A Scheduling and Certification Algorithm for Defeating Collusion in Desktop Grids

Louis-Claude Canon*, Emmanuel Jeannot†, Jon Weissman‡
*Grenoble-INP, Grenoble University, INRIA
†INRIA, LaBRI–Bordeaux, France
‡Dept. of Computer Science and Engineering
University of Minnesota, Twin Cities–Minneapolis, USA

***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 their scalability and low cost, relying on external resources may compromise the correctness of application execution due to the well-known unreliability of nodes. In this paper, we consider a very challenging threat model: correlated errors caused either by organized groups of cheaters that may collude to produce incorrect results, or by buggy or so-called "unofficial" clients. By using a previously described on-line algorithm for detecting collusion and characterizing the participant behaviors, we propose a scheduling and result certification algorithm that tackles collusion. Using several real-life traces, we show that our approach minimizes both replication overhead and the number of incorrectly certified results.**

***Keywords*-Desktop Grid; Collusion; Modeling; Sabotage**

I. Introduction

Volunteer platforms such as desktop grids remain an attractive environment for many science and engineering applications [1]–[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 isolated configuration problem at a node.

*This work is partially supported by the French Ministère de l'Économie, de l'Industrie, et de l'Emploi and the Global competitive cluster Minalogic, project SHIVA no 09-2-93-0473.

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 match 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 incorrect 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 extend that work and present a new collusion-resistant algorithm for on-line task scheduling and result certification that works hand-in-hand with the computation of collusion 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 offers collusion avoidance by temporally staggering same task allocation to prevent easy worker synchronization under malicious colluding. 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 [1] replication algorithm that does not account for correlated errors. The results show that for a broad set of collusion scenarios, not

only does our algorithm achieves significantly lower result certification inaccuracy, it also exhibits smaller replication overhead than BOINC in most cases.

II. 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 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]. However, these papers do not address the task allocation problem in the presence of collusion. 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.

III. Models and Definitions

A. Application and Platform

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

- We are given an unlimited set of *jobs* to be executed. For each job 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*.
- We have a pool W of *workers* each able to compute any job and worker reliability may fluctuate. 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 in a pull-based manner based on its internal policy (*e.g*, redundancy, quorum) and algorithm. 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.

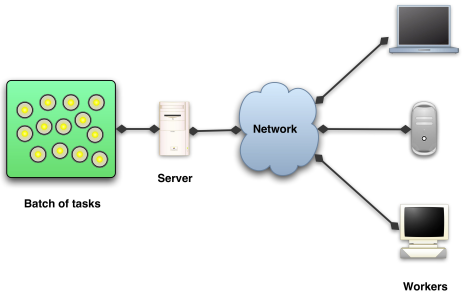

Figure 1: A Desktop Grid.

B. Threat Model

Collusion is defined as the correspondence between incorrect results sent by multiple workers. We distinguish between two types of collusion. The first is when several saboteurs voluntarily cooperate to send more than one incorrect result, thus 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 a *colluding group*. In order to reduce the chance of being detected, members of a group may sometimes act as colluders (returning the same incorrect result) and sometimes as non-colluding (returning the same correct result). The probability that a group decides to collude or not is fixed.

We model collusion as a fixed stochastic process where both the number of colluders and their likelihood of collusion is a drawn from a fixed probability distribution. We address the issue of non-stationarity as future work discussed later. We also model a case where colluders may initially "decompose" into smaller groups to avoid

detection and then occasionally collude across groups called inter-collusion (*i.e.*, where 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 by returning a unique incorrect result. We assume that non-colluding workers form a majority - either absolute over all workers, or relative, that is, larger than any single group of colluders. For sake of simplicity, we assume that if two or more groups do not collude together on a given job, then their decision to collude on this job is pairwise independent. Also, inter-collusion between two colluding groups is independent of the inter-collusion of two distinct colluding groups.

The set of groups (colluding and non-colluding) is denoted by G. A colluding group is defined as one that has a non-zero probability of collusion. To be efficient, colluders may 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: groups can cooperate, colluders may send correct results 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.

