Parallel Processing Letters

RELIABLE INTERNET-BASED MASTER-WORKER COMPUTING IN THE PRESENCE OF MALICIOUS WORKERS*

ANTONIO FERNÁNDEZ ANTA
Institute IMDEA Networks,
28918 Madrid, Spain

CHRYSSIS GEORGIOU
Dept. of Computer Science, University of Cyprus
1678 Nicosia, Cyprus

LUIS LÓPEZ, AGUSTÍN SANTOS
LADyR, GSyC, Universidad Rey Juan Carlos,
28933 Mostoles, Spain

ABSTRACT

We consider a Master-Worker distributed system where a master processor assigns, over the Internet, tasks to a collection of n workers, which are untrusted and might act maliciously. In addition, a worker may not reply to the master, or its reply may not reach the master, due to unavailabilities or failures of the worker or the network. Each task returns a value, and the goal is for the master to accept only correct values with high probability. Furthermore, we assume that the service provided by the workers is not free; for each task that a worker is assigned, the master is charged with a work-unit. Therefore, considering a single task assigned to several workers, our objective is to have the master processor to accept the correct value of the task with high probability, with the smallest possible amount of work (number of workers the master assigns the task). We probabilistically bound the number of faulty processors by assuming a known probability $p < 1/2$ of any processor to be faulty.

Our work demonstrates that *it is possible* to obtain, *with provable analytical guarantees*, high probability of correct acceptance with low work. In particular, we first show lower bounds on the minimum amount of (expected) work required, so that any algorithm accepts the correct value with probability of success $1-\varepsilon$, where $\varepsilon \ll 1$ (e.g., $1/n$). Then we develop and analyze two algorithms, each using a different decision strategy, and show that both algorithms obtain the same probability of success $1-\varepsilon$, and in doing so, they require similar upper bounds on the (expected) work. Furthermore, under certain conditions, these upper bounds are *asymptotically optimal* with respect to our lower bounds.

Keywords: Volunteer computing, Malicious processors, Dependable computing, Public-resource computing, Task execution, Cost.

*A conference version of this work appeared as [12]. This work is partially supported by the Cyprus Research Promotion Foundation Grant ΤΠΕ/ΠΛΗΡΟ(ΒΕ)/0609/05, Comunidad de Madrid grant S2009TIC-1692, and Spanish MICINN grant TIN2008–06735-C02-01.

1. Introduction

Problem and Motivation. The demand for processing large amounts of data has increased over the last decade. As traditional one-processor machines have limited computational power, distributed systems consisting of hundreds of thousands of cooperating processing units are used instead. An example of such a massive distributed, Internet-based, cooperative computation is the SETI@home project [21], which was the first instance of *volunteer computing*. As the search for extraterrestrial intelligence involves the analysis of gigabytes of raw data that a fixed-size collection of machines would not be able to effectively carry out, the data are distributed to millions of voluntary machines around the world. A machine acts as a server and sends data (aka tasks) to these client computers, which they process and report back the result of the task computation. However, these client computers are not trustworthy and might act maliciously (or simply be faulty). This gives rise to a crucial problem: *how can we prevent malicious clients from damaging the outcome of the overall computation?*

In this work we abstract this problem in the form of a distributed system consisting of a *master* fail-free processor M and a collection of n processors, called *workers*, that can execute tasks. Worker processors might act maliciously, that is, they are Byzantine [24]. Since each task returns a value, we want the master to accept only correct values with high probability. Namely, given $\varepsilon \ll 1$, we want a *probability of success* of at least $1 - \varepsilon$ (e.g., $1 - 1/n$). However, we assume that the service provided by the workers is not free (as opposed to the SETI@home project). For each task that a worker is assigned, the master computer is charged with a *work-unit*. Furthermore, workers can be slow or fail during the computation of a task, and messages can get lost or arrive late; in order to introduce these assumptions in the model, we consider that there is a known probability d (which may depend on n) of M receiving the reply from a given worker on time. We also consider that the number of malicious workers is bounded probabilistically: there is a known probability $p < 1/2$ of any processor to be faulty (p may depend on n). Given the above model, and considering a single task assigned to several workers, our goal is to have the master computer to accept, *with provable analytical guarantees*, the correct value of the task with probability of success at least $1 - \varepsilon$, and with the smallest possible amount of *work* (number of workers that M has assigned the task). (The problem and model are presented in detail in Section 2.)

Prior/Related Work. A real system that is very related to the model presented in this paper is the Berkeley Open Infrastructure for Network Computing (BOINC) [3, 4]. This system allows volunteers to provide free computational cycles to perform intensive computation in a form similar to the one proposed in this paper. With BOINC, an application can submit to the system a task to be executed. Then, instances of the task are dispatched to several clients and a validation process is used to decide which returned value to accept as correct output of the task. In BOINC the number of instances of a task executed and the validation procedure is provided by the application. Like in the original SETI@home system, and unlike in our model,

applications in BOINC are not restricted on the number of instances of a task they request and are not charged for the computational power they use. This could be dangerous if applications act selfishly and start a large number of instances. On the other hand, application programmers may not have enough information to be able to appropriately tune up the number of instances and the validation mechanism.

An interesting model of volunteer computing was proposed by Sarmenta [28]. This work assumes that workers are malicious (he calls them *saboteurs*) with probability f, and that a malicious worker returns a wrong answer with probability s. It considers a collection of tasks instead of a single one, and the objective is to bound the expected number of wrong results accepted by the master, and the amount of work performed. It compares several mechanisms to achieve this, namely, majority for each task and spot-checking (giving workers tasks whose result is known to identify malicious workers). This second mechanism can be combined with blacklisting or credibility techniques. Compared with the model in this paper, it assumes that all workers reply, while we have a explicit parameter d to model the worker's unavailability and the network's unreliability.

