Abstract—Internet supercomputing is an approach to solving partitionable, computation-intensive problems by harnessing the power of a vast number of interconnected computers. For the problem of using network supercomputing to perform a large collection of independent tasks, prior work introduced a decentralized approach and provided randomized synchronous algorithms that perform all tasks correctly with high probability, while dealing with misbehaving or crash-prone processors. The main weaknesses of existing algorithms is that they assume either that the average probability of a non-crashed processor returning incorrect results is inferior to \(\frac{1}{2}\), or that the probability of returning incorrect results is known to each processor. Here we present a randomized synchronous distributed algorithm that tightly estimates the probability of each processor returning correct results. Starting with the set \(P\) of \(n\) processors, let \(F\) be the set of processors that crash. Our algorithm estimates the probability \(p_i\) of returning a correct result for each processor \(i \in P-F\), making the estimates available to all these processors. The estimation is done with the help of “test tasks,” as failures of computers and network links, thus it is also necessary to consider fully distributed peer-to-peer solutions.

Interestingly, worker computers returning bogus results is a phenomenon of increasing concern. While this may certainly occur unintentionally, e.g., as a result of over-clocked processors, workers may in fact deceitfully claim to have performed assigned work so as to obtain incentives associated with the system, e.g., a higher rank. To address this problem, several works, e.g., [5], [12], [13], [17], study approaches based on a reliable master dealing with a collection of unreliable workers. The drawback in these approaches is the reliance on a reliable, bandwidth-unlimited master processor.

In our recent work [8], [9], [10] we began to address this drawback of centralized systems by removing the assumption of an infallible and powerful master processor. We introduced a decentralized approach, where a collection of worker processors cooperates on a large set of independent tasks without the reliance on central control. Our synchronous algorithms in [9], [8] are able to perform all tasks with high probability (whp), while dealing with misbehaving processors under a rather strong assumption that the average probability of live (non-crashed) processors returning incorrect results remains inferior to \(\frac{1}{2}\) during the computation. In [9] we considered a linearly bounded adversary, where the adversary is constrained so that a constant fraction of workers never fails. Our algorithm in [8] works additionally for a fractional polynomial constrained adversary and a poly-log constrained adversary. The adversary may assign arbitrary constant probabilities to processors, provided that the processors remaining in the computation return bogus results with the average probability inferior to \(\frac{1}{2}\). Thus in [9] and in [8], the adversary is severely limited in its ability to crash processors that normally return correct results. To address this limitation, in [10] we considered a linearly bounded model where the average probability of non-crashed processors returning bogus results can become greater than \(1/2\). However, the algorithm in [10] assumes that every processor knows the probability of returning an incorrect result for all processors. This is a very strong assumption, thus it is important to develop decentralized algorithms that can efficiently estimate these probabilities in the setting of cooperative distributed computation.

Contributions. For the general setting of network supercomputing we consider the problem of estimating the probability of each participating processor performing a task correctly. The requirement here is that these estimates are computed efficiently in a distributed system of \(n\) workers without centralized control. The estimation is done with the help of “test tasks,”

I. INTRODUCTION

Cooperative network supercomputing is becoming increasingly popular for harnessing the power of the global Internet computing platform. A typical Internet supercomputer, e.g., [1], [2], consists of a master computer and a large number of computers called workers, performing computation on behalf of the master. Despite the simplicity and benefits of a single master approach, as the scale of such computing environments grows, it becomes unrealistic to assume the existence of the infallible master that is able to coordinate the activities of multitudes of workers. Large-scale distributed systems are inherently dynamic and are subject to perturbations, such as failures of computers and network links, thus it is also necessary to consider fully distributed peer-to-peer solutions.

Estimating Reliability of Workers for Cooperative Distributed Computing

Seda Davtyan* Kishori M. Konwar† Alexander A. Shvartsman*

Abstract—Internet supercomputing is an approach to solving partitionable, computation-intensive problems by harnessing the power of a vast number of interconnected computers. For the problem of using network supercomputing to perform a large collection of independent tasks, prior work introduced a decentralized approach and provided randomized synchronous algorithms that perform all tasks correctly with high probability, while dealing with misbehaving or crash-prone processors. The main weaknesses of existing algorithms is that they assume either that the average probability of a non-crashed processor returning incorrect results is inferior to \(\frac{1}{2}\), or that the probability of returning incorrect results is known to each processor. Here we present a randomized synchronous distributed algorithm that tightly estimates the probability of each processor returning correct results. Starting with the set \(P\) of \(n\) processors, let \(F\) be the set of processors that crash. Our algorithm estimates the probability \(p_i\) of returning a correct result for each processor \(i \in P-F\), making the estimates available to all these processors. The estimation is done with the help of “test tasks,” as failures of computers and network links, thus it is also necessary to consider fully distributed peer-to-peer solutions.