C. Metrics and Problem Definition

The performance of a scheduling system for desktop grids can be quantified according to two main criteria. The first concerns the efficiency of the platform usage. Small degrees of replication lead to greater efficiency or higher throughput. This criteria will be measured by the *overhead* defined as the average replication ratio, *i.e.*, the expected number of times each job is computed on a distinct worker. For example, a system that does not replicate jobs has minimum overhead.

The second 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. For assessing this criteria, we propose to measure the *inaccuracy*. The inaccuracy is the ratio of results incorrectly certified over the total number of certified results. The inaccuracy must be as close to zero as possible.

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 its 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 execution, 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 minimize both the inaccuracy and the overhead.

IV. Overview of the Architecture

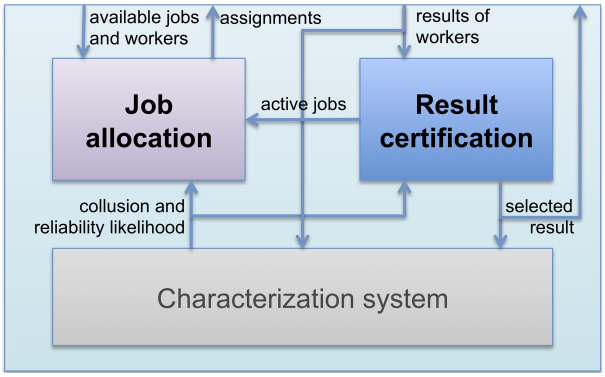

Figure 2: Architecture of the proposed solution. Job allocation and result certification (shown in bold) are described in this paper.

The architectural framework is based on three components (see Figure 2): a characterization system, a result certification system, and a job allocator. The first component characterizes the worker reliabilities and provides information to the other two components. The latter two components are new to this paper. In our previous work [8], we have proposed a *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). The precision of the system improves with time as more and more observations are made. 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 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 VI.

The *result certification* component is used to decide which result among the returned set is the correct one. The result estimated to be correct 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. A result with a high correctness probability is then certified. How this probability is estimated is described in Section V. When no result can be certified (because the confidence is too low), new results are computed until the confidence threshold is reached.

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

V. 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. Our procedure is to compute the probability that each result is correct, given: the results that were received, the workers that computed them and their estimated group membership and probabilities of collusion.

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*. Teams with only one worker are not considered. We denote by P the set of teams for a job. We denote F_P as the event that refers to a *specific* instance of P, that is a configuration of teams for a job that matches P. Although P and F_P are specific to each job, the correctness probability of any result is the same for any job. Thus, the specific job identifier need not appear in the notation. Let E_a be the event that all the workers in set a (a union of teams) collude to return the same incorrect result. If a is a team, then the event $\overline{E_a}$ occurs when the workers in a return the correct result. Therefore, the probability that the result returned by team i is correct given a configuration of teams F_P is $\Pr[\overline{E_i}|F_P]$. Note that this probability can be computed for each job by adapting the teams that are obtained each time.

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 colluding and non-colluding groups (*i.e.*, the set of groups G). The derivation that follows assumes that these probabilities and the composition of the groups (colluding and non-colluding) are exact. However, the characterization system provides only an estimation of these values. The impact of this approximation on the general criteria is assessed in Section VII.

Further, we assume several independence conditions (collusion within groups occur independently when there is no inter-collusion, and inter-collusions between distinct set of groups are independent). With this assumption, we can compute the exact value of $\Pr[\overline{E_i}|F_P]$. However, the cost of this evaluation is exponential in the number of workers. Hence, we propose two approximations. The first enables a tractable, yet precise evaluation. The second is used when the number of teams reaches a given limit in order to speed-up computation. We now provide some mathematical results that address these approximations.

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_{QP} that defines an intersection of distinct collusion events. The probability of this event is further bounded in Lemma 2 (our first approximation is to consider that the equality of the bound holds). To allow for an effective computation of this bound, Lemma 3 describes how a union of collusion events can be calculated (the second approximation is used when necessary). Due to lack of space, the proof of these lemmas are given in [20].