The mechanisms proposed by Sarmenta [28] are contrasted with real data by Kondo et al. [19]. They analyze the errors of a volunteer computing system reaching some interesting conclusions. For instance, they show little correlation between simultaneous malicious workers. Additionally, there is a large variability of the set of malicious workers over time, with the exception of a few frequent offenders. They conclude that care has to be taken if blacklisting or credibility is used. They also conclude that a large number of tasks and time are required to achieve low error rates with spot-checking, and that, in general, to achieve low errors it is better to use majority.

Motivated by the conference version of this work [12], Konwar et al. [20] have studied an extension of our problem in which there are n workers and n tasks to be performed. Their computational model is somewhat stronger than the one used here (and more similar to that of Sarmenta [28]), since they assume a synchronous system in which the result of a task assigned to a non-faulty worker is always received by the master on time. This allows them to obtain efficient algorithms even if the failure parameter p is unknown (they can efficiently estimate it).

As mentioned, the above papers assume that the replies from all the workers are received by the master. However, the characteristics of volunteer computing allows to make few guarantees on communication and computation reliability. For instance, Heien at al. [17] have found that in BOINC only around 5% of the workers are available more than 80% of the time, and that half of the workers are available less than 40% of the time. This fact, combined with the length of the computation incurred by a task, justifies the interest of considering in the model of volunteer computing the possibility of workers not replying (as modeled by parameter d). Additionally, the large typical times to run tasks justifies the use of single-round mechanisms (as considered in this work).

In a related research direction, there is work (e.g., [30, 9]) that has considered

similar master-worker frameworks, where the workers are not assumed to be destructively malicious (as assumed in this paper and in all previously mentioned models) but instead are assumed to be *rational* [1] and simply try to maximize their benefit (that is, workers act based on their own self-interest). This assumption leads to game-theoretic approaches to the master-workers problem, in which the master has to find ways to incentive workers to return the correct reply. This has been usually done by having the master verifying/auditing the values returned with some probability, and punishing the workers caught lying. More recently, Fernández Anta et al. [10] considered a master-worker framework with altruistic, malicious and rational workers. For this purpose, an approach that combines voting strategies with game-theoretic and mechanism design incentive-based techniques was deployed. Unlike the present work, where the performance measure is the number of workers used for performing the task, in [10] the measure to be optimized is the master's utility (a quantified metric that takes into consideration various system parameters and reward/punishment schemes). Due to the very different objectives and mechanisms deployed in [10], the results of [10] and of the present paper, cannot be compared, *even* if no rational workers are assumed in [10]. The present work studies thoroughly the case of malicious workers having as a goal to minimize the number of workers while obtaining the desired success probability without relying on any incentive-based mechanisms.

The problem we consider in this work is clearly related to the *voting problems* (e.g., [5, 26, 22]). In these problems there is a set of entities or "voters," some of which can be faulty. Each voter proposes a value (usually obtained from some computation) to a deciding agent, such that non-faulty voters always propose the correct value, while faulty voters can have different behaviors. From the set of proposed values, the agent uses a strategy to choose a value that it believes to be the correct one. The purpose of a good strategy is to maximize the probability of choosing the correct value. The main difference of these problems with the problem studied in this paper is that they usually assume that all the entities in the system propose a value, that these values are received by the deciding agent, and only the probability of a bad choice has to be minimized.

A related reference from voting to the present work is that of Paquette and Pelc [26]. There, the authors consider a voting system in which voters can lie with different and known probabilities. In this general framework, synchronous and asynchronous models are considered. The main difference between [26] and the present paper has to do with the problem statement. While they want to maximize the probability of a correct decision, in this paper the target is to minimize the cost (i.e., the number of selected workers), while maintaining a certain level of confidence. A second difference has to do with the modeling and analysis of asynchrony. In their model they assume that the message latencies are controlled by an adversary. Consequently, the decision can be arbitrarily delayed. Additionally, the decision considers a fixed number of values (with a default value if the node/s response has not been received). In the model considered here, the latency is modeled with a

probability of late arrival d, and the decision is taken after a fixed amount of time with only the values received are considered to make it.

Another problem related to the problem we consider in this work is the *Do-All* problem, in which a collection of k message-passing processors need to cooperatively perform t independent tasks in the presence of failures (e.g., [6, 15]). The work in [11] has considered this problem under Byzantine processors. Several deterministic lower and upper bound results were introduced on the complexity of solving the *Do-All* problem in a *synchronous* distributed system where up to f nodes might behave maliciously. Although the idea of reliably executing tasks in the presence of malicious processors is the same, both the model and the problem we consider here are different. For example, in [11], processors attempt to collectively decide whether a task has been correctly performed without in fact having to learn the result of the task, as opposed to our problem where a single processor must decide the validity of a task result (and of course obtain that value).

There is an interesting connection between the problem considered in this work and the problems of reliably computing Boolean k-variable functions with noisy Boolean circuits (e.g., [27, 13]), noisy Boolean decisions trees (e.g., [27, 18, 8, 7]), and noisy broadcast (e.g., [14, 16, 23]). Also, the fact that the master has to decide upfront the number of queries connects our model with the model of *static* noisy Boolean decision trees. In particular, our problem can be viewed as the problem of reliably computing the identity function of one variable ($F(x) = x$) with a noisy static Boolean decision tree. However, we have identified several differences between our model and the models considered in the literature for these problems. For example, in their models, a query of a bit always returns an answer (0 or 1) as opposed to our model in which it is possible not to get a reply for a query (either a malicious worker chooses not to reply or a message is not received on time). Recent work [29] investigated the reliable computation of Boolean k-variable functions assuming that ℓ p-faulty copies of each input bit are received. However, it is assumed that ℓ is fixed as opposed to our model where the number of received replies is not fixed.

