. Failures are modeled as follows: a percentage of workers are completely reliable; unreliable workers return a correct result with the same reliability probability. For colluding groups, a set of worker participation percentages (*e.g.*, $k\%$) with corresponding collusion probabilities are given. According to these values, workers are distributed into either the non-colluding group or a colluding group. A worker that is in a colluding group returns a colluding result (incorrect) with the associated collusion probability. To specify each cooperation between colluding group (inter-collusion), a set of colluding groups is given with the inter-collusion probability. Several inter-collusion relations may be defined, however, the adversity scenario must be feasible (the aggregated collusion and inter-collusion probabilities must not exceed 1). For example, in the first simulations, we considered that the platform is fully reliable, that there is one colluding group. As there is only one colluding group, there is no possible inter-collusion. The participation percentage and collusion probability of this group is the parameter that changes to study its effect.

Finally, each adversity scenario that is considered is instantiated 10 times with a different seed. As our study leads to 39 different scenarios, it represents 390 distinct simulations for our method and the same quantity for comparing to the BOINC algorithm. These scenarios are summarized in Table 2 by distinguishing the tested parameter(s) from the default values for the other parameters.

Studied parameter(s)	Tested values	Default values
Participation and probability of collusion	(10%,100%), (15%,67%), (20%,50%), (30%,33%), (40%,25%)	2,000 workers, reliability of 100%, one colluding group
Probability of collusion	100%, 85%, 70%, 55%, 40%, 25%, 10%	2,000 workers, reliability of 100%, one colluding group
Number of colluding groups	0, 1, 2, 3, 4, 5, 6, 7, 8	2,000 workers, reliability of 100%, several colluding groups (200 workers in each), collusion probability of 25%
Number of colluding groups	1, 2, 4, 5, 8, 10, 20, 40	2,000 workers, reliability of 100%, several colluding groups (800 workers in total), collusion probability of 25%
Number of colluding groups in each inter-colluding group	6, 3, 2, 1	2,000 workers, reliability of 100%, 6 colluding groups (200 workers in each), several non-overlapping groups of inter-collusion, collusion probability of 10%, inter-collusion probability of 25%
Percentage of reliable workers and reliability probability of unreliable workers	100%, 85%, 70%, 55%, 40%, 25%, 10%	2,000 workers, 4 colluding groups (200 workers in each), 2 non-overlapping groups of inter-collusion, collusion probability of 10%, inter-collusion probability of 25%
Worker availability trace and number of workers	(Overnet, 1,000), (Microsoft, 1,500), (SETI@Home, 2,000)	Reliability of 85%, 4 colluding groups (200 workers in each), 2 non-overlapping groups of inter-collusion, collusion probability of 10%, inter-collusion probability of 25%

Table 2: Summary of the experimental parameters for each study

8.2 Results Analysis

Figures 4 to 7 depicts the precision and the overhead of the BOINC algorithm and our method. Each figure shows the effect of the variation of one parameter (or a combination of parameters) while the others are set to fixed values. The result are represented through the use of boxplots. In a boxplot, the bold line is the median, the box shows the quartiles, the bars show the whiskers (1.5 times the interquartile range from the box) and additional points are outliers. A line that links each median is added in order to ease the reading of the figures.

We first study the impact of the collusion parameters related to a single colluding group on the performance of the scheduling method. To this end, we fix the average number of incorrect generated results that are due to collusion. It involves to keep constant the product of the proportion of colluders and their probability to collude. In Figure 4, the extreme cases correspond to one colluding group with 10% of the workers that always collude and one colluding group with 40% of the workers that collude in 25% of the cases.

In term of precision, we observe that the BOINC algorithm is more resistant to small groups of collusion even though they collude more often than large ones. It is indeed more likely that a large group reaches a quorum (either with a correct or an incorrect result) than a smaller one. We can then conclude that the adversity is largely determined by the size of the colluding group rather than its probability to collude. On the other hand, the overhead of the BOINC algorithm is stable. Our method, however, is able to resist any kind of collusive group behavior while adapting the overhead to harder settings (when the colluding group is large). There is a slight decrease in the precision for the last group (the one with 40% of the workers). Overall, our method always dominates BOINC in these settings.

The next constant parameter is the size of a single colluding group, while the collusion probability is studied. On Figure 5, the collusion probability of a group with 40% of the workers varies from 100% to 10%.

