Evaluating Reliability Techniques in the Master-Worker Paradigm

Evgenia Christoforou*†, Antonio Fernández Anta*, Kishori M. Konwar‡, Nicolas Nicolaou*
† Universidad Carlos III, Madrid, Spain. evgenia.christoforou@imdea.org
* IMDEA Networks Institute, Madrid, Spain. antonio.fernandez@imdea.org, nicolas.nicolaou@imdea.org
‡ MIT, Cambridge MA, USA. kishori@csail.mit.edu

Abstract*—A distributed system is considered that carries out computational tasks according to the master-worker paradigm. A master has a set of computational tasks to resolve. She assigns each task to a set of workers over the Internet, instead of computing the task locally. For each task each worker reply to the master with the task result. Since the task was not computed locally, the master can not trust the result for two main reasons: (i) workers might deliberately provide an incorrect result, (ii) the result is corrupted due to some hardware or software failure during the execution of the task. Given the above, we can model our workers as either "altruistic", always willing to provide the correct result to each task, or "troll" that are trying to provide an incorrect result to each task. Moreover we model the failure of the worker to comply with her intended behavior, as an error probability ϵ. The goal of the master is to compute the correct result of all the tasks with high probability. In the literature two techniques have been used to achieve this goal: (i) "voting", that determines the correct result of a task given multiple replies of distinct workers; (ii)"challenges", that are tasks whose result is known and can be used to detect altruistic workers. What separates our work from the current literature is the realistic modelling of the worker's behavior and the fact that we do not restrict the task result to a binary set of answers; the domain of possible replies for a task can have multiple correct and multiple incorrect results. Given the above we evaluate the performance of the two techniques described in the literature in the scenario where $\epsilon = 0$ and when $\epsilon > 0$. Performance is measured in terms of: (1) time, i.e., the number of rounds performed by an algorithm for the computation of all the tasks, and (2) work, i.e., the number of total task computations performed by the workers. The case where $\epsilon = 0$ is used as a best case scenario that provides the optimal time and work bounds of the problem. In the case where $\epsilon > 0$ we propose two "natural" algorithms: one using a combination of both voting and challenges, and a second one using only voting. Both algorithms assume that certain system parameters are known. Since this might not always be the case we also provide an algorithm that estimates correctly these parameters with high probability.

I. Introduction

Distributed computing systems following the master-worker paradigm are increasing in popularity over the past decades. In the literature, this paradigm is encountered under several names like: volunteer computing, desktop grid computing, public resource computing and so on. This work focuses on master-worker task computations, where a master entity has a set of tasks to be computed that is unable or unwilling to compute locally. Hence, she assigns these tasks over the Internet to worker entities willing to perform the task and reply back to the master with a result. The inherent limitation of this load distribution scheme is the unreliable nature of the workers. We assume that there is no mean of verifying an answer provided by a worker unless we know the set of solutions for the particular task.

One classical example of the master-worker paradigm is SETI@home [13] that uses the BOINC [1], [3] infrastructure. In the case of SETI@home, the project is collecting and analysing signals from space in a search for extraterrestrial forms of life. Volunteers willing to help the search register their machines, receive tasks to be executed (when available), and report the results back to the master.

Evidence exist though that volunteers might actually misreport values [1], [2], [10], [11]. The most straight forward explanation is that workers might have ulterior motives for misreporting a result. Another reason might be that actually a hardware or software failures happened during the computation of the task that was not detected.

Drawing from the above example and the work of Kondo et al. [11], where they have characterized errors in BOINC systems, we can infer the existence of two type of workers. **(i) Altruistic**[1] **workers:** This type of worker is positive towards executing the task, and willing to provide the correct result. In the case of a BOINC system, a "positive" worker will let its machine execute the task and report back the result. **(ii) Troll**[2] **workers:** This type of worker is negative towards executing a task and wants to convey an incorrect result to the master. Hence, it can miscalculate a task on purpose and tries to report an incorrect result. In this work we do not assume any type of intelligent strategy to fool the system, nor that a troll has any information on the actions taken by the master or the other workers.

We can safely assume that both types of workers can be susceptible to a small error probability ϵ that is related to

Supported in part by FP7-PEOPLE-2013-IEF grant ATOMICDFS No:629088, Ministerio de Economia y Competitividad grant TEC2014-55713-R, Regional Government of Madrid (CM) grant Cloud4BigData (S2013/ICE-2894, co-funded by FSE & FEDER), NSF of China grant 61520106005 and Spanish Ministry of Education grant FPU2013-03792 .

[1]For historical reasons, to match with original work in the field of master-worker task computing.

[2]Historically they are called malicious workers, but since we are not assuming any intelligent behavior to harm the system here we call them troll workers.

hardware or software failures or to any other factor that would force them to deviate from their intended behavior. Thus an altruistic worker, will have a high probability of reporting a correct result, while a troll will have a low probability of reporting a correct result.

The goal of the master is to identify the correct result for each task assigned with high probability. As far as we know, two techniques are used (individually or combined) for increasing reliability. **(i) Voting:** The master uses redundancy by assigning the same task to multiple workers. After all the task replies are collected, the master uses a voting scheme [4], [15], [14] to decide on the correct result. This may fail to provide optimal reliability since a high concentration of incorrect results reported may lead to a decision on the task result that is incorrect. **(ii) Challenges:** The master uses a set of tasks with known solutions to identify the workers that are replying correctly. Again this approach can not guarantee optimal reliability if the same workers in different time intervals provides correct and incorrect results due to an error during the computation of the task (as we discussed above). Similar or identical concepts to this approach are encountered by the name of spot-checking, auditing or quiz [16], [18].

Both techniques add an extra load on the computation of the task. On the one hand, with voting the same task needs to be executed by multiple workers, thus the resources of the system are not used in an optimal way. On the other hand, challenges require that the workers compute tasks with known solutions, thus not only resources are not used in an optimal way but also the execution time of a task increases.