Differences exist also in the complexity measures considered. In noisy circuits and decision trees, upper and lower bounds are usually given as functions of either (a) the *sensitivity* s (or the critical number) of a function (number of bits that are critical for the correct computation of the function), or (b) simply the number of variables k of the function. Moreover, it is assumed that the probability p of a bit to be given incorrectly and the probability ε that the function is computed incorrectly are constants (we do not impose this restriction in our model). Therefore, the asymptotic bounds presented, especially the lower bounds (e.g., $\Omega(s \lg s)$ or $\Omega(k \lg k)$) are meaningless in our model, since $s = k = 1$ (it is worth mentioning that their analytical results leading to the asymptotic expressions are usually not dependent on p and ε). In fact, in this work, we present a new lower bound on the depth required by noisy static Boolean decision trees for the reliable computation of the identity function that *depends* on p and ε. In the noisy broadcast model, bounds are given as functions of the number of broadcasts needed to compute a

given function. Again, these bounds do not apply to our model, since in our model we have a single convergecast (from the workers to the Master) and not multiple broadcasts between the workers.

Contributions. Our work demonstrates, *with provable analytical guarantees*, that *it is possible* to execute tasks reliably with high probability and with low cost in the presence of malicious processors and processors whose reply may never be received. In particular:

- We present lower bounds on work. Specifically, we identify lower bounds on the minimum amount of (expected) work required, so that any algorithm accepts the correct value with probability of success $1-\varepsilon$. Furthermore, we derive a new lower bound on the depth of *noisy static Boolean decision trees* [27] required for the reliable computation of the 1-variable identity function ($F(x) = x$); the bound is expressed as a function of p and ε.
- We develop two algorithms: (a) the *Majority Based Algorithm* (MBA) and (b) the *Threshold Based Algorithm* (TBA). MBA is a simple and natural algorithm that uses a standard technique of having M randomly choosing a subset of the workers and taking the majority of the returned values as the correct answers. However TBA does not follow this pattern, since it is designed to decide as early as possible. In TBA, if M receives a certain number of responses with equal value (threshold) it makes a decision, otherwise it decides on the majority of the received responses. Algorithm TBA is *early-terminating* as opposed to MBA that always waits for a time T and then makes a decision on the value to accept.
- We analyze the algorithms using Chernoff bounds. Both algorithms obtain the same probability of success $1 - \varepsilon$ and we derive similar upper bounds on the (expected) work required in doing so, expressed as functions of ε, d, and p. Furthermore, for the cases where p is a constant, both algorithms achieve the same *asymptotic* upper bounds on (expected) work, which are *asymptotically optimal* with respect to our lower bounds; in this case the work complexity is $\Theta((-\lg \varepsilon)/d)$.

Paper Organization. In Section 2 we present the model and definitions. In Section 3 we present lower bounds on work in order to achieve high probability of correct decision in the model we consider. In Section 4 we present algorithms MBA and TBA and we show that they achieve the desired probability of correct decision while maintaining low work. We also compare our two algorithms and the lower bounds. We conclude in Section 5.

2. Model and Definitions

We study execution of tasks in a system in which the processors can behave maliciously, i.e., are Byzantine [24]. We assume there is a fail-free *master* processor M which has a task to be executed. This task returns a value, which M wants to reliably obtain. Following the typical Master-Worker volunteer computing framework, a set P of n processors, $P = \{1, ..., n\}$, that can execute the task, is made available to processor M. We refer to these processors as *workers*. The workers are

continuously waiting for M to assign them a task to execute, they execute a task if they are assigned one, and return the computed value (as depicted in Figure 1).

Processor $i \in P$, does:
1 Wait to receive from M a task to be executed
2 Execute the task
3 Send to M the computed value v

Fig. 1. Algorithm executed by any worker processor.

Workers are not considered to be trustworthy and in fact, they might act maliciously (e.g., they might send an incorrect value, send no value, etc.). However, we assume that a malicious processor, that is a faulty processor, cannot impersonate another processor and cannot modify nor remove other processors' messages (including M). Clearly, in order to be able to do anything useful, the number of processors that may fail has to be bounded. We consider a probabilistic bound on the number of malicious processors: there is a probability $p < 1/2$ of any processor to be faulty (each processor is faulty with probability p, independently of the rest of processors). We assume that the set of faulty processors is fixed before M assigns the task to the workers and it does not change during the execution. We also assume that M knows *a priori* the corresponding value p, but has no a priori knowledge of which processor can be faulty.

We further assume that processors are asynchronous with respect to each other and the communication between them is not reliable. Therefore, processors can be slow, and messages can get lost or arrive late. In order to incorporate these assumptions in the model, we consider that there is a known probability d of M receiving the reply from a given worker (that is willing to reply) on time. This probability is identically distributed and independent for each worker. The reply may not arrive on time due to several reasons: the worker never receives the message from M, M never receives the reply from the worker, or the whole process takes too much time and the reply is simply late. Note that we do not differentiate whether the worker is faulty or not.

Then, under this model we assume that M is given a task, whose correct output value is v, and a probability $\varepsilon \ll 1$ (e.g., $1/n$), and M must accept v with *success probability* of at least $1 - \varepsilon$ and low cost. By success probability we mean the probability of M deciding the correct value that the task returns. To attempt to decide the correct value, M must assign the execution of the task to a set of workers (not necessarily all of them), wait for replies from them, and decide from the replies obtained. We refer to the above procedure as a *round*. Note that we do not allow a second round to take place; M must accept a value at the end of the first (and only) round. This guarantees fast termination of algorithms. Note also that once M accepts a value, it is not allowed to change its decision and choose a different value. For each worker M assigned the task, M is charged with one *work-unit*. Given a task assignment, its cost, or *work,* is defined as the total number of work-units that M is charged for, that is, the total number of workers that M assigned the task.

Then, the objective is to minimize the (expected) work of the assignment while obtaining a success probability of at least $1 - \varepsilon$.

Finally, we assume that M has no *a priori* knowledge of the correct value to be computed by the task. Let V be the set of possible values returned by the task to be executed. This means that M has no information on the probability that each of the values in V has to be the solution of the task. Only one of these values is the correct value, which all correct workers return. A faulty worker can decide not to reply, or reply with any value in V (correct or incorrect). In order to analyze *worst case scenarios*, it is assumed full collusion among faulty workers, i.e., they coordinate their behavior in order to maximize the probability that M accepts an incorrect value (for example, faulty workers arrange to send the same incorrect value to M).

