Reliably Executing Tasks in the Presence of Untrusted Entities[*]

Antonio Fernández [†]
LADyR, GSyC,
Universidad Rey Juan Carlos,
28933 Móstoles, Spain.
anto@gsyc.escet.urjc.es.

Chryssis Georgiou [‡]
Dept. of Computer Science,
University of Cyprus,
75 Kallipoleos Str.,
P.O. Box 20537, CY-1678,
Nicosia, Cyprus.
chryssis@ucy.ac.cy

Luis López [*]
llopez@gsyc.escet.urjc.es.

Agustín Santos [*]
asantos@gsyc.escet.urjc.es.

Abstract

In this work we consider a distributed system formed by a master processor and a collection of n processors (workers) that can execute tasks; worker processors are untrusted and might act maliciously. The master assigns tasks to workers to be executed. Each task returns a binary value, and we want 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 goal is to have the master computer 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 explore two ways of bounding the number of faulty processors: (a) we consider a fixed bound $f < n/2$ on the maximum number of workers that may fail, and (b) a probability $p < 1/2$ of any processor to be faulty (all processors are faulty with probability p, independently of the rest of processors).

Our work demonstrates that it is possible *to obtain high probability of correct acceptance with low work. In particular, by considering both mechanisms of bounding the number of malicious workers, 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.*

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 cooperative computation is the SETI@home project [17]. 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. 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 (powerful) processors, called *workers*, that can execute tasks; worker processors might act maliciously, that is, they are Byzantine [20]. Since each task returns a value, we want the master to accept only correct values with high probability. Namely, if $\varepsilon \ll 1$ is the probability of accepting an incorrect value, 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

[*]This work is supported in part by the IST Program of the European Union under contract number IST-2005-015964 (AEOLUS). Preliminary results of this work have been announced in [9].

[†]Partially supported by the Spanish MEC under grant TIN2005-09198-C02-01 and the Comunidad de Madrid under grant S-0505/TIC/0285.

[‡]Partially supported by funds at the University of Cyprus.

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, processors can be slow, 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 two types of known bounds on the number of malicious workers: we either consider a fixed bound $f < n/2$ on the maximum number of workers that may fail, or a probability $p < 1/2$ of any processor to be faulty (f and p may depend on n). Given the above model, and considering a single task (which returns a binary value) assigned to several workers, our goal is to have the master computer to accept 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 M assigned the task). (The problem and model are presented in detail in Section 2.)

Observe that a trivial solution to the above problem when $d = 1$ (all messages are delivered on time) and there are no more than $f < n/2$ malicious workers is to have M assign the task to $2f + 1$ workers. This guarantees that the correct value is accepted (with probability 1). Note, however, that if $f = \Theta(n)$, then the work is linear on n (which is not desired). Furthermore, if $d < 1$, there are less than $2f + 1$ workers available to execute the task, or we consider a probabilistic model of failures (each processor is faulty with probability $p < 1/2$), then it is not so obvious how to fully guarantee that a correct value is accepted with high probability. In this work, we develop two non-trivial algorithms for this problem and we show that it is in fact possible to obtain high probability of success with low work (for example, in the above case of $d = 1$ and linear f, if $\varepsilon = 1/n$, processor M accepts the correct value with probability at least $1 - 1/n$ and with work *logarithmic* on n instead of linear). Furthermore, we provide lower bound results on the work required to achieve high probability of success.

Prior/Related Work. The problem we consider in this work is clearly related to the *voting problems* (e.g., [4, 21, 18]). 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 (implicitly they assume that proposing a value involves no cost), and only the probability of a bad choice has to be minimized. In our model this probability is chosen a priori and the cost, measured as the number of entities involved, is minimized.

Additionally, differences exist between the models considered and our model. For instance, both Blough and Sullivan [4] and Paquette and Pelc [21] assume that the values proposed by non-faulty voters are always received by the deciding agent, and that there is a priori knowledge of the probability of each possible value to be correct. Also, in [4] it is assumed a priori knowledge of the probability for a faulty entity to propose each possible value (faulty entities are not really Byzantine). In [21], Byzantine failures are considered and the authors are concerned with the computational cost of the strategy, proposing strategies with linear cost on the number of voters.

To our knowledge, the work on voting closest to our model is that of Kumar and Malik [18], since they define a reliability level that has to be achieved and try to minimize the cost of achieving it. However, they still assume that the deciding agent gets proposals from all the entities. More importantly, they assume that each entity has associated a cost versus reliability curve that defines the cost that has to be invested in that entity in order to have a given probability of the entity proposing the correct value. Then, under this model the strategies are able to tune the failure probability of each voter to optimize the total cost. In our model, the failure probability is given, the master gets to choose how many entities are asked to propose, and the cost is the number of entities chosen.