With a high collusion probability, the BOINC algorithm performance is the worst (for both the precision and the overhead). Similarly, a high collusion probability involves a high overhead. In term of precision, however, our approach shows its worst performance when the collusion probability is low. Indeed, this makes the colluding groups harder to detect for the underlying characterization system. As the precision is perfect when the probability of collusion is zero, there is a special collusion probability value that minimizes the precision achieved by our method. Therefore, in a majority-based environment, a malicious peer should always collude in order to maximize their effect. In the presence of a mechanism specifically designed to defeat collusion, they should use a low collusion probability. However in this case, our approach still outperforms the BOINC algorithm for both objectives.

As we have empirically validated our approach with one colluding group, we now focus our study to the case of several colluding groups. We choose a collusion probability of 25% because it corresponds to a difficult setting (yet, not the hardest). Figure 6 depicts the performance obtained in presence of 0 to 8 colluding groups, each having 10% of the workers.

As expected, the performance of the BOINC algorithm decreases with the number of colluding groups. However, the worst precision, which is obtained with 8 colluding groups, is not bad (97%) given that there is only 20% honest

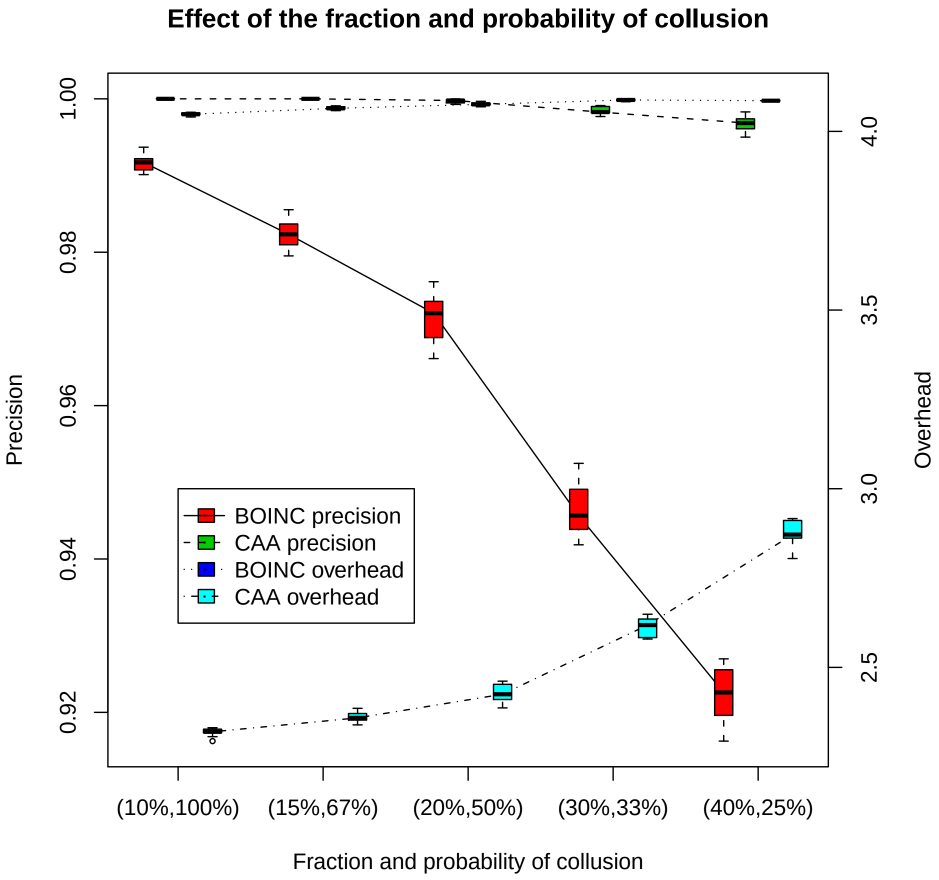

Figure 4: Performance of two methods with 2,000 workers, fully reliable plus one colluding group with varying participation and probability (10% of the results returned by workers to the server are incorrect).

workers. The same remark also applies to the overhead. We explain this by highlighting the chosen collusion probability. We have indeed specifically selected a low value in order to test the limits of our method by posing a greater challenge. The overhead of our method increases as the adversity becomes stronger. The precision achieved by our method remains stable except with 8 colluding groups. Overall, our method is almost always better except for the overhead with 7 and 8 groups.