3. Lower Bounds on Work

In this section we give lower bounds on the (expected) work of any algorithm with success probability no less than $1-\varepsilon$. To do so, we lower bound the minimum number of replies M must have in order to decide with the desired success probability.

To simplify the analysis, it is considered that the set of possible returned values is $V = \{0, 1\}$. This assumption does not affect the model because M decides in one round. Then, to maximize the probability of M accepting an incorrect value, all faulty workers must reply to M with the same incorrect value, rather than choosing not to reply at all or to reply with the correct value (of course their message might be lost or delayed). Then, we assume that if $v \in \{0, 1\}$ is the result, they respond with $1 - v$.

We begin with the following lemma, which states that the algorithms that accept the most received value among the replies have the maximum success probability.

Lemma 3.1. *If an algorithm A has success probability $1 - \varepsilon$, then there is an algorithm A' with success probability no less than $1-\varepsilon$ that always accepts the most frequent value among the received replies, such that A' does not use more workers than A.*

The proof of the above lemma follows from a result proved in [26] which states that if $p < 1/2$ then in voting, optimal reliability is obtained by the majority strategy. Then, for the lower bounds we only need to consider algorithms that accept the most replied value. The following theorem shows that any algorithm must have runs in which the same task is assigned to a minimum number of workers.

Theorem 1. If workers fail with probability p, for any d, any algorithm must have runs in which it assigns the task to at least $2\frac{\lg \varepsilon}{\lg p} - 2$ workers in order to decide with probability of success at least $1 - \varepsilon$.

Proof. Suppose that the algorithm uses majority to decide and always assigns the task to less than $2\frac{\lg \varepsilon}{\lg p}-2$ workers. This implies that in each run M gets $r < 2\frac{\lg \varepsilon}{\lg p}-2 = 2\log_p \varepsilon - 2$ replies. Then, the probability that a majority of them come from faulty

processors is $\sum_{c>r/2}\binom{r}{c}p^c(1-p)^{r-c} \geq p^{\frac{r}{2}+1}\sum_{s=0}^{\lceil\frac{r}{2}\rceil-1}\binom{r}{\lfloor\frac{r}{2}\rfloor+s+1}p^s(1-p)^{\lceil\frac{r}{2}\rceil-1-s} \geq p^{\frac{r}{2}+1}\sum_{s=0}^{\lceil\frac{r}{2}\rceil-1}\binom{\lceil\frac{r}{2}\rceil-1}{s}p^s(1-p)^{\lceil\frac{r}{2}\rceil-1-s} = p^{\frac{r}{2}+1} > p^{\log_p\varepsilon} = \varepsilon$. The first inequalities follow from Calculus and the fact that $\binom{r}{\lfloor\frac{r}{2}\rfloor+s+1} \geq \binom{\lceil\frac{r}{2}\rceil-1}{s}$, for $s \in [0, \lceil\frac{r}{2}\rceil-1]$; the last inequality follows from the above bound on r.

If this happens, a majority of replies come from faulty processors which return the same incorrect value $1-v$, and M will decide incorrectly. Then, the success probability of the execution is below $1-\varepsilon$. Since this happens for all runs, the success probability of the majority algorithm is below $1-\varepsilon$. This and Lemma 3.1 complete the proof. □

The above Theorem leads to a new non-trivial lower bound result on the depth of *noisy static Boolean decision trees* [27].

Corollary 3.1. *Any noisy static Boolean decision tree for the function $F(x) = x$ when the error probability is p and the probability of a correct answer is at least $1-\varepsilon$ has depth at least $2\frac{\lg\varepsilon}{\lg p} - 2$.*

The above bounds show the existence of runs with a minimum number of processors assigned to a task, but do not give conditions on the distribution of these assignments. (The expected number of workers a task is assigned could be much lower.) The following results give lower bounds on the expected number of workers to which any algorithm assigns a task. These bounds are very close to the above bounds.

Theorem 2. If $\varepsilon \leq 1/2$, workers fail with probability p, and worker replies are delivered on time with probability d, then the expected number of workers to which any algorithm assigns a task must be more than $\frac{1}{d}(\frac{\lg(2\varepsilon)}{\lg p} - 1)$ in order to decide with probability of success at least $1-\varepsilon$.

Proof. Suppose that the algorithm uses majority to decide and assigns on average the tasks to no more than $\frac{1}{d}(\frac{\lg(2\varepsilon)}{\lg p} - 1)$ workers. This implies that M gets on average $\overline{R} \leq d(\frac{1}{d}(\frac{\lg(2\varepsilon)}{\lg p} - 1)) = \log_p(2\varepsilon) - 1$ replies. Let R be the random variable of number of replies obtained by M, using Markov's inequality we have that $\Pr\left[R \geq 2\overline{R}\right] \leq 1/2$. Then, we can lower bound the probability that in any run M gets less than $2\overline{R}$ replies and a majority of then return the same incorrect value $1-v$ as follows. Let X be the number of incorrect replies. Then, $\Pr\left[(R < 2\overline{R})\right]\Pr\left[X > R/2 | R < 2\overline{R}\right] \geq \frac{1}{2}\Pr\left[X = \lfloor R/2\rfloor + 1 | R < 2\overline{R}\right] \geq \frac{1}{2}p^{\lfloor R/2\rfloor+1} > \frac{1}{2}p^{\overline{R}+1} \geq \varepsilon$.

M will decide incorrectly only in this case, and hence the success probability of the majority algorithm is smaller than $1-\varepsilon$. This and Lemma 3.1 complete the proof. □

4. Proposed Algorithms

In this section we present two algorithms that the master processor M can run in order to solve the proposed problem. The first algorithm, called *Majority Based Al-*

gorithm (MBA) is a simple and natural algorithm where M decides on the majority of received responses. In the second algorithm, called *Threshold Based Algorithm* (TBA), if M receives a certain number of responses with equal value (threshold) it makes a decision, otherwise it decides on the majority of the received responses.