An added difficulty when using these techniques is related to the nature of the task. If you have tasks that can have multiple correct and multiple incorrect results, maybe you can not guarantee that the result of each task will be correct with high probability. The common assumption in the literature is that tasks have a binary set of solutions (i.e. only one correct and one incorrect solution are feasible).

Voting and challenges have been widely used either alone or in combination to provide the correct task result for each task with high probability. What is yet unclear is the advantage of one technique over the other or the combination of these techniques in terms of time and work complexity, given tasks with multiple correct and multiple incorrect results.

Related Work: In this section we review a few of the most representative works that use voting and challenges in the master-worker paradigm. In the work of Fernández et al. [9], [8] two voting mechanisms are presented under the assumption of a binary set of possible responses to the master. Targeting at a high reliability while minimizing redundancy, the authors provide analytical results assuming that the number of malicious workers or the probability of a worker acting maliciously is known. In the work of Konwar et al. [12] these last assumptions on having information on the malice are removed and new algorithms are proposed to approximate these probabilities. The lower bounds on the amount of work necessary, when the set of responses is binary, are comparable to the work complexity of our proposed algorithms. These works assume that all workers might fail in every round with a certain probability. In our work we take a more realistic assumption, considering that workers might reply incorrectly with a different probability related to their nature (that is, we assume two types of workers' failures).

In his work, Sarmenta [16] assumed the presence of malicious and altruistic workers, with only the malicious workers having a constant probability of submitting an erroneous result and binary set of response. Based on these assumptions he introduced sabotage-tolerance mechanisms that used voting and a spot checking technique (what we are calling challenges). Given the assumption that altruistic workers will always reply with the correct task result, the mechanisms combine voting with challenges to create a reputation for each worker and increase the systems' reliability while trying to keep redundancy low.

In their work, Zhao and Lo [18] compare voting to challenges (called Quiz) under two assumptions: that all malicious workers return the same incorrect result or that malicious workers return distinct results. They use as performance metrics the accuracy and overhead, and through simulations they show the trade-off among these performance metrics and the two reliability techniques. Their work is mostly experimental and does not provide any complexity analysis. Additionally they do not assume a density of solutions nor an error probability on the altruistic workers.

Finally, Sonnek et al. [17] design algorithms for efficient task allocation based on the reputation of each worker. Hence the reliability of each worker is a statistical property that depends on the results submitted by each worker. Each task is being redundantly assigned to a group of workers with a targeted reliability. These groups are formed based on different algorithms. Their algorithms are evaluated through simulations. In contrast, we provide analytical bounds on the performance metrics of our proposed algorithms.

Contributions: Our contributions can be summarized as follows:

- We model the master-worker paradigm in the presence of altruistic and troll workers using five parameters $\langle \epsilon, f_a, s, r, T \rangle$ (Section II): (i) ϵ is the probability that an altruistic worker may reply with an incorrect result or a troll worker with a correct result, (ii) f_a is the fraction of altruistic workers over the set of workers, (iii) s, r are the number of correct and incorrect answers for a task respectively, leading to the more realistic assumption that tasks are not only binary but rather may have solutions in a broader domain, and (iv) $T \subseteq \{C, V\}$ the set of reported result evaluation techniques, i.e., *challenges* and *voting*. Additionally we define two measures: (i) "time" and (ii) "work", for evaluating the complexity of the proposed algorithms.
- We fix $\epsilon = 0$ in Section III, i.e., the simple case where altruistic workers always reply with a correct result and troll workers always with an incorrect result. Given this idealistic scenario, we identify asymptotically optimal bounds on the time and work complexity when challenges and voting are

used separately. While it is clear that when challenges are used the master can receive the correct task result with probability one, this is not always the case when voting is used alone. In the case of voting, we were able to show a negative result giving conditions on the parameters of s, r, n_a and n_t, where the master will not always be deciding on the correct task result with probability one. This result reveals that the domain of reported results is important even in the simple case where $\epsilon = 0$.

- We then make the realistic assumption that $\epsilon > 0$ (Section IV) and we provide two algorithms MWMIX and MWVOTE that solve correctly all the tasks whp. Both algorithms assume that s, ϵ and f_a are known. Algorithm MWMIX uses both challenges and voting, and can be applied if $1 - \epsilon > \frac{s}{s+1}$. Algorithm MWVOTE uses only voting, and can be applied if $f_a(1-\epsilon) + (1-f_a)\epsilon > \frac{s}{s+1}$. What is interesting to observe is that these algorithms have a log factor overhead compared to the case where $\epsilon = 0$, which as shown in [12], is a necessary price to pay when voting is used.
- Finally, in the case where ϵ and f_a are not known, in Section V we provide algorithm E_1 that estimates these parameters within tight bounds *whp*.

II. Model

Our setting consists of a master process M and a set $W = \{w_1, \ldots, w_n\}$ of n worker processes. Workers might be unreliable and produce an incorrect result.

Definition 2.1 (Problem statement.): The master must guarantee with high probability (see below) the correct result for each task t_i where $t_i \in \mathcal{T} = \{t_1 \cdots t_n\}$, without computing the task locally.
To keep the pseudocode simple for the purpose of exposition we assume that $|\mathcal{T}| = n$. (The algorithms presented in this paper can be easily extended to run for $|\mathcal{T}| = O(poly(\log n))$ without violating the statements of correctness.)

Computation and Communication model: Processes in the system communicate by exchanging messages via reliable communication channels. Computation proceeds in synchronous rounds. For a process p a round consists of the following steps: (i) receive incoming messages, (ii) perform computation on the received messages and produce a set of outgoing messages, and (iii) send the produced messages. During a round each worker can compute only one task from the master, and report back the result of the task. A synchronous algorithm A is a collection of processes, and its state is defined over the vector of the states of each process in that collection. An execution ξ of A is an (infinite) sequence of states. We denote by $execs(A)$ the set of executions of A.