In Figure 7, we see the effect of the number of colluding groups given that there is always 40% of colluders with a 25% probability of collusion. As expected, the BOINC precision increases with the number of colluding groups as the probability that colluders achieve a quorum decreases. The precision of our approach slightly decreases when the number of groups increases until the value 20. Therefore, colluders make their detection more difficult if they split into several groups. However, if they are too fragmented, they can no longer collude.

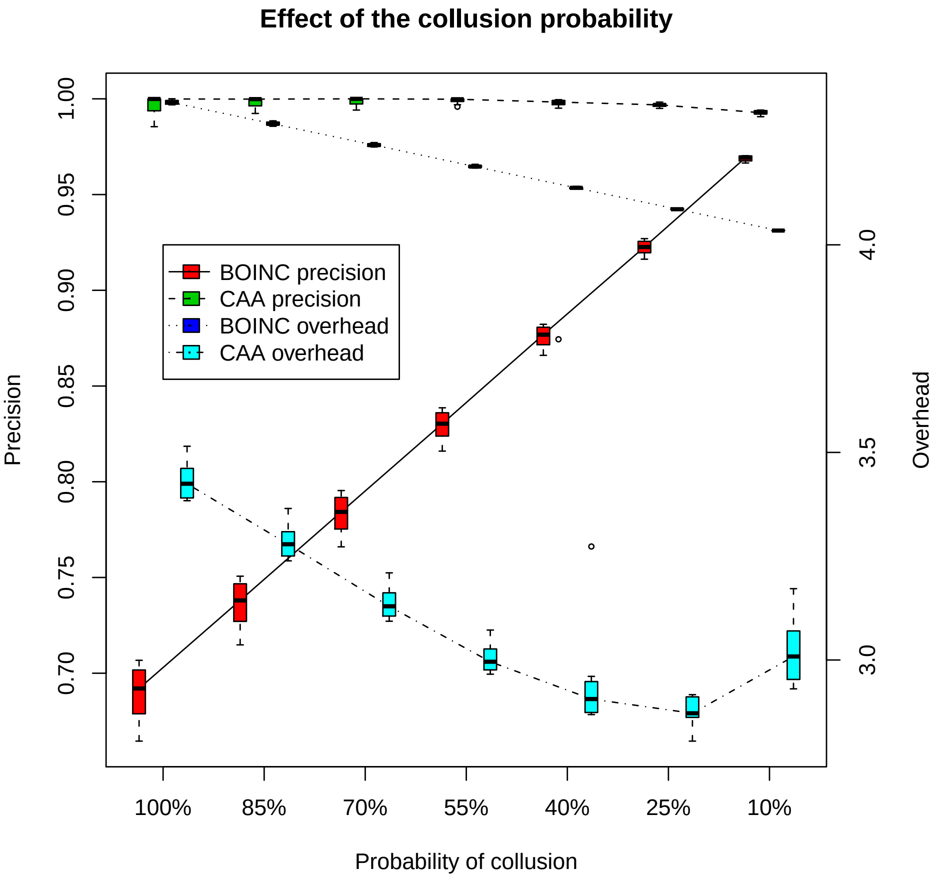

Figure 5: Performance of two methods with 2,000 workers, fully reliable plus one colluding group with 800 workers and varying collusion probability.

As for the collusion probability, there exists an optimal number of colluding groups for the colluders. On this study, our approach dominates the BOINC algorithm up to 5 colluding groups. For more groups, our approach is less precise but provides a far better overhead. This limit on the number of groups is related to the characterization system we used. It is indeed more difficult to detect smaller groups than larger ones.

Figure 8 shows the effect of inter-collusion. When the number of colluding groups in each inter-colluding group is one, there is no inter-collusion, *i.e.*, the colluding groups are not cooperating. When there are 2 colluding groups in each inter-colluding group, 3 inter-colluding groups are built. With 3 groups, there are 2 inter-colluding groups plus another one that comprises one colluding group from each inter-colluding group.

The precision of the BOINC algorithm increases as the number of colluding groups per inter-colluding group decreases. As there is less and less cooperation, majority is harder to achieve and the quorum-based mechanism fails less often.