Both algorithms operate under a time restriction, that is, M needs to decide by some time T. More precisely, the value T determines how long M will wait for replies from the worker processors. Algorithm TBA might terminate before time T, that is, the algorithm is *early-terminating*. Following the definitions given in Section 2, d denotes the probability of M receiving a reply from a worker (that is willing to reply) within time T. Clearly, M can choose this parameter T to tune the probability d. Like in the previous section, to simplify the analysis, it is considered that the set of possible returned values is $V = \{0, 1\}$, and that all faulty workers return the incorrect value. This leads to worst case analyses.

4.1. *Algorithms Description*

4.1.1. *The Majority Based Algorithm*

We first present the Majority Based Algorithm (MBA). In this algorithm, processor M first chooses among the workers in set P a subset S and assigns the task to be executed to them. Then it waits for replies for a fixed time T. After that, it decides the value by simple voting (breaking ties at random). The workers in S are chosen *uniformly at random* from those in P. We consider two ways of choosing the subset S: either (i) M fixes the size s of S and chooses s processors uniformly at random from P, or (ii) M fixes a probability q and chooses each processor in P independently with probability q. Hence, M can choose either the size s or the probability q. The formulation of the MBA algorithm is shown in Figure 2.

Processor M does:
1 Choose a set $S \subseteq P$ uniformly at random
2 Send the task to be executed to the workers in S
3 Wait T time for replies from the workers in S
4 Accept v, where v is the most frequently returned value

Fig. 2. Majority based algorithm executed by master processor M.

4.1.2. *The Threshold Based Algorithm*

We now present the Threshold Based Algorithm. This early-terminating algorithm is described in pseudocode in Figure 3.

As in algorithm MBA, processor M chooses subset $S \subseteq P$ uniformly at random and either by fixing the size s of by fixing the probability q of a processor being chosen (see previous subsection). The threshold value τ is the number of equal replies (coming from workers in S) that will be needed to accept a given value v, before or on time T. The value of τ has to be large in order to prevent faulty processors to drive M to make a wrong decision. On the other hand, the value of τ should not be too large, because otherwise M will not get enough replies from

```
Processor M does:
1  Choose a set S ⊆ P uniformly at random
2  Send the task to be executed to the workers in S
3  Wait for replies from the workers in S
4  If there are τ replies with the same value v on or before time T
5      Accept v
6  Else
7      Accept v, where v is the most frequently returned value
```

Fig. 3. Threshold based algorithm executed by processor M.

correct processors to accept the correct value quickly. If by time T, M does not receive τ replies, then it follows the strategy of algorithm MBA and accepts the most frequently returned value v (breaking ties at random).

4.2. *Analyses*

In this section we bound the probability of failure for both algorithms. We have derived exact bounds for this probability, which give exact values for the probability of success. However, the expressions found are hard to handle in order to find the most appropriate parameters of the algorithms that M can use in each case. Even attempts for computing and plotting these values failed, as the computations require a big degree of floating point accuracy and range of arithmetic values. Therefore, we perform looser analyses with Chernoff bounds. These analyses allow us to obtain much simpler expressions to find suitable values for the parameters of the algorithms, and are easy to compute.

In the rest of the section we denote by X the random variable that accounts for the number of replies that the master M gets from faulty workers (that is, replies with the incorrect value) and by Y the random variable that accounts for the number of replies that M gets from non-faulty workers (that is, replies with the correct value by the end of period T). Hence, for the Majority Based Algorithm our target is to bound the probability $\Pr[X \geq Y]$.

Additionally, to simplify the analyses of the Threshold Based Algorithm algorithm we assume that, if M would have gotten τ replies from malicious workers by time T, it decides the incorrect value (this is like assuming that bad replies always reach M before the good ones). Moreover, even if M does not get τ bad replies, we assume that it decides the incorrect value unless it gets at least τ good replies (that is, in Line 7 of Figure 3 the incorrect value is always accepted). All these assumptions lead to a correct but pessimistic analysis. Hence, for the Threshold Based Algorithm, our target is to bound the probability $\Pr[(X \geq \tau) \vee (Y < \tau)]$.

4.3. *Analysis of the Majority Based Algorithm*

We now show that algorithm MBA achieves high probability of success while restricting the amount of work.

Theorem 3. Algorithm MBA guarantees a success probability of at least $1 - \varepsilon$ with (for $p < 1/4$ the values for $p = 1/4$ are used),

(a) Expected Work $E[|S|] = nq = \frac{18(\ln 2 - \ln \varepsilon)p}{(1-2p)^2 d}$ when parameter q is considered.
(b) Work $|S| = s = \lceil \frac{18(\ln 2 - \ln \varepsilon)p}{(1-2p)^2 d} \rceil$ when parameter s is considered.

Proof. As previously defined, X is the random variable that accounts for the number of replies that M gets from faulty workers and Y the random variable that accounts for the number of replies that M gets from non-faulty workers. Define $R = X + Y$, and let $\overline{R} = E[R]$ be its expectation. Then the probability of the algorithm MBA making an incorrect decision (that is, accepting the incorrect value) can be bounded as follows.

$$\begin{aligned}\Pr[X \geq Y] &= \sum_{c<2\overline{R}/3} \Pr[R=c] \Pr[X \geq c/2 | R=c] + \sum_{c \geq 2\overline{R}/3} \Pr[R=c] \Pr[X \geq c/2 | R=c] \\ &\leq \sum_{c<2\overline{R}/3} \Pr[R=c] + \sum_{c \geq 2\overline{R}/3} \Pr[R=c] \Pr[X \geq c/2 | R=c] .\end{aligned}$$

We now treat each term separately. The first term can be bounded with a Chernoff bound, like Theorem 4.5 of [25] with $\delta = 1/3$, as $\sum_{c<2\overline{R}/3} \Pr[R=c] = \Pr\left[R < 2\overline{R}/3\right] \leq e^{-\overline{R}/18}$.