Performance Measures: An algorithm A is evaluated in terms of: (i) *time*, and (ii) *work*. Time is defined as the number of rounds that an algorithm A requires in order to determine the result of all n tasks in $\mathcal{T}$. Work represents the number of aggregated results computed by each worker in algorithm A. More formally, let $comp_\xi(w,t)$ be the number of times a worker w computes task t during an execution ξ of algorithm A. The task t can be one the tasks in $\mathcal{T}$ or a task chosen from a set $\mathcal{C}$ of challenge tasks available to the master M. Then,

$$work_\xi = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} comp_\xi(w_i, t_j) + \sum_{t \in \mathcal{C}} comp_\xi(w_i, t) \right)$$

is the work of all the workers in ξ. Thus, the work of an algorithm A is defined as $Work(A) = \max_{\xi \in execs(A)}(work_\xi)$ over all executions of A. The work captures the redundancy used by an algorithm (i.e., the number of workers that compute the same task), as well as the computation performed on challenges.

Density of Solutions: A reported result v for a task t may take values from a domain $D(t)$. Let $S(t) \subset D(t)$ be the set of *correct solutions* and $R(t) = D(t) \backslash S(t)$, the set of reported results that are incorrect solutions for task t. We only consider tasks t that have at least one correct solution, i.e., $|S(t)| \geq 1$.

Given $D(t)$ and $S(t)$ we define the *density of solutions* for task t as $\frac{|S(t)|}{|D(t)|}$. The density of solutions affects the techniques used to determine the correct task result. For simplicity of presentation, we will assume that the size of the domain $d = |D(t)|$, the number of correct solutions $s = |S(t)|$, and the number of incorrect solutions $r = |R(t)|$ are the same for every $t \in \mathcal{T}$. Hence, the density of solutions is the same for all tasks t.

Worker Types: We assume that each worker $w_i \in W$ is either *altruistic* or a *troll* and its type remains the same throughout the execution of the algorithm. Let $W_a \subseteq W$ be the set of altruistic and $W_t \subseteq W$ the set of trolls. We use $n_a = |W_a|$ and $n_t = |W_t|$ to denote the sizes of these sets. We assume that $n_a \geq 1$. Observe that $W = W_a \cup W_t$ and $n = n_a + n_t$. We also use $f_a = \frac{n_a}{n}$, to denote the fraction of altruistic workers. Altruistic workers are "positive" towards executing a task, always aiming at returning a correct result. Trolls are "negative" towards the task, always trying to provide an answer from the set of incorrect solutions. All workers are subject to an *error probability* ϵ that deviates a worker from its specification. For simplicity of presentation we will assume that ϵ is the same for all workers. In particular, an altruistic worker replies with a correct result with probability $1 - \epsilon$ and with an incorrect result with probability ϵ, while for a troll holds the contrary. For all workers, the correct and incorrect results are selected uniformly at random from the respective sets $S(t)$ and $R(t)$ for a task t. When $\epsilon = 0$ an altruistic worker always replies with the correct result and a troll always replies with an incorrect result.

Result Evaluation: We assume that the master can use two techniques to determine the correct task result: *challenges* (C) and *voting* (V) . A *challenge* is a task of which the master knows the correct result. When a challenge is used, the master knows if a worker replied with a correct or an incorrect result. This information can help the master determine the correct result for each task in $\mathcal{T}$. When *voting* is used, the same task is assigned to multiple workers, and the master uses some voting technique to decide upon the correct task result.

Definition 2.2 (Environmental Parameters.): A master-worker system environment can be characterized by the parameters (ϵ, s, r, f_a, T) where: (i) ϵ is the worker error probability, (ii) s is the set of correct replies for each task t, (iii) r is the set of incorrect replies for each task t (iv) f_a the fraction of altruistic workers, and (v) $T \subseteq \{C, V\}$ the technique used for the evaluation of the reported results.

Probabilities: We use the common definition of *an event $\mathcal{E}$ occurring with high probability* (*whp*) to mean that $\Pr[\mathcal{E}] = 1 - O(n^{-\alpha})$ for some constant $\alpha > 0$.

III. Exact Worker Behavior ($\epsilon = 0$)

In this section we examine the simple case where it is given that the error probability of each worker is $\epsilon = 0$. Hence, altruistic workers always reply with a correct result and trolls always reply with an incorrect result. This scenario is somewhat idealistic in practical master-worker systems, but it is used here to provide the best case analysis of our problem. The results of this section will be used as a reference from the next section, in order to evaluate the performance of the proposed algorithms compared to this optimistic scenario. Notice that any algorithm requires at least $\frac{n}{n_a}$ rounds to complete the computation of all the $|\mathcal{T}| = n$ tasks, and needs to perform n work, if all tasks have to be computed correctly with full reliability.

Algorithm 1 Simple algorithm MWSIMPLE_0 where $\epsilon = 0$ and $T = \{C\}$.

```
1: Send challenge task $t$ to all workers in $W$
2: $R[j] \leftarrow$ result received from $w_j \in W$, $j \in [1, |W|]$
3: $U_a \leftarrow \{w_i | R[i]$ is correct$\}$
4: for $i = 1 : |U_a| : n$ do          ▷ for loop increments $i$ by $|U_a|$
5:     Send task $t_{i+k-1}$ to $k$th worker in $U_a$, $k \in [1, |U_a|]$
6:     Add received result for $t_{i+k-1}$ into $Results[i + k]$
7: end for
8: return $Results$
```

In the simple algorithm MWSIMPLE_0 that appears in Algorithm 1, the set of altruistic workers is not known and thus we use challenges ($T = \{C\}$) to determine it in a single round. Once the altruistic workers are identified the master makes unique task assignments to each of them. Hence the master needs $1 + \frac{n}{n_a}$ rounds to decide for n tasks, and requires $2n$ amount of work.

Theorem 3.1: Algorithm MWSIMPLE_0 has asymptotically optimal time $\Theta(\frac{n}{n_a})$ and optimal work $\Theta(n)$, and compute all the n tasks with probability 1, when $\epsilon = 0$.