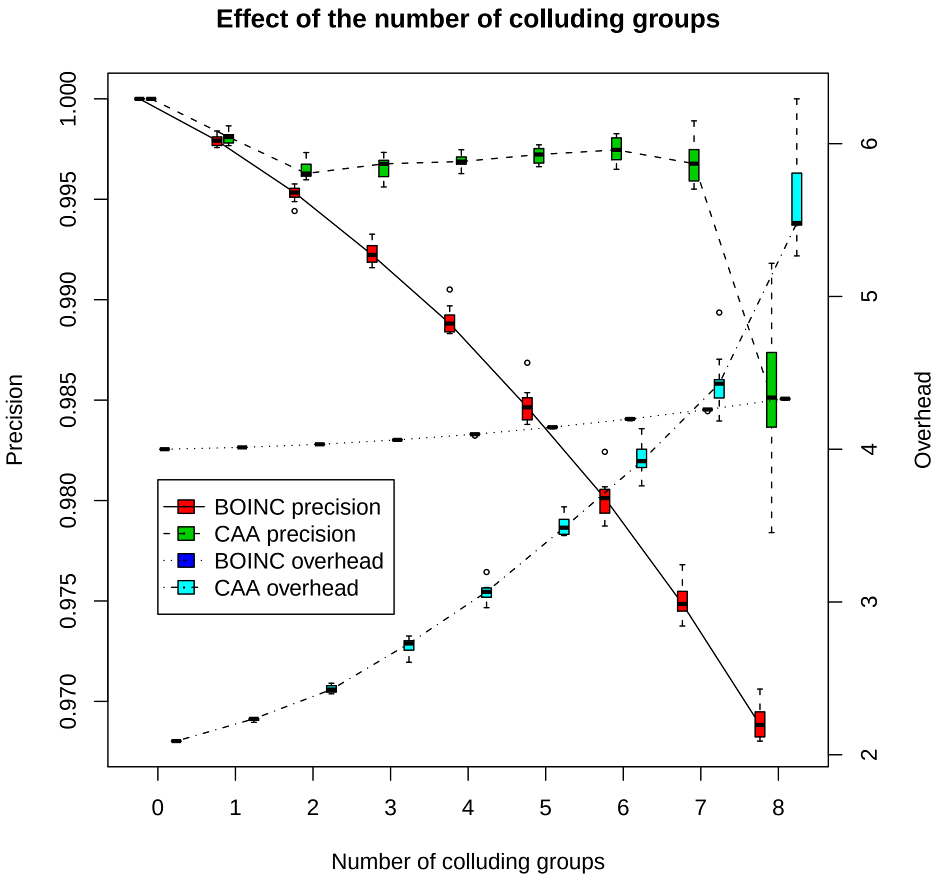

Figure 6: Performance of two methods with 2,000 workers, fully reliable plus several colluding groups with 200 workers each and a 25% collusion probability.

For our approach, the precision remains stable for all of the settings. Additionally, the overhead has a low variation (between 3.8 and 4.8). We conclude that inter-collusion has no impact on our method.

For studying the effect of reliability (Figure 9), the collusion settings are: 2 inter-colluding groups each having 2 colluding groups; the inter-collusion probability is 15% and the collusion probability is 10%; each colluding groups has 200 workers. Moreover, the fraction of reliable workers and the failure probability of the unreliable workers are varying.

As expected, the BOINC algorithm requires more results for each job as the platform is less reliable, and even with higher duplication, its precision decreases (the imprecision when the platform is reliable is caused by the colluding groups). In contrast, our method is insensitive to the reliability in term of precision. As expected, the overhead of our method increases and grows larger than the BOINC overhead for highly unreliable platforms.