To bound the second term, we use the following property, which trivially follows from $p < 1/2$. Let c and c' be two non-negative integers; if $c \leq c'$ then $\Pr[X \geq c/2 | R=c] \geq \Pr[X \geq c'/2 | R=c']$. This implies that $\sum_{c \geq 2\overline{R}/3} \Pr[R=c] \Pr[X \geq c/2 | R=c] \leq \Pr\left[X \geq \overline{R}/3 | R = 2\overline{R}/3\right] \sum_{c \geq 2\overline{R}/3} \Pr[R=c] \leq \Pr\left[X \geq \overline{R}/3 | R = 2\overline{R}/3\right]$.

We now use a Chernoff bound to bound this probability. Assume $R = 2\overline{R}/3$ and let $\mathcal{R}$ be the set of chosen processors whose reply arrives on time. We define, for each $i \in \mathcal{R}$, the Bernoulli random variable $X_i = 1$ if and only if processor i is faulty. Then it is easy to verify that $X = \sum_{i \in \mathcal{R}} X_i$ and that these variables are independent. Clearly, since $R = 2\overline{R}/3$ then the expected value of X is $\overline{X} = E[X] = 2p\overline{R}/3$. Then, we can apply Theorem 4.4 of [25] with $\delta = \frac{1}{2p} - 1$ as long as $1/4 \leq p < 1/2$ (to guarantee $0 < \delta \leq 1$), and obtain that $\Pr\left[X \geq \overline{R}/3 | R = 2\overline{R}/3\right] \leq e^{-\frac{(1-2p)^2 \overline{R}}{18p}}$.

Now, since $\frac{(1-2p)^2 \overline{R}}{18p} \leq \overline{R}/18$, we can add both bounds, and obtain that $\Pr[X \geq Y] \leq 2e^{-\frac{(1-2p)^2 \overline{R}}{18p}}$. In order to keep this value no larger than ε, it is enough to guarantee that $\overline{R} \geq \frac{18p(\ln 2 - \ln \varepsilon)}{(1-2p)^2}$. Since either $\overline{R} = sd$ or $\overline{R} = nqd$, the two cases of the statement of the theorem hold. □

4.4. *Analysis of the Threshold Based Algorithm*

We now show that algorithm TBA achieves high probability of correct acceptance with low (expected) work.

Theorem 4. Algorithm TBA guarantees a success probability of at least $1 - \varepsilon$ with (for $p < 1/6$ the values for $p = 1/6$ are used)

(a) $\tau = 2nqdp(1-p)$ and Expected Work $E[|S|] = nq = \frac{3(\ln 2 - \ln \varepsilon)}{(1-2p)^2 pd}$ when parameter q is considered.

(b) $\tau = 2sdp(1-p)$ and Work $|S| = s = \lceil \frac{3(\ln 2 - \ln \varepsilon)}{(1-2p)^2 pd} \rceil$ when parameter s is considered.

Proof. We first present a general analysis that is independent of the specific parameters considered (q or s) and then we derive the results for each case following the general analysis. We define the random variables X and Y as before. Our pessimistic view leads to the following non-success property for algorithm TBA: $\Pr[(X \geq \tau) \vee (Y < \tau)] \leq \varepsilon$.

To proceed we use Chernoff bounds. We define the appropriate Bernoulli variables later (when we consider each specific case). Denote $\overline{X} = E[X]$ and $\overline{Y} = E[Y]$. We set $\tau = (1+\delta)\overline{X} = (1-\delta)\overline{Y}$, where $0 < \delta \leq 1$. From this, we obtain that $\delta = \frac{\overline{Y}-\overline{X}}{\overline{Y}+\overline{X}}$ and $\tau = \frac{2\overline{X}\overline{Y}}{\overline{Y}+\overline{X}}$. Note that $\overline{Y} > \overline{X}$, and hence $0 < \delta \leq 1$. Then we can use Theorem 4.4 of [25] and obtain that $\Pr[(X \geq \tau) \vee (Y < \tau)] \leq \Pr[X \geq \tau] + \Pr[Y < \tau] = \Pr\left[X \geq (1+\delta)\overline{X}\right] + \Pr\left[Y < (1-\delta)\overline{Y}\right] \leq e^{\frac{-\overline{X}\delta^2}{3}} + e^{\frac{-\overline{Y}\delta^2}{2}} \leq 2e^{\frac{-\overline{X}\delta^2}{3}}$, where the last inequality also follows from the fact that $\overline{Y} > \overline{X}$.

Then, to bound the probability of non-success by ε, we force $2e^{\frac{-\overline{X}\delta^2}{3}} \leq \varepsilon$, which yields

$$\left(\frac{\overline{Y}-\overline{X}}{\overline{Y}+\overline{X}}\right)^2 \overline{X} \geq 3(\ln 2 - \ln \varepsilon). \tag{1}$$

We now show the results for case (a) and (b) by defining appropriate Bernoulli variables and replacing the values of $\overline{X}$ and $\overline{Y}$ on Eq. (1). We need to define a different set of Bernoulli random variables for each case in order to ensure their independence.

Case (a): *parameter* $\boldsymbol{q}$. We define the following Bernoulli random variables. For each $i \in P$, the Bernoulli random variable $X_i^{(\mathrm{a})} = 1$ if and only if, simultaneously, processor i is faulty, chosen (i.e., $i \in S$), and its (incorrect) reply reaches M on time. Similarly, for each $j \in P$, $Y_j^{(\mathrm{a})} = 1$ if and only if, simultaneously, processor j is correct, chosen, and its (correct) reply reaches M on time. It is easy to verify that $X = \sum_{i \in P} X_i^{(\mathrm{a})}$ and $Y = \sum_{j \in P} Y_j^{(\mathrm{a})}$, for the random variables X and Y, defined above. In this case we have that $\Pr[X_i^{(\mathrm{a})} = 1] = pqd$ and $\Pr[Y_j^{(\mathrm{a})} = 1] = (1-p)qd$, for any i and j. Then, $\overline{X} = npqd$ and $\overline{Y} = n(1-p)qd$. Plugging these values in Eq. (1), we obtain the stated result for case (a).