Looking more closely at the general case of algorithms that only use voting ($T = \{V\}$) we have found that it is possible to solve all the tasks efficiently if $n_a > s \cdot n_t$. The algorithm MWVOTE_0 that solves the problem is given in Algorithm 2. In this algorithm the master sends the first task t_1 to all the workers. No incorrect returned value can appear more than n_t times, while from the pigeonhole principle at least one correct value appears at least $n_a/s > n_t$ times. Then, the workers that return values with multiplicity larger than n_t are all altruistic. These workers are stored in U_a and used to solve the rest of tasks. The size of U_a is at least $n_a - n_t(s-1) > n_t$, and hence the master needs at most $1 + \frac{n-1}{n_a - n_t(s-1)}$ rounds and $2n - 1$ work.

Algorithm 2 Simple algorithm MWVOTE_0 where $\epsilon = 0$, $n_a > s \cdot n_t$, and $T = \{V\}$.

```
1: Send task $t_1$ to all workers in $W$
2: Add worker $w_j$ to set $R[v]$ if it replied with value $v$
3: $U_a \leftarrow \bigcup_{v:|R[v]|>n_t} R[v]$
4: $Results[1] \leftarrow$ any value $v : |R[v]| > n_t$
5: for $i = 2 : |U_a| : n$ do          ▷ loop increments $i$ by $|U_a|$
6:     Send task $t_{i+k-1}$ to $k$th worker in $U_a$, $k \in [1, |U_a|]$
7:     Add received result for $t_{i+k-1}$ into $Results[i + k - 1]$
8: end for
9: return $Results$
```

From the above we have the following theorem:

Theorem 3.2: The algorithm MWVOTE_0 compute all the n tasks with probability 1 when $\epsilon = 0$ and $n_a > s \cdot n_t$. It has time $O(\frac{n}{n_t})$ and optimal work $\Theta(n)$.

In the case that $n_a = s \cdot n_t$ we have a negative result.

Theorem 3.3: If $\epsilon = 0$ and $n_a = s \cdot n_t$, then for any $r > 0$ there exists no algorithm that allows the master node to returns the correct result of a task t with probability greater than $\frac{s}{s+1}$ in any execution.

Proof: Lets assume that there exists such an algorithm A_m. The master M assigns the task t to a subset W' of the set of workers W. We assume w.l.o.g. that it sends the task to all the workers in W, since A_m can disregard the replies from the workers not in W'. Note that the master does not have any prior information on the correct answers or the answers that each worker returns. So we can examine possible executions for the same task t, where the incorrect results and the troll workers are different. Since $\epsilon = 0$, an altruistic worker always returns a correct answer and a troll returns an incorrect answer. We assume that the same worker returns the same answer if asked to compute the task more than once.

Consider now an execution ξ_1 of A_m constructed as follows. Let $D(t) = \{a_1, \ldots, a_s, a_{s+1}, \ldots, a_{s+r}\}$ be the domain of possible replies for t, where $S(t) = \{a_1, \ldots, a_s\}$ is the set of correct results for t and the rest of the answers are incorrect. Since $\epsilon = 0$ then the master receives n_a correct answers and n_t incorrect answers. Let us assume, that the troll workers are the set $\{w_{n_a+1}, \ldots, w_{n_a+n_t}\}$. Furthermore in ξ_1, let each answer $a_i \in S(t)$ be returned by $\frac{n_a}{s}$ workers. W.l.o.g assume that workers $\{w_{(i-1)\frac{n_a}{s}+1}, \ldots, w_{i\frac{n_a}{s}}\}$ reply with answer $a_i \in S(t)$. Observe that all the trolls reply with the same incorrect answer a_{s+1}. Since, $n_a = s \cdot n_t$, it follows that $n_t = \frac{n_a}{s}$ troll workers reply with a_{s+1}. Since, according to our assumption, algorithm A_m returns a correct answer with probability $p_c > \frac{s}{s+1}$, then it follows by the pigeonhole principle, that A_m will return some answer $a_i \in S(t)$ with probability $Pr[a_i] > \frac{s}{(s+1)s} = \frac{1}{s+1}$, in ξ_1. Let w.l.o.g. a_1 be that answer.

Assume now a second execution ξ_2 which is similar with ξ_1 with the difference that the troll workers are the set $\{w_1, \ldots, w_{\frac{n_a}{s}}\}$ and the correct answers is the set

$S(t) = \{a_2, \ldots, a_{s+1}\}$. Each answer is returned by the same set of workers as in ξ_1. Note that correct workers $\{w_{n_a+1}, \ldots, w_{n_a+n_t}\}$ all reply with a_{s+1}. Also the troll workers in ξ_2 all reply with a_1. Since all the workers reply with the same answers to the master in both ξ_1 and ξ_2, and since the master does not have any prior info on the correct answers, then M will not be able to distinguish ξ_1 from ξ_2. Thus, if according to A_m, M returned a_1 with probability $Pr[a_1] > \frac{1}{s+1}$ in ξ_1 then M will return a_1 with the same probability in ξ_2 as well. Since a_1 is an incorrect answer in ξ_2, then M returns a correct answer with probability $\mathbf{Pr}[\text{ return } a_i \in S(t)] = 1 - Pr[a_1] < 1 - \frac{1}{s+1} = \frac{s}{s+1}$. This however contradicts our initial assumption that A_m returns a correct answer with probability greater than $\frac{s}{s+1}$ in ξ_2 and completes our proof. ■

Looking at the above two results one may conjecture that $n_a = s \cdot n_t$ is in fact the boundary between solvability and unsolvability. However, this is not the case, since, for instance, if $r = 1$ and n_a is not a multiple of n_t, even if $n_a < s \cdot n_t$ it is possible to solve all tasks efficiently. Using the opposite logic the incorrect value will appear n_t times, while from the pigeonhole principle at worst one correct value appears $n_a \mod n_t$ times. Then, the workers that return values with cardinality smaller than n_t are all altruistic. These workers like in Algorithm 2 can be stored in U_a and used to solve the rest of tasks following an analogous algorithm. The size of U_a is at least $n_a - \left\lfloor \frac{n_a}{n_t} \right\rfloor \cdot n_t$, and hence the master needs at most $1 + \frac{n-1}{n_a - \lfloor \frac{n_a}{n_t} \rfloor \cdot n_t} < n$ rounds and $2n - 1$ work. Thus, an algorithm analogous to Algorithm 2 has time $O(n)$ and optimal work $\Theta(n)$.