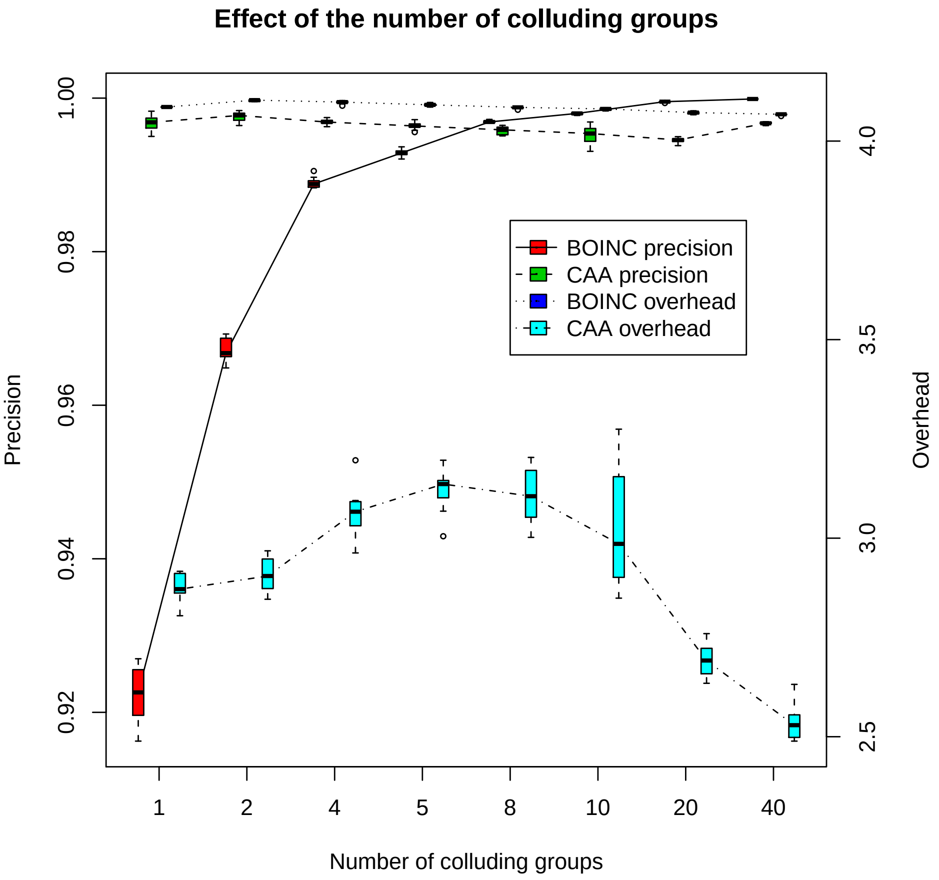

Figure 7: Performance of two methods with 2,000 workers, fully reliable plus several colluding groups with 800 workers on total and 25% collusion probabilities.

The last figure (Figure 10) depicts the performance of our method using different availability traces and number of workers. Both *Overnet* and *Microsoft* traces belong to FTA. On overall, the BOINC algorithm and our method have relatively comparable behaviors with the different traces.

The studies that are described from Figure 4 to 10 allow us to conclude that our method is resistant to collusion in the following scenarios: when the colluding behaviors are not deterministic; when several colluding groups coexist; when distinct colluding groups cooperate together; and, when the machines are unreliable. Additionally, the overhead of our method adapts to the degree of adversity that is present in the network. Lastly, in most situations, our method is able to certify results correctly while keeping a low overhead.