A real system that is very related to the model presented in this paper is the Berkeley Open Infrastructure for Network Computing (BOINC) [2, 3]. This system allows volunteers to provide free computational cycles to perform intensive computation in a form similar to the one proposed in this paper. In fact, the SETI@home project now runs over BOINC. 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 application dependent: the application has to provide the number of instances, a function to compare the received results, a function to validate, and the minimum number of received results in order to start validating. Once this latter minimum is reached, the validation process is invoked with the set of received responses, after each new response is received, until some value is accepted. If an instance does not respond by some given time another instance is started.

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. The theoretical model and algorithms proposed in this paper could be adapted by BOINC designers to incorporate the validation mechanism as part of the system, and letting the applications simply ask for a certain level of reliability. The results of this paper could be used to derive the number of instances that have to be started for a task, and the validation strategy to be used.

Another problem related to the problem we consider in this work is the *Do-All* problem, in which a collection of k processors need to cooperatively perform t independent tasks in the presence of failures (e.g., [5, 14, 15, 12]). Recently, this problem was studied under Byzantine processors [8]. 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 the above *Do-All* paper, 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).

Finally, 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., [22, 10]), noisy Boolean decisions trees (e.g., [22, 16, 7, 6]), and noisy broadcast (e.g., [11, 13, 19]). 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 trivial 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 at all or a message is not received on time). Recent work [23] 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 trivial 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. We study an interesting variation of the voting problem under a model that captures realistic systems of distributed computation. To the best of our knowledge, the problem and model as presented here have not been studied in prior work. Our work demonstrates that *it is possible* to execute tasks reliably in the presence of malicious processors with high probability and with low cost. In particular,

- We present lower bounds on work, considering both mechanisms of bounding the number of malicious workers (maximum number of malicious workers $f < n/2$, probability of each worker failing $p < 1/2$). Particularly, 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* [22] required for the reliable computation of the 1-variable trivial 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) which is a simple and natural algorithm where M decides on the majority of received responses, and (b) the *Threshold Based Algorithm* (TBA) in which 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 either p or f. Furthermore, for the cases where p is a constant or f is linear 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. The rest of the paper is organized as follows. 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 show that they achieve the desired probability of correct decision while maintaining low work. Finally, in Section 5 we discuss and compare our two algorithms and the lower bounds and identify interesting future research directions.

2 Model and Definitions

We study execution of tasks in a system in which the processors can behave maliciously, i.e., are Byzantine [20]. We assume there is a fail-free *master* processor M which has a task to be executed. This task returns a binary value, which M wants to reliably obtain. Processor M is not capable of executing the task itself, so a set P of n (powerful) processors, $P = \{1, ..., n\}$, that can execute the task, is made available to 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).

The 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 two kinds of mechanisms to bound the number of malicious processors. We either assume that (i) there is a fixed bound $f < n/2$ on the maximum number of processors that fail, or (ii) 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 f or 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 non-decreasing 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. In this work we consider only cases where $V = \{0, 1\}$. Note that since M decides in one round, it makes sense to assume that faulty workers prefer to reply to M with an incorrect value rather than to choose not to reply at all (of course their message might be lost or delayed). During the rest of the paper we will assume that this is always the case: a faulty worker always replies with the incorrect value. The faulty workers can obtain the incorrect value either by collaborating, or by simply computing the task (if v is the result, then they respond with $1 - v$).

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. Since we have two different ways to characterize processor failures, p and f, we have different bounds for each case.

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.

```
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 $v$
```

Figure 1. Algorithm executed by any worker processor.

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.*

Proof: Due to lack of space, this proof is given in the Appendix. ■

Then, for the lower bounds we only need to consider algorithms that accept the most replied value. The following theorem, for the case when workers fail with probability p, shows that any algorithm must have runs in which the same task is assigned to a minimum number of workers.

Theorem 3.2 *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

$$\sum_{c > r/2} \binom{r}{c} p^c (1-p)^{r-c} \geq p^{\frac{r}{2}+1} > p^{\log_p \varepsilon} = \varepsilon.$$

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 completes the proof. ■

The above Theorem leads to a new non-trivial lower bound result on the depth of *noisy static Boolean decision trees* [22].

Corollary 3.3 *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 following theorem presents a similar lower bound for the case when at most f workers can fail. For this bound to hold we need f to be large enough.