Case (b): *parameter* $\boldsymbol{s}$. For this case, we only define Bernoulli random variables for the processors in S. Then, for each $i \in S$, the variable $X_i^{(\mathrm{c})} = 1$ if and only if, simultaneously, processor i is faulty and its reply reaches M on time. Similarly, for each $j \in S$, the variable $Y_j^{(\mathrm{c})} = 1$ if and only if, simultaneously, processor j is correct and its reply reaches M on time. Again, it is easy to verify that $X = \sum_{i \in S} X_i^{(\mathrm{c})}$ and $Y = \sum_{j \in S} Y_j^{(\mathrm{c})}$. In this case we have that $\Pr[X_i^{(\mathrm{c})} = 1] = pd$ and $\Pr[Y_j^{(\mathrm{c})} = 1] = (1-p)d$, for any $i, j \in S$. Then, $\overline{X} = spd$ and $\overline{Y} = s(1-p)d$. Plugging these values in Eq. (1), we obtain the stated result for case (b).

Finally, using basic calculus (derivatives) it is easily shown that the equations for (expected) work are minimized when $p = 1/6$. This completes the proof. □

4.5. *Comparison of the Algorithms*

Both algorithms obtain the same probability of success $1 - \varepsilon$ and we derive similar upper bounds on the work required in doing so. In particular, the bounds for both algorithms only differ on a factor of $6 \cdot p^2$, that is, algorithm MBA requires less work than algorithm TBA if $p < 1/\sqrt{6}$ (≈ 0.4). Therefore, for the cases where p is a constant, both algorithms achieve the same asymptotic upper bounds on work, which are asymptotically optimal with respect to the lower bounds obtained in this work; in this case the work complexity is $\Theta((-\lg \varepsilon)/d)$.

Additionally, note that when using the parameter s in the algorithm MBA, it is enough to receive $\lfloor s/2 \rfloor + 1$ equal replies to safely decide the corresponding value. Otherwise, majority is used to decide. Since this behavior is exactly the same that would be observed if the TBA algorithm is used with the same value of s and defining $\tau = \lfloor s/2 \rfloor + 1$, we can safely use TBA instead of MBA with the same parameter s and obtain the bound of work of case (b) of Theorem 3.

Given the above discussion, and *based on the analysis we obtained*, we consider more preferable to use parameter s instead of q, since the former allows to know the exact value of work that will be incurred. Additionally, from an implementation point of view, the use of s only requires to make s random choices to obtain the set of chosen workers, while using q implies making a linear number of random choices (one per worker). Having chosen to use parameter s, we consider algorithm TBA to be more preferable than algorithm MBA, since TBA is *early-terminating* (discussed in the introduction of Section 4) as opposed to MBA that always waits for time T and then makes a decision on the value to accept. As we previously argued, even when our analyses yield that TBA requires more work than MBA, we can still use TBA with the parameter s of MBA and $\tau = \lfloor s/2 \rfloor + 1$, and reach the smaller work bound.

Figure 4 shows graphical comparisons of the two algorithms and the lower bounds we obtain by plugging certain values on the analytical expression we have derived. Additionally, it presents the minimum value of s that would satisfy the desired success probability for MBA, obtained by simulation (called optimal in the figure). These simulations have been carried out by generating 100 million random instances for each value of s and each value of p, and counting, in each case, the proportion (probability) of cases with a majority of incorrect answers. With this, the evaluation of the optimal s for each p is straightforward. From the left plot it can be observed how the work of MBA is below that of TBA when p is smaller than $1/\sqrt{6}$ (≈ 0.4), that they match at this point, and it is above that of TBA for larger values of p. From the right plot it can be observed the similar behavior of our upper and lower bound results as ε changes.

5. Future Work

Several research directions emanate from our work. First, it would be interesting to investigate whether the gap (that exists in some cases) between our lower and upper bounds could be decreased; to this respect it seems that both the lower bound and

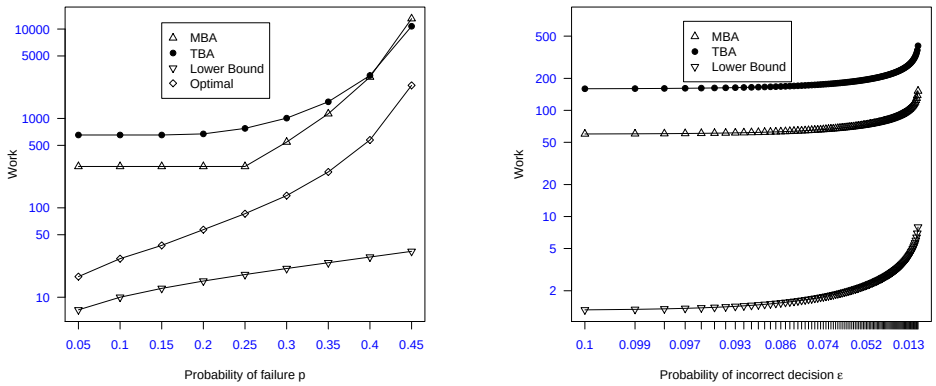

Fig. 4. Comparison of the bounds obtained when parameters p and s are considered. The plot on the left depicts the work s (y-axis) over p (x-axis), for $n = 10^6$, $d = 0.9$ and $\varepsilon = 1/n$. The plot on the right depicts (in log scale) the work s (y-axis) over ε (x-axis), for $d = 0.9$ and $p = 1/4$.

the algorithm analyses could improve. Especially for the algorithms' analyses, we believe that if we are able to use less pessimistic assumptions or to avoid the use of Chernoff bounds (or perhaps use/devise a more appropriate Chernoff bound for our problem) then we should be able to improve the bounds on work while maintaining the same probability of success.