IV. Probabilistic Worker Behavior ($\epsilon > 0$)

We are now moving to the case where the error probability of each worker is $0 < \epsilon < \frac{1}{2}$. In this case an altruistic worker may reply with an incorrect result, or a troll with a correct result with probability ϵ. Note that we do not consider the case when $\epsilon \geq \frac{1}{2}$, because in that case essentially the roles are switched. We provide algorithms that cover the full spectrum of values for the density of solutions $\rho \in (0, 1)$ and the fraction of altruistic workers $f_a \in (0, 1]$. Notice that the algorithms presented here also apply in the case where $\epsilon = 0$, but they may induce extra performance overhead.

Under this model the master receives the correct result from a randomly selected worker with probability at least $f_a(1 - \epsilon)$. We provide two different algorithms for this setting: (i) algorithm MWMIX that uses both challenges and voting, i.e. $T = \{C, V\}$, and (ii) algorithm A_2 that uses only voting, i.e. $T = \{V\}$, which is possible only when the density of solutions satisfy a given bound.

Note that in this section we do not present an algorithm that only relies on challenges to compute all tasks in $\mathcal{T}$ with high probability. This is so, because even if the set W_a of altruistic workers is known, and only these workers are used, the value returned by the execution of a task is correct only with constant $1-\epsilon$ probability. An algorithm that does not use some form of voting will not execute a task more than once, and cannot improve this probability.

In this section we assume that the algorithms know the parameters s, f_a and ϵ. For the case that this is not true, we provide algorithm E_1 in the next section, that uses challenges to estimate them. Due to lack of space the proofs of correctness and performance of the algorithms are omitted.

Algorithm 3 The pseudo-code for algorithm MWMIX, at the master, with n workers W computing the results of n tasks in $\mathcal{T}$, where $\frac{s}{s+1} < 1 - \epsilon$ and $T = \{C, V\}$.

```
    Phase 1
 1: $R[1..n] \leftarrow \emptyset^n$          ▷ $R[j]$ is the list of results from worker $w_j$
 2: for $i = 1 : \lceil c \log n \rceil$ do
 3:     Send challenge task $t$ to all workers in $W$
 4:     Add received result from worker $w_j$ to $R[j]$
 5: end for
 6: for $i = 1 : n$ do
 7:     if # correct results in $R[i] \geq \lceil \frac{1}{2} c \log n \rceil$ then
 8:         $U_a \leftarrow U_a \cup \{w_i\}$
 9:     end if
10: end for
    Phase 2
11: $F[i] \leftarrow \emptyset$                    ▷ initially empty for all $1 \leq i \leq n$
12: for $j = 1 : \lceil k \log n \rceil$ do
13:     for $i = 1 : |U_a| : n$ do          ▷ loop increments $i$ by $|U_a|$
14:         Send task $t_{i+k-1}$ to $k$th worker in $U_a$, $k \in [1, |U_a|]$
15:         Add received result for $t_{i+k-1}$ to $F[i+k-1]$
16:     end for
17: end for
18: for $i = 1 : n$ do
19:     $Results[i] \leftarrow plurality(F[i])$
20: end for
21: return $Results$
```

A. *Algorithm* MWMIX*: Challenges and Voting*

In this section we present algorithm MWMIX, that uses a combination of challenges and voting. The general idea of the algorithm is to identify the workers in W_a, correctly *whp* and then use the estimated set U_a to compute *all* the tasks correctly *whp*. Below we describe algorithm MWMIX, with the pseudo code in Algorithm 3, using this strategy. As explained earlier, it is preferable to avoid using *challenge* method ($T = \{C\}$) because challenges imply computational burden on the master. Therefore, the algorithm uses challenges only to estimate U_a, and afterwards it uses voting ($T = \{V\}$) to determine the correct result for each task, i.e. $T = \{C, V\}$.

Description of algorithm MWMIX*:* Algorithm MWMIX consists of two phases. During the first phase, MWMIX computes an estimate of the set W_a, denoted by U_a, by using the challenge method. Phase 1 has $c \log n$ rounds, for a constant $c > 0$ that depends on ϵ (L:2). During each round the master sends out a distinct challenge task to every worker in W (L:3), and upon receiving the reponses from the workers stores the results in an array of lists $R[1], R[2], \cdots, R[n]$, where $R[i]$ denotes the list of results received from process $w_i \in W$ (L:4). Next based on the results in $R[\ ]$, the ID of

Algorithm 4 Algorithm MWVOTE, at the master process, performs n tasks using n workers for the case $\frac{s}{s+1} < f_a(1-\epsilon) + (1-f_a)\epsilon$ and $T = \{V\}$.

```
1: $F[i] \leftarrow \emptyset$                    ▷ initially empty for all $1 \leq i \leq n$
2: for $i = 1$ to $\lceil k \log n \rceil$ do            ▷ for some constant $k > 0$
3:     Choose a random permutation $\pi \in \Pi_n$
4:     Send each task $t_j \in \mathcal{T}$ to worker $w_{\pi(j)}$
5:     Add received result from worker $w_{\pi(j)}$ to $F[j]$
6: end for
7: for $i = 1 : n$ do
8:     $Results[i] \leftarrow plurality(F[i])$
9: end for
10: return $Results$
```

any worker that answered the majority of the challenge tasks correctly is included in the set U_a (L:8).