Theorem 3.4 *If $f > -\lg \varepsilon$ workers fail, for any d, any algorithm must have runs in which it assigns the task to more than $2\frac{\lg \varepsilon}{\lg \frac{f + \lg \varepsilon}{n}} - 2$ 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 no more than $2\frac{\lg \varepsilon}{\lg \frac{f+\lg \varepsilon}{n}} - 2$ workers. This implies that M gets $r \leq 2\frac{\lg \varepsilon}{\lg \frac{f+\lg \varepsilon}{n}} - 2 = 2\log_{\frac{f+\lg \varepsilon}{n}} \varepsilon - 2$ replies. Then, since M did not have knowledge of which processors are faulty when assigning the task, and all selected processors have the same probability of getting their replies through, the probability that a majority of the replies come from faulty processors is at least

$$\frac{f(f-1)\cdots(f-\lfloor r/2 \rfloor)}{n(n-1)\cdots(n-\lfloor r/2 \rfloor)} \geq \left(\frac{f - r/2}{n}\right)^{\frac{r}{2}+1} > \left(\frac{f + \lg \varepsilon}{n}\right)^{\frac{r}{2}+1} \geq \left(\frac{f + \lg \varepsilon}{n}\right)^{\log_{\frac{f+\lg \varepsilon}{n}} \varepsilon} = \varepsilon,$$

where the second inequality follows from the fact that $\lg \varepsilon < 0$, which implies that $(f + \lg \varepsilon)/n < f/n < 1/2$. Then, $\lg \frac{f+\lg \varepsilon}{n} < -1$ and hence $r/2 < -\lg \varepsilon$.

Then, if this happens, a majority of the replies will 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 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 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 3.5 *If $\varepsilon \leq 1/2$ and workers fail with probability p, 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,

$$\Pr\left[(R<2\overline{R})\right]\Pr\left[X>R/2|R<2\overline{R}\right]\geq$$
$$\frac{1}{2}\Pr\left[X=\lfloor R/2\rfloor+1|R<2\overline{R}\right]\geq$$
$$\frac{1}{2}p^{\lfloor R/2\rfloor+1}>\frac{1}{2}p^{\overline{R}+1}\geq\varepsilon.$$

Then, M will decide incorrectly if this happens, and hence the success probability of the majority algorithm is smaller than $1-\varepsilon$. This and Lemma 3.1 complete the proof. ■

Theorem 3.6 *If $f>-\lg(2\varepsilon)$ workers fail and $\varepsilon\leq 1/2$, the expected number of workers to which any algorithm assigns a task must be more than $\frac{1}{d}(\frac{\lg(2\varepsilon)}{\lg\frac{f+\lg(2\varepsilon)}{n}}-1)$ 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 $\frac{1}{d}(\frac{\lg(2\varepsilon)}{\lg\frac{f+\lg(2\varepsilon)}{n}}-1)$ workers. This implies that M gets on average $\overline{R}\leq d(\frac{1}{d}(\frac{\lg(2\varepsilon)}{\lg\frac{f+\lg(2\varepsilon)}{n}}-1))=\log_{\frac{f+\lg(2\varepsilon)}{n}}(2\varepsilon)-1$ replies. Let R and X be random variables as defined in the proof of the previous theorem, again we have that $\Pr\left[R\geq 2\overline{R}\right]\leq 1/2$. Then, we have that

$$\Pr\left[(R<2\overline{R})\right]\Pr\left[X>R/2|R<2\overline{R}\right]\geq$$
$$\frac{1}{2}\left(\frac{f-R/2}{n}\right)^{\lfloor\frac{R}{2}\rfloor+1}>\frac{1}{2}\left(\frac{f+\lg(2\varepsilon)}{n}\right)^{\lfloor\frac{R}{2}\rfloor+1}>$$
$$\frac{1}{2}\left(\frac{f+\lg(2\varepsilon)}{n}\right)^{\overline{R}+1}\geq\frac{1}{2}\left(\frac{f+\lg(2\varepsilon)}{n}\right)^{\log_{\frac{f+\lg(2\varepsilon)}{n}}(2\varepsilon)}=\varepsilon,$$

where the second inequality follows from the fact that $\lg(2\varepsilon)\leq 0$, which implies that $(f+\lg(2\varepsilon))/n\leq f/n<1/2$. Then, $\lg\frac{f+\lg(2\varepsilon)}{n}<-1$ and hence $R/2<\overline{R}\leq\frac{\lg(2\varepsilon)}{\lg\frac{f+\lg(2\varepsilon)}{n}}-1<-\lg(2\varepsilon)$. Then, M will decide incorrectly if this happens, and hence the success probability of the majority algorithm is smaller than $1-\varepsilon$. This and Lemma 3.1 complete the proof. ■

Note that the above lower bounds do not restrict the assignments nor the decision policy of the algorithm. Furthermore, the workers assigned to the same task can be so at different times. In all cases, the bounds give the total number of workers that must be assigned to a task until accepting a value.