Let $I_{Q\,P}$ be the event that occurs when the workers in each of the teams in Q return the same incorrect result, 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. Then, $I_{Q\,P}$ is defined as: $I_{Q\,P} = \bigcap_{i\in Q}\left(E_i \cap \bigcap_{g\in h(P\setminus\{i\})} \overline{E_{i\cup g}}\right)$

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

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

The following lemma proposes a bound of $\Pr[I_{Q\,P}]$ 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:* $\Pr[I_{Q\,P}] \leq \prod_{i\in Q}\left(\Pr[E_i] - \Pr\left[\bigcup_{g\in h(P\setminus\{i\})} E_{i\cup g}\right]\right)$

Equality holds in Lemma 2 when the events $E_i \cap \bigcap_{g\in h(P\setminus\{i\})} \overline{E_{i\cup g}}$ are independent. With a small number of colluding groups or inter-colluding groups, this is the case and the computation is hence exact.

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 $i \in P$,* $\Pr\left[\bigcup_{g\in Q} E_{i\cup g}\right] =$

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

As the evaluation cost of this union is exponential in the number of teams, we can make 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) and computing the union of events reduces to a sum of the probabilities of each event.

To summarize, the correctness probability of a given result can be obtained using Lemma 1. It leads to the computation of probabilities of the form $\Pr[I_{Q\,P}]$ for which Lemma 2 proposes a bound. As the characterisation system is assumed to provide the probability that any subset of workers collude, these bounds require the computation of probabilities of the union of events E_a, which can be calculated with Lemma 3.

As stated in Section IV, the characterization system is assumed to provide the probability of collusion for any subset of workers 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 (the correlation between these random variables is then null). 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 (the variance of the denominator is then discarded).

VI. Job Allocation and Result Certification

Each job can have 4 states (see Figure 3): *available* when it has never been assigned to a worker (J_v represents the set of available jobs); *processing* when one worker is computing this job (J_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 (J_c represents the set of active jobs); and *terminated* when a result has been certified for this job (J_t represents the set of terminated jobs).

Workers ask the server for jobs and the server first tries to assign active jobs. If there are no active jobs, then available jobs are tried next. Also, each assigned job to a given worker is different (no job is computed twice by the same worker).

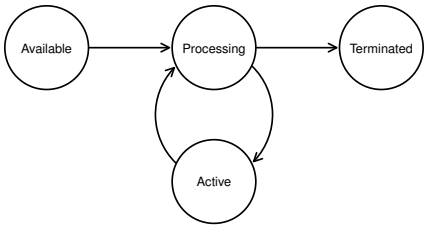

Figure 3: States and transitions between states for a job

The job is then processed by the worker and a result is returned to the server. The characterization system is informed that a result is returned.This information is used to improve the estimation of the group compositions and their collusion probabilities.

We assume that the probability of random (*i.e.*, non-colluding) matching errors to be negligibly small due to a sufficiently large answer space. Therefore, individual results cannot be certified. 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 the estimations (structure and probability) of the groups given by the characterization system, the probability p that the result is correct and the variance v of this probability (as it is a random variable) is computed using the lemmas of Section V. 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 the value of $p - \sqrt{v}$ 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 a large variance). This happens when a division by a value close to zero happens during the computation of p. To prevent this situation, we add another criterion: variance v must be greater than a second threshold (in practice, a large value like 0.1 is sufficiently discriminative). If $p-\sqrt{v}$ is under the $1-\epsilon$ 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 makes it hard for colluders to decide, out-of-band, to collude since they do not know if the same job will be submitted (or 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 too long to see if his/her cohorts have the same task, its job will time out. The combination of time staggering allocation and job time out makes result caching ineffective for colluders.

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 observed in a given simulation that there was at most one active job in 90% of the cases and only two active jobs in 8% of the cases. Hence, no other strategy than the random policy was tested. Moreover, as the jobs from the active pool are always chosen with higher priority over the available pool, this ensures that any job entering the active pool will necessarily becomes terminated provided that each execution is timed out. Therefore, if we consider the time during which a job stays active without being processed, these strategies are similar.

VII. Empirical Validation

We compare our method called CAA (Collusion Aware Algorithm) with the 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 of the simulations, we have fixed $\epsilon = 0.05$ for our approach.