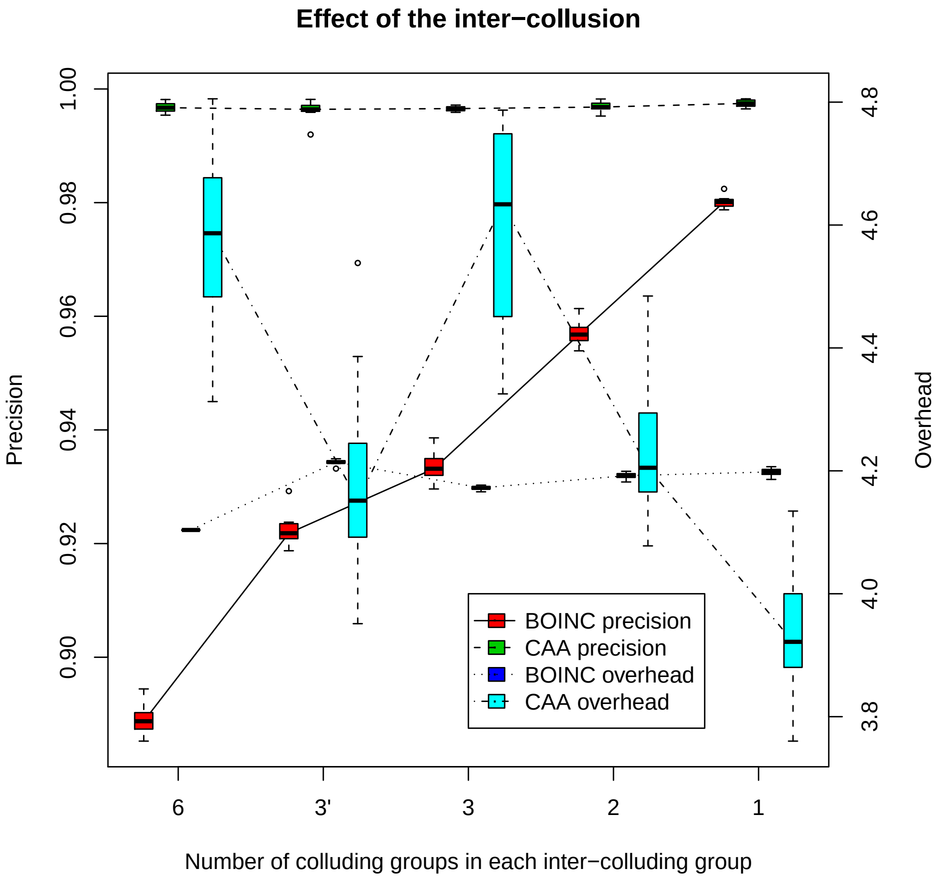

Figure 8: Performance of two methods with 2,000 workers, fully reliable plus 6 colluding groups with 200 workers each, a 10% collusion probability and a 15% inter-collusion probability.

9 Conclusion

Coordinated attacks against a desktop grid remain a major threat to their correct functioning. Solutions adopted in standard environments for tackling the issue of correctness (such as the quorum in BOINC) are not sufficient in the event of collusion.

In this paper, we have proposed a modular framework based on three components (a characterization system, job allocation, and result certification) to address this problem. Moreover, we present a method for determining the probability that a given result is correct. Last an algorithm for allocating jobs to resources and certifying results is described. For the characterization system, we use the agreement method described in our previous work.

Despite the fact that the threat model is very strong (a worker may belong to one or several groups, groups can cooperate, colluders may sometime send