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 Algorithm* (MBA for short) is a simple and natural algorithm where M decides on the majority of received responses. In the second algorithm, called *Threshold Based Algorithm* (TBA for short), 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.

The exact analyses of the algorithms 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. The Chernoff bounds we choose to use for the analyses of the two algorithms are the following:

Lemma 4.1 ([1]) *Let $Z_1,Z_2,...,Z_n$ be n independent Bernoulli distributed random variables with $Pr[Z_i=1]=p_i$ and $Pr[Z_i=0]=1-p_i$, then it holds for $Z=\sum_{i=1}^n Z_i$ and $\mu=E[Z]=\sum_{i=1}^n p_i$ that*

(α) *$Pr[Z\geq(1+\delta)\mu]\leq e^{\frac{-\mu\delta^2}{3}}$ for all $0<\delta\leq 1$, and*

(β) *$Pr[Z<(1-\delta)\mu]\leq e^{\frac{-\mu\delta^2}{2}}$ for all $0<\delta\leq 1$.*

4.1 The Majority Based Algorithm

We first present and analyze 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.

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

Theorem 4.2 *Algorithm MBA guarantees a success probability of at least $1-\varepsilon$ with*

```
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
```

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

(a) Expected Work $E[|S|] = nq = \frac{18(\ln 2 - \ln \varepsilon)p}{(1-2p)^2 d}$ when parameters p and q are considered,

(b) Expected Work $E[|S|] = nq = \frac{18(\ln 2 - \ln \varepsilon)f/n}{(1-2f/n)^2 d}$ when parameters f and q are considered,

(c) Work $|S| = s = \lceil \frac{18(\ln 2 - \ln \varepsilon)p}{(1-2p)^2 d} \rceil$ when parameters p and s are considered, and

(d) Work $|S| = s = \lceil \frac{18(\ln 2 - \ln \varepsilon)f/n}{(1-2f/n)^2 d} \rceil$ when parameters f and s are considered.

For $p < 1/4$ and $f < n/4$ the values for $p = 1/4$ and $f = n/4$ have to be used, respectively.

Proof: We denote by X the random variable that accounts for the number of replies that 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). 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.

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

We now treat each term separately. The first term can be bounded with the Chernoff bound of Lemma 4.1(β), fixing $\delta = 1/3$.

$$\sum_{c < 2\overline{R}/3} \Pr[R = c] = \Pr\left[R < 2\overline{R}/3\right] \leq e^{-\overline{R}/18}.$$

To bound the second term we need the following claim, which trivially follows from $p < 1/2$ and $f < n/2$, whichever the case.

Claim: Let c and c' be two non-negative integers. If $c \leq c'$ then $\Pr[X \geq c/2 | R = c] \geq \Pr[X \geq c'/2 | R = c']$.

From the Claim we have 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 want to use now a Chernoff bound to bound this probability. Let S be the set of chosen processors and F be the set of faulty processors. If parameter p is used, we define, for each $i \in S$, the Bernoulli random variable $X_i^{(1)} = 1$ if and only if processor i is faulty and its reply reaches M on time. Then it is easy to verify that $X = \sum_{i \in S} X_i^{(1)}$ and that these variables are independent. If parameter f is used, we define, for each $i \in F \cap S$, the Bernoulli random variable $X_i^{(2)} = 1$ if and only if processor i's reply reaches M on time. It is also easy to verify that $X = \sum_{i \in F \cap S} X_i^{(2)}$ and that these variables are independent. Clearly, if $R = 2\overline{R}/3$ then the expected value of X is $\overline{X} = E[X] = \varphi 2\overline{R}/3$, where φ is either p or f/n. Then, we can apply Lemma 4.1(α) with $\delta = \frac{1}{2\varphi} - 1$ as long as $1/4 \leq \varphi < 1/2$ (to guarantee $0 < \delta \leq 1$), and obtain that $\Pr\left[X \geq \overline{R}/3 | R = 2\overline{R}/3\right] \leq e^{-\frac{(1-2\varphi)^2 \overline{R}}{18\varphi}}$. Now, since $\frac{(1-2\varphi)^2 \overline{R}}{18\varphi} \leq \overline{R}/18$, we can add both bounds and obtain that $\Pr[X \geq Y] \leq 2e^{-\frac{(1-2\varphi)^2 \overline{R}}{18\varphi}}$. In order to keep this value no larger than ε, it is enough to guarantee that $\overline{R} \geq \frac{18\varphi(\ln 2 - \ln \varepsilon)}{(1-2\varphi)^2}$. Since either $\overline{R} = sd$ or $\overline{R} = nqd$ and either $\varphi = p$ or $\varphi = f/n$, the four cases of the statement of the theorem hold. ■