A. Settings

To assess the performance of our method, we simulate a platform based on 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 provide the speed of each machine in number of floating operations per second. As these traces contain more than 220,000 workers, we limit the number of machines that are considered for scalability reasons. By default, we use 2,000 workers.

The simulated workload is also based on a trace. Michela Taufer provided a workload trace of the Docking@Home volunteer project to us. Each job is characterized by an estimated number of floating operations to perform. The duration of any job execution is then given by its cost over the speed of the machine. The provided workload contains about 150,000 jobs. This workload is replicated until one million jobs have been computed. A simulation terminates 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 inaccuracy and the overhead 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 vary a set of parameters that control the degree of platform reliability relative to our threat model. Every worker will return the correct result for any job by default, except if it fails or colludes. Failures are modeled as follows: a percentage of workers are completely reliable; unreliable workers return an incorrect result with the same reliability probability. For colluding groups, a set of worker participation percentages (*e.g.*, $k\%$ of workers will collude) with corresponding collusion probabilities are given. According to these values, workers are grouped into either the non-colluding group or a colluding group over time. A worker that is in a colluding group returns a colluding result (incorrect) with the associated collusion probability. To specify cooperation between colluding groups (inter-collusion), a set of colluding groups will collude with an inter-collusion probability. Several inter-collusion relations may be defined, however, the resulting scenario must be feasible (the aggregated collusion and inter-collusion probabilities must not exceed 1). In our first study, we considered the platform to be fully reliable (i.e. no uncorrelated errors) and otherwise contains only one colluding group. In this case, we adjust the percentage of colluders and the collusion probability of this group within this study.

Finally, each threat scenario is instantiated 10 times with a different seed. As our study has 38 different scenarios, it contains 380 distinct simulations for our method and for BOINC.

B. Results Analysis

Figures 4 to 7 depict the inaccuracy and the overhead of both the BOINC and CAA algorithm. Each figure shows the effect of varying one parameter (or a combination of parameters) while the others are set to fixed values. The results 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. Recall that both criteria are to be minimized. Hence, if curve A is above curve B, our solution outperforms the BOINC algorithm for inaccuracy (the same for curve C and D concerning overhead).

We first study the impact of the collusion parameters related to a single colluding group on the performance

of the scheduling methods. To this end, we fix the average number of incorrectly generated results that are due to collusion. To do this, the product of the proportion of colluders and their probability to collude are held constant. 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 terms of inaccuracy, we observe that the BOINC algorithm is more resistant to small groups of colluders even though they collude more often than large groups. 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 threat 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 more challenging threat scenarios (e.g. when the colluding group is large). Overall, our method always dominates BOINC in these settings.

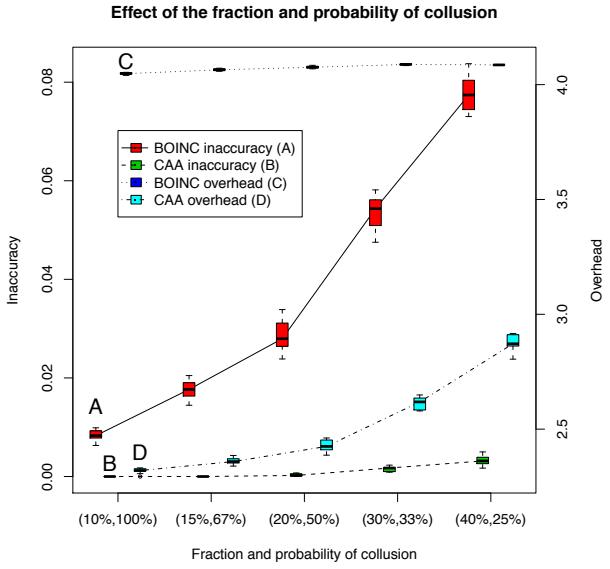

Figure 4: Performance of the two methods with 2,000 workers, fully reliable plus one colluding group with varying participation and probability (10% of the results returned by workers to the server are incorrect).