During the Phase 2, only the workers in U_a are used to compute the n tasks. Each task is executed $\lceil k \log n \rceil$ times. The results sent back by the workers in U_a are stored in the array of lists $F[1], F[2], \cdots, F[n]$ (L:15), where the results for task t_i are stored in list $F[i]$. Finally, the master decides for every task t_i the plurality of results in $F[i]$ to be the correct result. The results of the tasks in $\mathcal{T}$ are hence stored in the array variable $Results$, where $Results[i]$ is the result of task t_i (L:19).

Correctness and Performance Analysis: Now, we prove the correctness of all the tasks *whp* and the complexity of results. In Lemma 4.1, we show that at the end of Phase 1 every altruistic worker and only altrusitic workers are included in U_a. Using this lemma, we prove Theorem 4.1 which states that every task in $\mathcal{T}$ is computed correctly *whp*.

Lemma 4.1: In any execution of MWMIX, at the end of Phase 1 we have $U_a = W_a$, whp.

Theorem 4.1: If $\frac{s}{s+1} < 1 - \epsilon$, then Algorithm MWMIX computes all n tasks correctly, whp.

Theorem 4.2: Algorithm MWMIX runs in $\Theta(\frac{n}{n_a} \log n)$ synchronous rounds and performs $\Theta(n \log n)$ work.

B. Algorithm MWVOTE*: Use Voting Alone*

In this section, we present algorithm MWVOTE that uses voting mechanism alone (i.e., when $T = V$) to compute all the n tasks *whp*. Algorithm MWVOTE can be used when $\frac{s}{s+1} < f_a(1-\epsilon) + (1-f_a)\epsilon$. Note here that, without identifying the altruistic workers, the probability that a randomly selected worker will reply with the correct answer for a task t is $f_a(1-\epsilon)+(1-f_a)\epsilon$; i.e. receive the correct answer from an altruistic worker with probability $1-\epsilon$ or to receive the correct answer from a troll with probability ϵ.

Description of algorithm MWVOTE*:* Below we present our algorithm MWVOTE, and the pseudo-code for the algorithm is in Algorithm 4. The basic idea of MWVOTE is to exploit the fact that if a troll worker picks an answer for a task t randomly from $R(t)$ and $\frac{s}{d}$ is small, then the likelihood of an incorrect result being a majority or plurality among all the results is small. To apply this idea, the master distributes the n tasks according to a random permutation from Π_n, which is the set of all permutations over $[n] = \{1, 2, \ldots, n\}$. In other words, if the random permutation is π then the j^{th} task t_j is delegated to worker $w_{\pi(j)}$ (L:4). The whole process is repeated $\lceil k \log n \rceil$ rounds (L:2-6). The constant k is used to tune the exponent $\ell > 0$ in the denominator of $\frac{1}{n^\ell}$, required for the high probability guarantee. The results for each task t_j is accumulated in the multiset $R[j]$. When the **for** loop in lines 2–6 terminates, the result for each task $t_j \in \mathcal{T}$ is chosen to be the one that forms a plurality of results in $R[j]$ (L:8).

Correctness and Performance Analysis: The following theorems state that algorithm MWVOTE computes all tasks correctly *whp* under the assumed case.

Theorem 4.3: If $\frac{s}{s+1} < f_a(1-\epsilon) + (1-f_a)\epsilon$, Algorithm MWVOTE computes all n tasks correctly *whp*.

Theorem 4.4: Algorithm MWVOTE runs in $\Theta(\log n)$ synchronous rounds and performs $\Theta(n \log n)$ work.

Remark 4.1: Note that the algorithms presented in this section have a log factor overhead on the optimal work (Theorem 3.1). According to [12] a factor of *log* work overhead is the least amount of work required per task when voting is used. Thus, we can safely conclude on the optimality of both algorithms MWMIX and MWVOTE.

V. ALGORITHM E_1: TIGHTLY ESTIMATING f_a AND ϵ

As shown in Section IV, algorithms MWMIX and MWVOTE are possible only if we know parameters of the system, like the probability of error ϵ, the fraction of altruistic workers f_a, and the number of correct results s, to check applicability. In this section, we assume s is known, but that neither the value of f_a and ϵ are known *a priori*. (Note that it is reasonable to assume that the number of correct answers s is known.) Hence, we provide an algorithm to estimate f_a and ϵ, *whp*. As a byproduct, the algorithm also estimates $f_a(1-\epsilon) + (1-f_a)\epsilon$, *whp*. Our goal is to estimate all these values, with user defined bounds, in a manner called (ε, δ)-approximation. By choosing $\varepsilon, \delta \in O(\frac{1}{n^c})$ for some $c > 0$ we can provide a tight estimate of the value within a $\pm\varepsilon$ factor and with high probability (greater than $1 - \delta$).

Formally, let Z be a random variable distributed in the interval $[0, 1]$ with mean μ_Z. Let $Z_1, Z_2, \ldots$ be independently and identically distributed according to the Z variable. We say that an estimate $\tilde{\mu}_Z$ is an (ε, δ)-approximation of μ_Z if $\mathbf{Pr}[\mu_Z(1-\varepsilon) \leq \tilde{\mu}_Z \leq \mu_Z(1+\varepsilon)] > 1 - \delta$. Estimating the value of μ_Z may be done by collecting *sufficient* samples and selecting the majority as the outcome. However, such a solution is not feasible if the number of samples are not known *a priori*.