4.2 The Threshold Based Algorithm

We now present and analyze 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 other-

```
**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 for replies from the workers in $S$
4  If there are $\tau$ 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
```

Figure 3. Threshold based algorithm executed by processor M.

wise M will not get enough replies from 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).

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

Theorem 4.3 *Algorithm TBA guarantees a success probability of at least $1-\varepsilon$ with*

(a) Expected Work $E[|S|] = nq = \frac{3(\ln 2 - \ln\varepsilon)}{(1-2p)^2pd}$ when parameters p and q are considered,

(b) Expected Work $E[|S|] = nq = \frac{3(\ln 2 - \ln\varepsilon)}{(1-2f/n)^2(f/n)d}$ when parameters f and q are considered,

(c) Work $|S| = s = \lceil\frac{3(\ln 2 - \ln\varepsilon)}{(1-2p)^2pd}\rceil$ when parameters p and s are considered, and

(d) Work $|S| = s = \lceil\frac{3(\ln 2 - \ln\varepsilon)}{(1-2f/n)^2(f/n)d}\rceil$ when parameters f and s are considered.

Moreover, for $p < 1/6$ and $f < n/6$ the values for $p = 1/6$ and $f = n/6$ are used, respectively.

Proof: We first present a general analysis that is independent of the specific parameters considered (p or f and q or s) and then we derive the results for each case following the general analysis. To simplify the analysis 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 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.

We define the random variables X and Y as in the proof of Theorem 4.2. 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 the Chernoff bounds given in Lemma 4.1. 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 Lemma 4.1 and obtain the following

$$\begin{gathered}\Pr[(X \geq \tau) \vee (Y < \tau)] \leq \\ \Pr[X \geq \tau] + \Pr[Y < \tau] = \\ \Pr\left[X \geq (1+\delta)\overline{X}\right] + \Pr\left[Y < (1-\delta)\overline{Y}\right] \leq \\ e^{\frac{-\overline{X}\delta^2}{3}} + e^{\frac{-\overline{Y}\delta^2}{2}} \leq 2e^{\frac{-\overline{X}\delta^2}{3}},\end{gathered}$$

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

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

We now show the results for each case (a)-(d) 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): *parameters p and 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): *parameters f and q*. Denote by F the set of faulty processors. Then we define the following Bernoulli random variables. For each $i \in F$, the variable $X_i^{(\mathrm{b})} = 1$ if and only if, simultaneously, processor i is chosen and its reply reaches M on time. Similarly, for each $j \in P \setminus F$, $Y_j^{(\mathrm{b})} = 1$ if and only if, simultaneously, processor j is chosen and its reply reaches M on time.

Again, $X = \sum_{i \in F} X_i^{(\mathrm{b})}$ and $Y = \sum_{j \in P \setminus F} Y_j^{(\mathrm{b})}$. Then, $\Pr[X_i^{(\mathrm{b})} = 1] = \Pr[Y_j^{(\mathrm{b})} = 1] = qd$, for any $i \in F$ and $j \in P \setminus F$, and $\overline{X} = fqd$ and $\overline{Y} = (n - f)qd$. From Eq. (1), we obtain the stated result for case (b).

Case (c): *parameters p and 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 (c).

Case (d): *parameters f and s.* Finally, we define the following Bernoulli random variables. For each $i \in F \cap S$, $X_i^{(\mathrm{d})} = 1$ if and only if processor i's reply reaches M on time. Similarly, for each $j \in (P \setminus F) \cap S$, $Y_j^{(\mathrm{d})} = 1$ if and only if processor j's reply reaches M on time. Observe once again that $X = \sum_{i \in F \cap S} X_i^{(\mathrm{d})}$ and $Y = \sum_{j \in (P \setminus F) \cap S} Y_j^{(\mathrm{d})}$. Then, $\Pr[X_i^{(\mathrm{d})} = 1] = \Pr[Y_j^{(\mathrm{d})} = 1] = d$, for any $i \in F \cap S$ and $j \in (P \setminus F) \cap S$, and $\overline{X} = fsd/n$ and $\overline{Y} = (n - f)sd/n$. From Eq. (1), we obtain the stated result for case (d).