We next study the effect of collusion probability, while the size of a single colluding group is held constant. In Figure 5, the collusion probability of a group with 40% of the workers varies from 100% to 10%. The BOINC algorithm performance is the worst (for both inaccuracy and overhead) when the collusion probability is high. In term of inaccuracy, however, our approach suffers when the collusion probability is low. Indeed, this makes the colluding groups harder to detect for the underlying characterization system. There exists a specific collusion probability that maximizes the inaccuracy achieved by our method. This is intuitive as the corner cases are easy to detect: always collude or never collude. Therefore, a malicious peer should collude to achieve this probability. In the presence of a mechanism specifically designed to defeat collusion, they should use a low collusion probability. However, even in this case, our approach still outperforms the BOINC algorithm for both objectives.

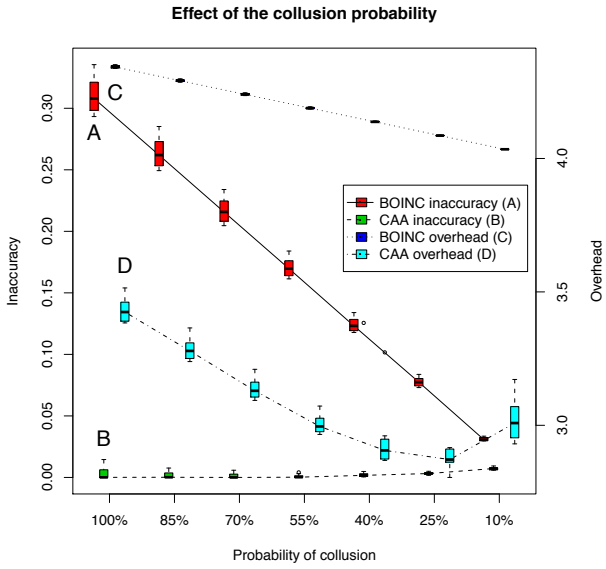

Figure 5: Performance of the two methods with 2,000 workers, fully reliable plus one colluding group with 800 workers and varying collusion probability.

We now focus our study on the case of several colluding groups. We choose a collusion probability of 25% because it intuitively represents a hard case (sufficient collusion to lead to certification errors, but perhaps small enough to make detection difficult). 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 inaccuracy, which is obtained with 8 colluding groups, is not bad (3%) given that there are only 20% honest 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 threat becomes stronger. The inaccuracy achieved by our method remains stable

except with 8 colluding groups. Overall, our method is always better except for the overhead with 7 and 8 groups.

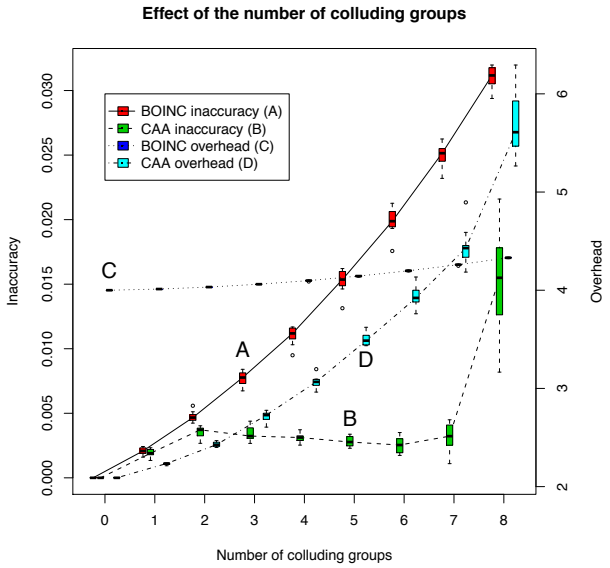

Figure 6: Performance of the two methods with 2,000 workers, fully reliable plus several colluding groups with 200 workers each and a 25% collusion probability.