One important contribution of this work is the introduction of the parameter d, that models the probability of a worker not replying. Studying a model in which these events are not stochastic, but controlled by an adversary (like, e.g., in [26]), seems interesting. Another interesting research direction is to relax the one-round assumption of our model (which was used to guarantee fast termination of algorithms) and allow for M to decide in more than one round. For instance, M could start a second round if it did not receive enough replies in the first round. Intuitively, in such a case, M should be able to obtain better probability of success or perhaps less expected work. This gives rise to the following question: By how much is the probability of success increased and how are algorithm termination and the bounds on work affected?

Another direction to explore further is to consider the more general problem in which there is a sequence of tasks whose values M must reliably obtain, while maintaining the overall work low, as done in [28] and [20]. To this respect, possibly a mechanism for identifying or suspecting workers as malicious needs to be devised. Care has to be taken when choosing this mechanism since, as shown by Kodo et al. [19], very often workers may return incorrect results without being frequent offenders.

Finally, recall that in order to analyze worst case scenarios we considered full collusion among faulty workers. It would be also interesting to study what happens in the case where faulty workers do not collaborate or their collaboration is restricted.

References

[1] I. Abraham, D. Dolev, R. Goden, and J.Y. Halpern. Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In *Proc. of PODC 2006*, pages 53–62.

[2] N. Alon and J.H. Spencer. *The Probabilistic Method.* J. Wiley and Sons, 2^{nd} Ed., 2000.

[3] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *Proc. of GRID 2004*, pages 4–10.

[4] D. P. Anderson, E. Korpela, and R. Walton. High-performance task distribution for volunteer computing. In *Proc. of e-Science 2005*, pages 196–203.

[5] D. Blough and G. Sullivan. A comparison for voting strategies for fault-tolerant distributed systems. In *Proc. of SRDS 1990*, pages 136–145.

[6] C. Dwork, J. Halpern, and O. Waarts. Performing work efficiently in the presence of faults. *Siam Journal on Computing*, 27(5):1457–1491, 1998.

[7] W. S. Evans and N. Pippenger. Average-case lower bounds for noisy Boolean decision trees. *SIAM Journal on Computing,* 28(2):433–446, 1998.

[8] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.

[9] A. Fernández Anta, C. Georgiou, and M. A. Mosteiro. Cost-sensitive Mechanisms for Internet-based Computing. In *Proc. of NCA 2008*, pages 315–324.

[10] A. Fernández Anta, C. Georgiou, and M. A. Mosteiro. Algorithmic Mechanisms for Internet-based Master-Worker Computing with Untrusted and Selfish Workers. In *Proc. of IPDPS 2010.*

[11] A. Fernández, C. Georgiou, A. Russell, and A. Shvartsman. The Do-All problem with Byzantine processor failures. *Theoretical Computer Science*, 333(3):433–454, 2005.

[12] A. Fernández, C. Georgiou, L. López, and A. Santos. Reliably Executing Tasks in the Presence of Untrusted Entities. In *Proc. of SRDS 2006*, pages 39-50.

[13] P. Gács and A. Gál. Lower bounds for the complexity of reliable Boolean circuits with noisy gates. *IEEE Transactions on Information Theory,* 40(2):579–583, 1994.

[14] R. G. Gallanger. Finding parity in simple broadcast networks. *IEEE Transactions on Information Theory,* 34:176–180, 1988.

[15] Ch. Georgiou and A.A. Shvartsman. *Do-All Computing in Distributed Systems: Cooperation in the Presence of Adversity.* Springer, 2008.

[16] N. Goyal, G. Kindler, and M. Saks. Lower bounds for the noisy broadcast problem. In *Proc. of FOCS 2005*, pages 40–52.

[17] E. M. Heien, D. P. Anderson, and K. Hagihara. Computing Low Latency Batches with Unreliable Workers in Volunteer Computing Environments. *J. Grid Comput.,* 7:501-518, 2009.

[18] C. Kenyon and V. King. On Boolean decision trees with faulty nodes. *Random Structures and Algorithms,* 5(3):453–464, 1994.

[19] D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. Silva. G. Fedak, and F. Cappello. Characterizing Result Errors in Internet Desktop Grids. In *Proc. of Euro-Par 2007*, pages 361–371.

[20] K. M. Konwar, S. Rajasekaran, and A. A. Shvartsman. Robust Network Supercomputing with Malicious Processes. In *Proc. of DISC 2006*, pages 474–488.

[21] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. SETI@home: Massively distributed computing for SETI. *Computing in Science and Engineering*, 3(1):78–83, 2001.

[22] A. Kumar and K. Malik. Voting mechanisms in distributed systems. *IEEE Transactions on Reliability,* 40(5):593–600, 1991.

[23] E. Kushilevitz and Y. Mansour. Computation in noisy radio networks. *SIAM Journal*

on Discrete Mathematics, 19(1):96–108, 2005

[24] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[25] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[26] M. Paquette and A. Pelc. Optimal decision strategies in Byzantine environments. *Journal of Parallel and Distributed Computing*, 66(3):419-427, 2006.

[27] R. Reischuk and B. Schmeltz. Reliable computation with noisy circuits and decision trees – A general $n \log n$ lower bound. In *Proc. of FOCS 1991*, pages 602–611.

[28] L. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Computer Systems*, 18(4):561–572, 2002.

[29] M. Szegedy and X. Chen. Computing Boolean functions from multiple faulty copies of input bits. *Theoretical Computer Science,* 321(1):149–170, 2004.

[30] M. Yurkewych, B.N. Levine, and A.L. Rosenberg. On the cost-ineffectiveness of redundancy in commercial P2P computing. In *Proc. of CCS 2005*, pp. 280–288.