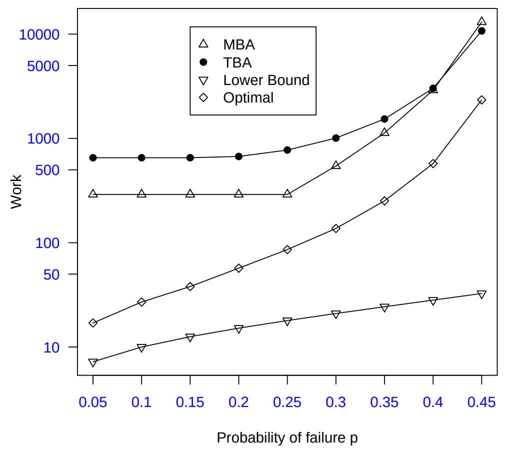
Finally, using basic calculus (derivatives) it is easily shown that the equations for (expected) work for cases (a) to (d) are minimized when $p = 1/6$ or $f = n/6$. This completes the proof of the theorem. ■

5 Discussion

In this work we consider the problem of executing tasks reliably in the presence of untrustworthy processors that may act maliciously. We consider a model of a distributed system with a master processor M and a set of n untrustworthy workers. Processor M must assign a task to a subset of the workers so that the probability of accepting the correct value of the task is high, while the amount of work (number of workers assigned the task) required in doing so is minimized. We bound the potential number of faulty workers in two ways: either by considering a fixed bound $f < n/2$ on the maximum number of processors that fail, or by considering a probability $p < 1/2$ of any processor to be faulty (all processors are faulty with probability p, independently of the rest of processors). Additionally we allow for unreliable and slow communications, including in the model the probability d that the reply from a worker reaches M on time.

We first present lower bounds on work for this model, considering both mechanisms of bounding the number of malicious workers. Particularly, we identify lower bounds on the minimum amount of work required, so that any algorithm accepts the correct value with probability of success $1 - \varepsilon$, where $\varepsilon \ll 1$. Then, we develop and analyze two algorithms: (a) algorithm MBA which is a simple and natural algorithm where M decides on the majority of received responses, and (b) algorithm TBA in which 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 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 $6 \cdot p^2$ or $(6 \cdot f/n)^2$ (depending on the mode of failures considered), that is, algorithm MBA requires less work than algorithm TBA if $p < 1/\sqrt{6}$ or $f < n/\sqrt{6}$. Therefore, for the cases where p is a constant or f is linear 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 $\Theta((-\lg \varepsilon)/d)$. Given the above discussion, and *based on the analysis we obtained*, we consider algorithm TBA to be most preferable than algorithm MBA, since TBA is *early-terminating* (discussed in Section 4) as opposed to MBA that always waits for time T and then makes a decision on the value to accept.

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. From the left plot it can be observed how the work of MBA is below that of TBA when p is smaller than $1/\sqrt{6}$, 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. As previously discussed in the paper, performing an exact analysis of our algorithms proved to be very difficult, mainly due to the complicated derived formulas that do not allow us to express the important parameters of the model and algorithms in a meaningful way. Therefore, we have chosen to analyze our algorithms using Chernoff bounds, which has enabled us to obtain closed-form equations for these parameters. Additionally, to simplify the analyses we had to make pessimistic assumptions. Not surprisingly, it appears that in some cases there is a big gap between our upper and lower bound results. We believe that the gap can be decreased by improving both the analyses of the lower bounds and of the algorithms. 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



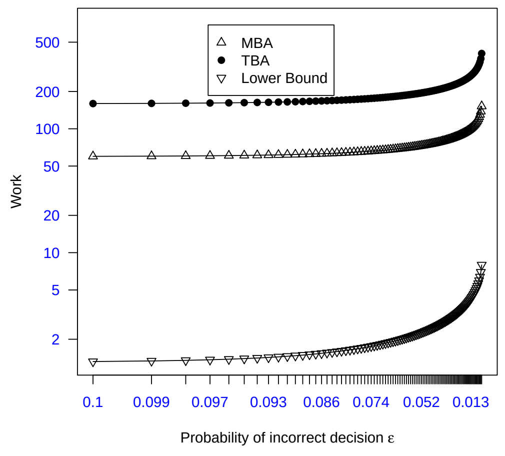


Figure 4. Comparison of the bounds obtained when parameters p and s are considered. The plot on the left depicts the work s (y-axis) over p (x-axis), for $n = 10^6,\ d = 0.9$ and $\varepsilon = 1/n$. The plot on the right depicts (in log scale) the work s (y-axis) over ε (x-axis), for $d = 0.9$ and $p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

of success.