In Figure 7, we see the effect of the number of colluding groups given that there are always 40% colluders with a 25% probability of collusion. Unsurprisingly, the BOINC inaccuracy decreases with the number of colluding groups as the probability that colluders achieve a quorum decreases. The inaccuracy 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 smaller groups. However, if they are too fragmented, they can no longer collude. As for the collusion probability, there exists an optimal number of colluding groups for the colluders. In this study, our approach dominates the BOINC algorithm up to 5 colluding groups. For more groups, our approach is less accurate but provides a far better overhead. This limit on the number of groups is related to our characterization system which is better at detecting larger groups. But even in the case of smaller groups, the inaccuracy is below 1%.

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, we create 3 inter-colluding groups. With 3 colluding groups, we create 2 inter-colluding groups.

The inaccuracy of the BOINC algorithm decreases

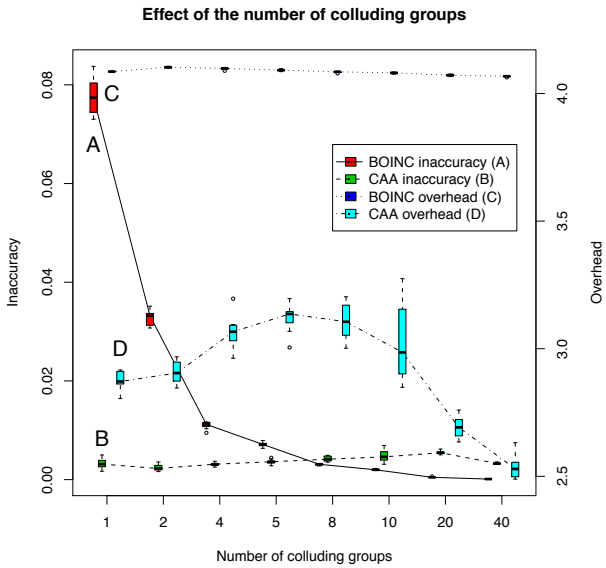

Figure 7: Performance of the two methods with 2,000 workers, fully reliable plus several colluding groups with 800 workers on total and 25% collusion probabilities.

as the number of colluding groups per inter-colluding group decreases. As the degree of cooperation decreases, a majority is harder to achieve and the quorum-based mechanism fails less often. For our approach, the accuracy remains stable for all of the settings, which indicates that inter-collusion has little or no impact on our method.

To study the effect of reliability (i.e. no uncorrelated errors), the collusion settings are: 2 inter-colluding groups each having 2 colluding groups, the inter-collusion probability is 15%, the collusion probability is 10%, and each colluding group has 200 workers. The fraction of reliable workers and the failure probability of the unreliable workers are varying. The results are shown in Figure 9. The BOINC algorithm requires greater replication for each job as the platform is less reliable. Despite higher replication, its inaccuracy increases due to the presence of colluding groups. In contrast, our method is insensitive to the reliability in term of accuracy. The overhead of our method also increases and grows larger than the BOINC overhead for highly unreliable platforms.

The last figure (Figure 10) depicts the performance of our method using different availability traces and number of workers. Both *Overnet* and *Microsoft* traces are part of the FTA. Overall, the BOINC algorithm and our method have relatively comparable behaviors across the different traces.

The results in Figures 4 to 10 allow us to conclude

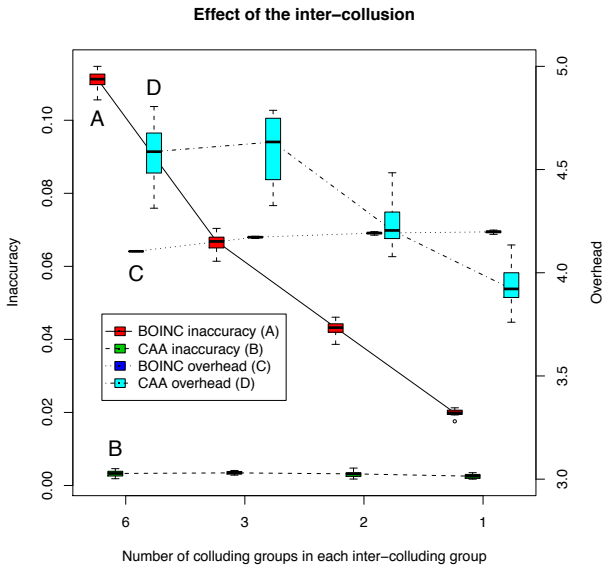

Figure 8: Performance of the two methods with 2,000 workers, fully reliable plus 6 colluding groups with 200 workers each, a 10% collusion probability and a 15% inter-collusion probability.