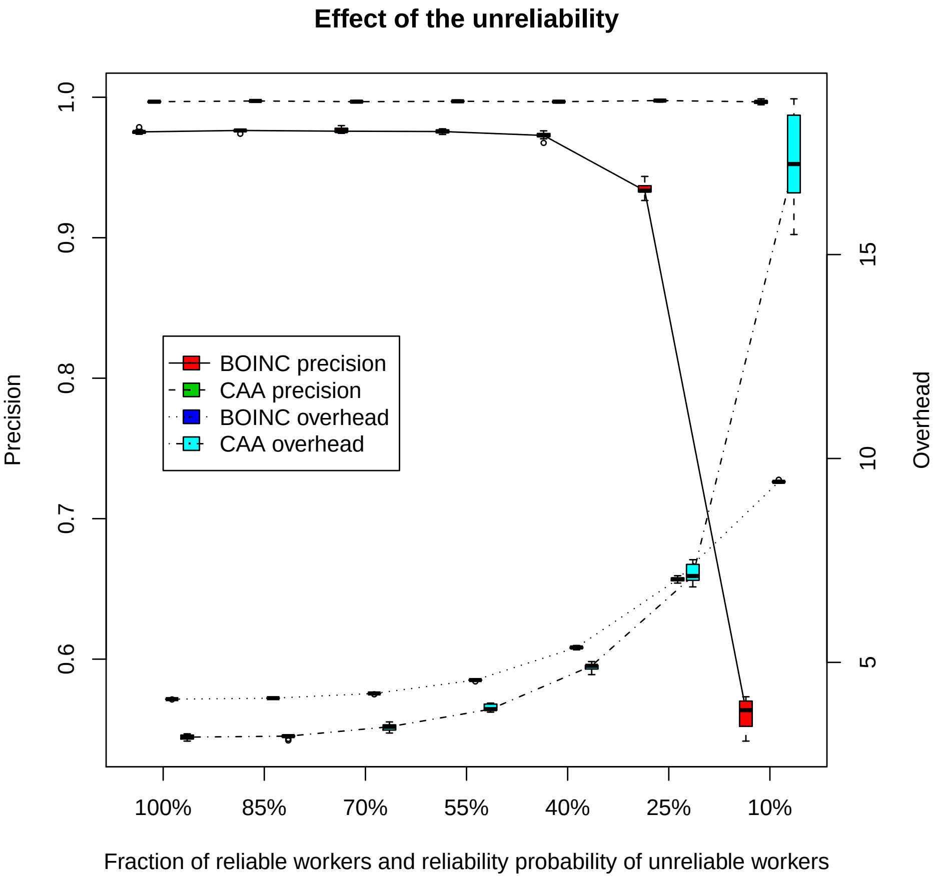

Figure 9: Performance of two methods with 2,000 workers, varying unreliability and 4 colluding groups with 200 workers each, a 10% collusion probability and a 15% inter-collusion probability (2 non-overlapping inter-colluding groups).

a correct result to stay undetected, colluders are not required to synchronize to send the same incorrect result, and no information is known a-priori by the server), the proposed solution is very effective. Indeed, experimental results, based on real traces, shows that in almost all the cases we outperform the quorum strategy of BOINC both in terms of precision and overhead.

The fact that precision is better in our case than for quorum relies on the fact that we take into account collusion and that we prevent simple synchronization of workers by never assigning the same job to several different workers at the same time. The fact that the overhead is lower in our case clearly shows that the quorum strategy is also suboptimal in terms of resource usage and tends to over-provision workers for computing a given job.

Another remark from the experiment is that to best attack our proposed solution, colluders have to split into small groups and never always collude. In

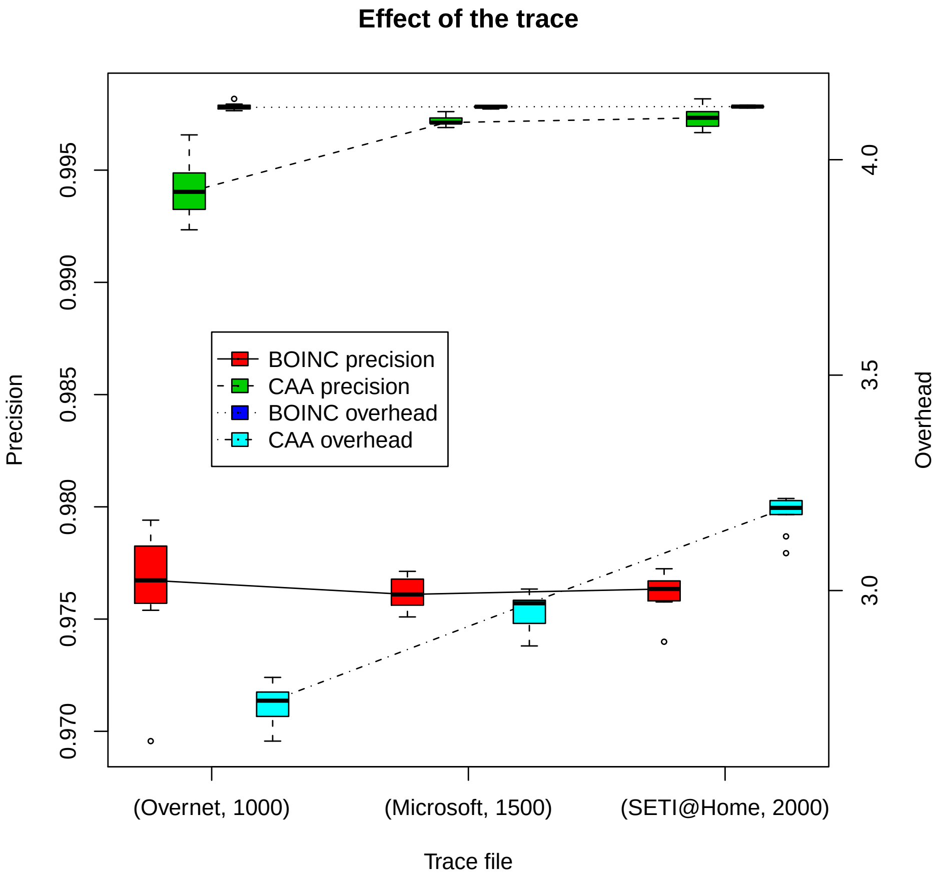

Figure 10: Performance of two methods with varying trace files, 85% of reliable workers, 4 colluding groups with 200 workers each, a 10% collusion probability and a 15% inter-collusion probability (2 non-overlapping inter-colluding groups).

this case, strategies such as black-listing are inefficient and suboptimal: as in this case colluders send correct results most of the time.