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 is to consider the more general problem where there is a sequence of tasks whose values M must reliably obtain while maintaining the overall work (required for all tasks) low. Our current algorithms provide trivial bounds on work for this model (work as computed in this paper times the number of tasks that must be executed) with the same probability of success for each task execution. These trivial bounds are possibly too loose and one could improve them by taking into account the possibility of avoiding re-using identified faulty processors in the upcoming task executions. For that a mechanism for identifying or suspecting workers as faulty needs to be devised.

Finally, an interesting extension to this work would be to consider the situation where it is possible for a task to return more than two values (that is, $|V| > 2$). In the case where the faulty processors can collaborate and agree on the incorrect values they will return to M, then our analysis trivially holds. However it would be very interesting to study what happens in the case where faulty processors do not collaborate or their collaboration is restricted. Ongoing work is underway toward this direction.

References

[1] N. Alon and J.H. Spencer. *The Probabilistic Method.* J. Wiley and Sons, Inc., Second Edition, 2000.

[2] David P. Anderson. BOINC: A system for public-resource computing and storage. In *Proceedings of the* 5^{th} *IEEE/ACM International Workshop on Grid Computing (GRID 2004)*, pages 4–10, 2004.

[3] David P. Anderson, Eric Korpela, Rom Walton. High-performance task distribution for volunteer computing. In *Proceedings of the* 1^{st} *IEEE International Conference on e-Science and Grid Technologies (e-Science 2005)*, pages 196–203, 2005.

[4] D. Blough and G. Sullivan. A comparison for voting strategies for fault-tolerant distributed systems. In *Proceedings of the* 9^{th} *Symposium on Reliable Distributed Systems (SRDS 1990)*, pages 136–145, 1990.

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

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

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

[8] 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.

[9] A. Fernández, C. Georgiou, L. Lopez, and A. Santos. Brief Announcement: Reliably executing tasks in the presence of malicious processors. In *Proceedings of the* 19^{th} *International Symposium on Distributed Computing (DISC 2005)*, pages 490–492, 2005.

[10] 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.

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

[12] Ch. Georgiou, A. Russell, and A.A. Shvartsman. Work-competitive scheduling for cooperative computing with dynamic groups. *SIAM Journal on Computing,* 34(4):848–862, 2005.

[13] N. Goyal, G. Kindler, and M. Saks. Lower bounds for the noisy broadcast problem. In *Proceedings of the* 46^{th} *IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pages 40–52, 2005.

[14] P. Kanellakis and A. Shvartsman. *Fault-Tolerant Parallel Computation*. Kluwer, 1997.

[15] Z.M. Kedem, K.V. Palem, A. Raghunathan, and P. Spirakis. Combining tentative and definite executions for dependable parallel computing. In *Proceedings of the* 23^{rd} *ACM Symposium on Theory of Computing (STOC 1991)*, pages 381–390, 1991.

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

[17] 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.

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

[19] E. Kushilevitz and Y. Mansour. Computation in noisy radio networks. *SIAM Journal on Discrete Mathematics,* 19(1):96–108, 2005

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

[21] M. Paquette and A. Pelc. Optimal decision strategies in Byzantine environments. In *Proceedings of the* 11^{th} *Colloquium on Structural Information and Communication Complexity (SIROCCO 2004)*, pages 245–254, 2004.

[22] R. Reischuk and B. Schmeltz. Reliable computation with noisy circuits and decision trees – A general $n \log n$ lower bound. In *Proceedings of the* 32^{nd} *IEEE Symposium on Foundations of Computer Science (FOCS 1991)*, pages 602–611, 1991.

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

Appendix

Proof of Lemma 3.1

Let D_A denote the value returned (i.e., decided) by algorithm A, and let v denote the correct result of the task. We first prove the following claim:

Claim: The success probability of A is $\min(\Pr[D_A = 0|v = 0], \Pr[D_A = 1|v = 1])$.