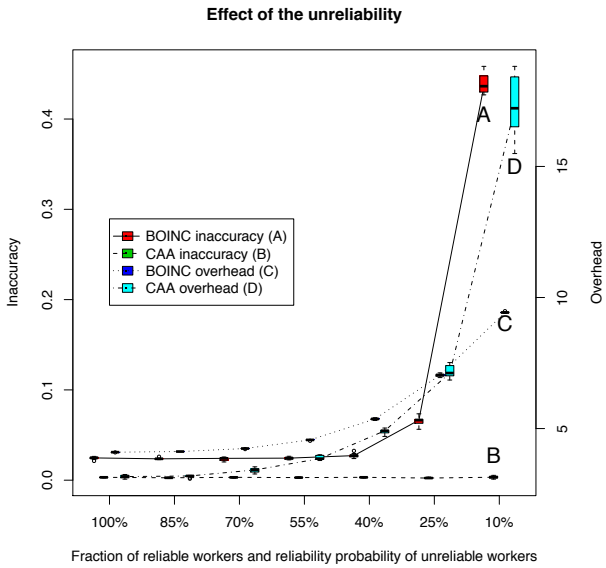

Figure 9: Performance of the two methods with 2,000 workers, varying unreliability and 4 colluding groups with 200 workers each, a 10% collusion probability and a 15% inter-collusion probability (2 non-overlapping inter-colluding groups).

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 collusion that is present in the network. Lastly, in most situations, our method is able to certify results correctly while keeping overhead low.

VIII. Conclusion

Coordinated attacks against a desktop grid remain a major threat to their correct functioning. Standard solutions for achieving correctness (such as the quorum in BOINC) are not sufficient in the event of collusion. Indeed, as shown in our experiments, inaccuracy of the results output by a BOINC-like system can be as high as 30% 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 (groups can cooperate, colluders may sometime send 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. Experimental results, based on real traces, shows that our approach outperforms the quorum strategy of BOINC in 89% of the cases both in terms of inaccuracy and overhead. Moreover, the inaccuracy of our solution is below 1% in 97% of the cases (even in unfavorable scenarios).

The fact that our accuracy is better relies on the fact that we take collusion into account and we make very hard for colluders to make a collusion decision due to temporal staggering and job time out. We also show that the quorum strategy is suboptimal in terms of resource usage and tends to over-provision workers for computing a given job, relative to our approach.

Future work is 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 the aging of prior observations to enable more rapid transitions will be studied.

References

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

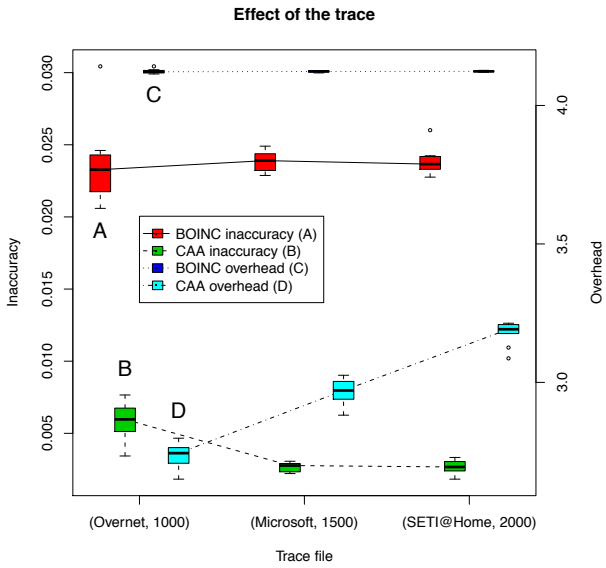

Figure 10: Performance of the two methods with varying trace files, 85% of reliable workers, 4 colluding groups with 200 workers each, a 10% collusion probability and a 15% inter-collusion probability (2 non-overlapping inter-colluding groups).

[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.