The Stopping Rule Algorithm: Algorithm 6 is an algorithm for calculating an (ε, δ)-approximation of the desired parameters, where the error tolerance bounds δ and ε are $O(\frac{1}{n^c})$, for some $c > 0$. The core idea behind E_1 is based on the Stopping Rule Algorithm (*SRA*) of Dagum *et al.* [6]. For completeness we reproduce in Algorithm 5 the *SRA* for estimating the mean of a random variable with support in $[0, 1]$, with (ε, δ)-approximation. *SRA* computes an (ε, δ)-approximation with an

Algorithm 5 The Stopping Rule Algorithm (*SRA*) for estimating μ_Z.

```
   input parameters: $(\varepsilon, \delta)$ with $0 < \varepsilon < 1$, $\delta > 0$
1: Let $\Gamma = 4\lambda \log(\frac{2}{\delta})/\varepsilon^2$          $\triangleright \lambda = (e-2) \approx 0.72$
2: Let $\Gamma_1 = 1 + (1+\varepsilon)\Gamma$
3: initialize $N \leftarrow 0, S \leftarrow 0$
4: while $S < \Gamma_1$ do
5:     $N \leftarrow N + 1$
6:     $S \leftarrow S + Z_N$
7: end while
8: return $\tilde{\mu}_Z \leftarrow \frac{\Gamma_1}{N}$
```

optimal number of samplings, within a constant factor [6]. Thus *SRA*-based method provides substantial computational savings. Let us define $\lambda = (e-2) \approx 0.72$ and $\Gamma = 4\lambda \log(\frac{2}{\delta})/\varepsilon^2$. Now, Theorem 5.1 (slightly modified, from [6]) tells us that *SRA* provides us with an (ε, δ)-approximation with the number of trials within $\frac{\Gamma_1}{\mu_Z}$ *whp*, where $\Gamma_1 = 1+(1+\varepsilon)\Gamma$.

Theorem 5.1: [*Stopping Rule Theorem*] Let Z be a random variable in $[0,1]$ with $\mu_Z = \mathbb{E}[Z] > 0$. Let $\tilde{\mu}_Z$ be the estimate produced and let N_Z be the number of experiments that *SRA* runs with respect to Z on inputs ε and δ. Then, (i) $\mathbf{Pr}[\mu_Z(1-\varepsilon) \leq \tilde{\mu}_Z \leq \mu_Z(1+\varepsilon)] > 1-\delta$; (ii) $\mathbb{E}[N_Z] \leq \frac{\Gamma_1}{\mu_Z}$, and (iii) $\mathbf{Pr}[N_Z > (1+\varepsilon)\frac{\Gamma_1}{\mu_Z}] \leq \frac{\delta}{2}$.

Description of algorithm E_1: The idea behind algorithm E_1 is to sample two binary random variables: (i) $Z^1 \in \{0,1\}$, whose mean is "close" to f_a; and (ii) $Z^2 \in \{0,1\}$, whose mean is $f_a(1-\epsilon) + (1-f_a)\epsilon$. Then E_1 creates (ε, δ)-approximation estimates for both of these means, using the *SRA* algorithm for $\delta, \varepsilon \in O(\frac{1}{n^c})$, for some $c > 0$. Using these estimates, it solves for a (ε, δ)-approximation for the different parameters. Below we explain the sampling process.

Z^1 is defined as follows: the master randomly picks a worker w from W, sends ℓ (a positive integer, explained later) challenges to w, and collects and verifies the results. If the majority of the results R are correct then $Z^1 = 1$, otherwise $Z^1 = 0$. We use $\mathit{CorrMaj}(R)$ to denote that the majority of the results in R are correct. Clearly,

$$\begin{aligned}\mathbb{E}[Z^1] &= \mathbf{Pr}[w \in W_a] \cdot \mathbf{Pr}[\mathit{CorrMaj}(R) | w \in W_a] \\ &\quad + \mathbf{Pr}[w \notin W_a] \cdot \mathbf{Pr}[\mathit{CorrMaj}(R) | w \notin W_a] \\ &= f_a \cdot \mathbf{Pr}[\mathit{CorrMaj}(R) | w \in W_a] \\ &\quad + (1-f_a) \cdot \mathbf{Pr}[\mathit{CorrMaj}(R) | w \notin W_a]\end{aligned}$$

Next, by exploiting the fact that $\epsilon < \frac{1}{2} - \zeta$, where $\zeta > 0$ is a constant, we choose ℓ appropriately, such that, $\mathbf{Pr}[\mathit{CorrMaj}(R) | w \in W_a] \approx 1$ (i.e., $1 - O(\frac{1}{n^c})$, for some $c > 0$) and $\mathbf{Pr}[\mathit{CorrMaj}(R) | w \notin W_a]$ becomes very small (i.e., $O(\frac{1}{n^c})$, for some $c > 0$). Hence $\mathbb{E}[Z^1]$ approximated suitably enough $f_a = \mathbf{Pr}[w \in W_a]$. Lines 4–15 for algorithm E_1 implements the *SRA* algorithm to estimate $\mathbb{E}[Z^1]$.

Z^2 is defined as follows: the master randomly picks a worker w from W, assigns a challenge to w, and verifies the reported result. If the result is correct then $Z^2 = 1$, otherwise $Z^2 = 0$. Note that

$$\begin{aligned}\mathbb{E}[Z^2] &= \mathbf{Pr}[w \in W_a] \cdot \mathbf{Pr}[\mathit{result\ is\ correct} | w \in W_a] \\ &\quad + \mathbf{Pr}[w \notin W_a] \cdot \mathbf{Pr}[\mathit{result\ is\ correct} | w \notin W_a] \\ &= f_a(1-\epsilon) + (1-f_a)\epsilon.\end{aligned}$$

Lines 16–24 for algorithm E_1 implement the *SRA* algorithm to estimate $\mathbb{E}[Z^2]$.