Proof: Let us assume, w.l.o.g., that $\Pr[D_A = 0|v = 0] \leq \Pr[D_A = 1|v = 1]$. Note that in our model there is no a priori restriction on the distribution of the correct values. Hence, if $\Pr[v = 0] = 1$, we have that $\Pr[\text{success}] = \Pr[D_A = 0|v = 0]\Pr[v = 0] + \Pr[D_A = 1|v = 1]\Pr[v = 1] = \Pr[D_A = 0|v = 0]$. This completes the proof of the claim. ■

Let S be the number of workers the algorithm A assigns the task, R the number of workers whose reply the algorithm receives before deciding, and Z the number of replies with value 0 among the R replies. We use $D_A(s, r, z)$ to denote the value decided by the algorithm when $S = s \leq n$, $R = r \leq s$, and $Z = z \leq r$. We introduce the function Q_A as follows,

$$Q_A(i, s, r, z) = \Pr[D_A(s, r, z) = i | S = s, R = r, Z = z],$$

for $i \in \{0, 1\}$. Note that Q_A is a characteristic of algorithm A. Since in our model any algorithm must always decide, we have that

$$Q_A(0, s, r, z) + Q_A(1, s, r, z) = 1. \quad (2)$$

Additionally, we use the notation, $B(z, i, s, r) = \Pr[Z = z|v = i, S = s, R = r]$. With these definitions, we have that, for $i \in \{0, 1\}$,

$$\Pr[D_A = i|v = i] = \sum_{s=0}^{n} \Pr[S = s] \sum_{r=0}^{s} \Pr[R = r|S = s] \sum_{z=0}^{r} B(z, i, s, r) Q_A(i, s, r,$$

Now, we have that when $v = 0$ a correct worker always returns 0 and a faulty worker always returns 1, while the behavior is exactly the opposite when $v = 1$. Then, for $z \leq r \leq s \leq n$,

$$B(z, 0, s, r) = B(r - z, 1, s, r). \quad (5)$$

Additionally, since $p < 1/2$ and $f < n/2$, we have that, when $z < r/2$,

$$B(z, 0, s, r) < B(r - z, 0, s, r), \quad (6)$$

and

$$B(r - z, 1, s, r) < B(z, 1, s, r). \quad (7)$$

Now we can show that the decision functions that decide by majority voting among the received replies maximize the success probability. In particular, we show that we can derive from A an algorithm A' whose success probability is no less than $1 - \varepsilon$ with decision function $D_{A'}$ such that $Q_{A'}(0, s, r, z) = 0$ and $Q_{A'}(1, s, r, r - z) = 0$, for all $r \leq s \leq n$ and $z < r/2$. Let us first observe, from Eq. (3), that for each $r \leq s \leq n$ and $z < r/2$, $\Pr[D_A = 0|v = 0]$ and $\Pr[D_A = 1|v = 1]$ have the terms

$$\Pr[S = s|v = 0]\Pr[R = r|v = 0, S = s] \cdot$$
$$(B(z, 0, s, r)Q_A(0, s, r, z) + B(r - z, 0, s, r)Q_A(0, s, r, r - z)),$$

and

$$\Pr[S = s|v = 1]\Pr[R = r|v = 1, S = s] \cdot$$
$$(B(z, 1, s, r)Q_A(1, s, r, z) + B(r - z, 1, s, r)Q_A(1, s, r, r - z)),$$

respectively. Additionally, these are the only terms in which $Q_A(0, s, r, z)$, $Q_A(0, s, r, r-z)$, $Q_A(1, s, r, z)$, and $Q_A(1, s, r, r-z)$ appear.

Let us design algorithm A' to behave exactly like A but with a decision function $D_{A'}$ such that $Q_{A'}(0, s, r, z) = 0$ and $Q_{A'}(1, s, r, r-z) = 0$. From Eq. (2) we have that $Q_{A'}(1, s, r, z) = 1$ and $Q_{A'}(0, s, r, r-z) = 1$. We also have, from Eqs. (6) and (7), that $B(z, 0, s, r) < B(r-z, 0, s, r)$ and $B(r-z, 1, s, r) < B(z, 1, s, r)$, and from Eq. (5) that $B(z, 0, s, r) = B(r-z, 1, s, r)$ and $B(r-z, 0, s, r) = B(z, 1, s, r)$. Then, with this definition of $D_{A'}$ we have that

$$\min(\Pr[D_A = 0|v = 0], \Pr[D_A = 1|v = 1]) = 1 - \varepsilon \leq$$
$$\min(\Pr[D_{A'} = 0|v = 0], \Pr[D_{A'} = 1|v = 1]),$$

and hence the success probability of A' is at least $1 - \varepsilon$. This completes the proof. ■