Future works are directed towards non-stationarity (when worker behavior changes with time). A simple method would be to reset the probabilities from time to time. Other techniques for non-stationarity including aging of prior observations to enable more rapid transitions will be studied. To lift the assumption that we need a majority of honest workers we plan to improve the characterization system to use trustworthy "beacon" resources.

References

[1] D. P. Anderson, “Boinc: A system for public-resource computing and storage,” in *GRID*. IEEE Computer Society, 2004, pp. 4–10.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: An Experiment in Public-Resource Computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[3] "Climateprediction.net," http://climateprediction.net/.

[4] "Folding@home," http://folding.stanford.edu/.

[5] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids," in *23th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Rome, Italy, May 2009, pp. 1–12.

[6] S. Zhao, V. M. Lo, and C. Gauthier-Dickey, "Result verification and trust-based scheduling in peer-to-peer grids," in *Peer-to-Peer Computing.* IEEE Computer Society, 2005, pp. 31–38.

[7] J. D. Sonnek, A. Chandra, and J. B. Weissman, "Adaptive reputation-based scheduling on unreliable distributed infrastructures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 11, pp. 1551–1564, 2007.

[8] L.-C. Canon, E. Jeannot, and J. Weissman, "A Dynamic Approach for Characterizing Collusion in Desktop Grids," in *24th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Atlanta, Georgia, USA, Apr. 2010.

[9] J. R. Douceur, "The sybil attack," in *IPTPS*, ser. Lecture Notes in Computer Science, vol. 2429. Springer, 2002, pp. 251–260.

[10] J. Jung, R. A. Milito, and V. Paxson, "On the adaptive real-time detection of fast-propagating network worms," *Journal in Computer Virology*, vol. 4, no. 3, pp. 197–210, 2008.

[11] A. Fernández, L. López, A. Santos, and C. Georgiou, "Reliably executing tasks in the presence of untrusted entities," in *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, 2006.

[12] L. F. G. Sarmenta, "Sabotage-tolerance mechanisms for volunteer computing systems," *Future Generation Comp. Syst.*, vol. 18, no. 4, pp. 561–572, 2002.

[13] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The EigenTrust Algorithm for Reputation Management in P2P Networks," in *WWW '03: Proceedings of the 12th international conference on World Wide Web.* New York, NY, USA: ACM, 2003, pp. 640–651.

[14] A. Jøsang, "An algebra for assessing trust in certification chains," in *NDSS.* The Internet Society, 1999.

[15] P. Domingues, B. Sousa, and L. M. Silva, "Sabotage-tolerance and trust management in desktop grid computing," *Future Generation Comp. Syst.*, vol. 23, no. 7, pp. 904–912, 2007.

[16] M. Yurkewych, B. N. Levine, and A. L. Rosenberg, "On the cost-ineffectiveness of redundancy in commercial p2p computing," in *ACM Conference on Computer and Communications Security*. ACM, 2005, pp. 280–288.

[17] E. Staab and T. Engel, "Collusion detection for grid computing," in *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 412–419.

[18] A. W. Krings, J.-L. Roch, S. Jafar, and S. Varrette, "A probabilistic approach for task and result certification of large-scale distributed applications in hostile environments," in *EGC*, ser. Lecture Notes in Computer Science, P. M. A. Sloot, A. G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, Eds., vol. 3470. Springer, 2005, pp. 323–333.

[19] G. Silaghi, P. Domingues, F. Araujo, L. M. Silva, and A. Arenas, "Defeating Colluding Nodes in Desktop Grid Computing Platforms," in *Parallel and Distributed Processing (IPDPS)*, Miami, Florida, USA, Apr. 2008, pp. 1–8.

[20] L.-C. Canon, E. Jeannot, and J. Weissman, "A Scheduling Algorithm for Defeating Collusion," INRIA, Research Report RR-7403, Oct. 2010. [Online]. Available: http://www.labri.fr/~ejeannot/publications/RR-7403.pdf

[21] D. Kondo, B. Javadi, A. Iosup, and D. H. J. Epema, "The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems," in *CCGRID*. IEEE, 2010, pp. 398–407.

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex (France)

Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399