Algorithm 6 Algorithm E_1 to estimate f_a, ϵ, and $f_a(1-\epsilon) + (1-f_a)\epsilon$.

```
1:  Let $\delta = \frac{1}{n^c}$ and $\varepsilon = \frac{1}{n^c}$ for $c > 0$
2:  Let $\Gamma = (4\lambda \log(\frac{2}{\delta}))/\varepsilon^2$ and $\Gamma_1 = 1 + (1+\varepsilon)\Gamma$
3:  Let $\ell = \lceil k \log n \rceil$, for some $k > 0$
4:  $N \leftarrow 0, S \leftarrow 0$
5:  while $S < \Gamma_1$ do
6:      $N \leftarrow N + 1$
7:      pick a worker $w$ randomly uniformly from $W$
8:      for $i = 1$ to $\ell$ do
9:          send challenge task $t_i$ to $w$
10:         $R[i] \leftarrow$ result received from $w$
11:     end for
12:     if $\mathit{CorrMaj}(R)$ then $Z^1_N \leftarrow 1$ else $Z^1_N \leftarrow 0$ end if
13:     $S \leftarrow S + Z^1_N$
14: end while
15: $\tilde{p} \leftarrow \frac{\Gamma_1}{N}$
16: $N \leftarrow 0, S \leftarrow 0$
17: while $S < \Gamma_1$ do
18:     $N \leftarrow N + 1$
19:     pick a worker $w$ randomly uniformly from $W$
20:     send challenge task to $w$
21:     if result received from $w$ is correct then $Z^2_N \leftarrow 1$ else $Z^2_N \leftarrow$
        0 end if
22:     $S \leftarrow S + Z^2_N$
23: end while
24: $\tilde{q} \leftarrow \frac{\Gamma_1}{N}$
25: return $\left(\tilde{p}, \frac{\tilde{q}-\tilde{p}}{1-2\tilde{p}}, \tilde{q}\right)$
```

Analysis of the algorithm: In the following theorem we state that E_1 provides suitable approximation for the different parameters of the system.

Theorem 5.2: The estimates $\tilde{p}$, $\frac{\tilde{q}-\tilde{p}}{1-2\tilde{p}}$, and $\tilde{q}$ returned by E_1 are (ε, δ)-approximations of the parameters f_a, ϵ, and $f_a(1-\epsilon) + (1-f_a)\epsilon$, respectively, where $\varepsilon, \delta \in O(\frac{1}{n^\gamma})$, for some $\gamma > 0$.

Theorem 5.3: The number of rounds or the work for algorithm E_1 is $n^c \log n$ for $c > 0$ *whp*.

Proof: The number of times the **while** loop iterates is the value of N at the end of the looping. The value of N at the end of any of the **while** loop is $n^c \log n$ *whp*, which can be proved by substituting $\frac{1}{n^c}$ for δ and ε in the Γ_1 (see Algorithm 6) and applying Theorem 5.1(iii). Now, in the first **while** the **for** loop runs for $\Theta(\log n)$ iterations. Therefore, the rounds and work are $O(n^c \log^2 n)$, *whp*. ■

VI. Conclusions and Future Work

This work considers the master-worker paradigm for Internet-based computation of tasks, in the presence of altruistic and troll workers subject to an error probability that can alter their intended behavior. A generic model capturing the parameters of the master-worker paradigm is presented; deviating from the usual conventions made in the literature. The error probability that workers might have and moving away from the usual assumption that a task can only have one correct and one incorrect result made also our modelling more realistic.

There are still many aspects of the system that are not captured by our proposed model. For example, we do not consider the possibility that workers might be unavailable at stages in the execution, or that they may express dynamic behavior by experiencing different error probability over time. Moreover, it would be interesting to study algorithms that adapt when each task has a different number of correct and incorrect results. We have assumed for simplicity of presentation that ϵ is the same for all workers, this assumption also helped to gain a better understanding of our master-worker setting. In the future we plan to remove this assumption and consider that each worker has it's own error probability that can even change over time. Thus, we will adapt our algorithms to reputation techniques designed to help us estimate the reliability of each worker. Such additional considerations will allow our model to capture more complex environments like, for example, the behavior of crowdsourcing systems.

References

[1] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4–10. IEEE, 2004.

[2] D. P. Anderson. Volunteer computing: the ultimate cloud. *ACM Crossroads*, 16(3):7–10, 2010.

[3] D. P. Anderson. BOINC, 2016. http://boinc.berkeley.edu/.

[4] D. M. Blough and G. F. Sullivan. A comparison of voting strategies for fault-tolerant distributed systems. In Proc. of *RDS'90*, pages 136–145, 1990.

[5] G. Casella and R. L. Berger. *Statistical Inference*. Duxbury Advanced Series, second edition, 2001.

[6] P. Dagum, R.M. Karp, M. Luby, and S. Ross. An optimal algorithm for monte carlo estimation. In *Proceedings of the Foundations of Computer Science*, pages 142–149, 1995.

[7] W. de Zutter. BOINC stats, 2016. http://boincstats.com/en/forum/10/4597.

[8] A. Fernández, C. Georgiou, L. López, and A. Santos. Reliably executing tasks in the presence of malicious processors. In *Distributed Computing*, pages 490–492. Springer, 2005.

[9] A. Fernández Anta, C. Georgiou, L. López, and A. Santos. Reliable internet-based master-worker computing in the presence of malicious workers. *Parallel Processing Letters*, 22(01), 2012.

[10] E. M. Heien, D. P Anderson, and K. Hagihara. Computing low latency batches with unreliable workers in volunteer computing environments. *Journal of Grid Computing*, 7(4):501–518, 2009.

[11] D. Kondo, F. Araujo, P. Malecot, P. Domingues, L. M. Silva, G. Fedak, and F. Cappello. Characterizing result errors in internet desktop grids. In *Euro-Par 2007 Parallel Processing*, pages 361–371. Springer, 2007.

[12] K. M. Konwar, S. Rajasekaran, and A. A. Shvartsman. Robust network supercomputing with unreliable workers. *J. Parallel Distrib. Comput.*, 75:81–92, 2015.

[13] E. Korpela, D. Werthimer, D. P Anderson, J. Cobb, and M. Lebofsky. Seti@ homemassively distributed computing for seti. *Computing in science & engineering*, 3(1):78–83, 2001.

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

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

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

[17] J. Sonnek, M. Nathan, A. Chandra, and J. Weissman. Reputation-based scheduling on unreliable distributed infrastructures. In *IEEE ICDCS06*, pages 30–30, 2006.

[18] S. Zhao, V. Lo, and C G. Dickey. Result verification and trust-based scheduling in peer-to-peer grids. In *IEEE P2P05*, 2005.