Interestingly, worker computers returning bogus results is a phenomenon of increasing concern. While this may certainly occur unintentionally, e.g., as a result of over-clocked processors, workers may in fact deceitfully claim to have performed assigned work so as to obtain incentives associated with the system, e.g., a higher rank. To address this problem, several works, e.g., [5], [12], [13], [17], study approaches based on a reliable master dealing with a collection of unreliable workers. The drawback in these approaches is the reliance on a reliable, bandwidth-unlimited master processor.

In our recent work [8], [9], [10] we began to address this drawback of centralized systems by removing the assumption of an infallible and powerful master processor. We introduced a decentralized approach, where a collection of worker processors cooperates on a large set of independent tasks without the reliance on central control. Our synchronous algorithms in [9], [8] are able to perform all tasks with high probability (whp), while dealing with misbehaving processors under a rather strong assumption that the average probability of live (non-crashed) processors returning incorrect results remains inferior to \(\frac{1}{2}\) during the computation. In [9] we considered a linearly bounded adversary, where the adversary is constrained so that a constant fraction of workers never fails. Our algorithm in [8] works additionally for a fractional polynomial constrained adversary and a poly-log constrained adversary. The adversary may assign arbitrary constant probabilities to processors, provided that the processors remaining in the computation return bogus results with the average probability inferior to \(\frac{1}{2}\). Thus in [9] and in [8], the adversary is severely limited in its ability to crash processors that normally return correct results. To address this limitation, in [10] we considered a linearly bounded model where the average probability of non-crashed processors returning bogus results can become greater than \(1/2\). However, the algorithm in [10] assumes that every processor knows the probability of returning an incorrect result for all processors. This is a very strong assumption, thus it is important to develop decentralized algorithms that can efficiently estimate these probabilities in the setting of cooperative distributed computation.

Contributions. For the general setting of network supercomputing we consider the problem of estimating the probability of each participating processor performing a task correctly. The requirement here is that these estimates are computed efficiently in a distributed system of \(n\) workers without centralized control. The estimation is done with the help of “test tasks,”

I. INTRODUCTION

Cooperative network supercomputing is becoming increasingly popular for harnessing the power of the global Internet computing platform. A typical Internet supercomputer, e.g., [1], [2], consists of a master computer and a large number of computers called workers, performing computation on behalf of the master. Despite the simplicity and benefits of a single master approach, as the scale of such computing environments grows, it becomes unrealistic to assume the existence of the infallible master that is able to coordinate the activities of multitudes of workers. Large-scale distributed systems are inherently dynamic and are subject to perturbations, such as failures of computers and network links, thus it is also necessary to consider fully distributed peer-to-peer solutions.

Interestingly, worker computers returning bogus results is a phenomenon of increasing concern. While this may certainly occur unintentionally, e.g., as a result of over-clocked processors, workers may in fact deceitfully claim to have performed assigned work so as to obtain incentives associated with the system, e.g., a higher rank. To address this problem, several works, e.g., [5], [12], [13], [17], study approaches based on a reliable master dealing with a collection of unreliable workers. The drawback in these approaches is the reliance on a reliable, bandwidth-unlimited master processor.

In our recent work [8], [9], [10] we began to address this drawback of centralized systems by removing the assumption of an infallible and powerful master processor. We introduced a decentralized approach, where a collection of worker processors cooperates on a large set of independent tasks without the reliance on central control. Our synchronous algorithms in [9], [8] are able to perform all tasks with high probability (whp), while dealing with misbehaving processors under a rather strong assumption that the average probability of live (non-crashed) processors returning incorrect results remains inferior to \(\frac{1}{2}\) during the computation. In [9] we considered a linearly bounded adversary, where the adversary is constrained so that a constant fraction of workers never fails. Our algorithm in [8] works additionally for a fractional polynomial constrained adversary and a poly-log constrained adversary. The adversary may assign arbitrary constant probabilities to processors, provided that the processors remaining in the computation return bogus results with the average probability inferior to \(\frac{1}{2}\). Thus in [9] and in [8], the adversary is severely limited in its ability to crash processors that normally return correct results. To address this limitation, in [10] we considered a linearly bounded model where the average probability of non-crashed processors returning bogus results can become greater than \(1/2\). However, the algorithm in [10] assumes that every processor knows the probability of returning an incorrect result for all processors. This is a very strong assumption, thus it is important to develop decentralized algorithms that can efficiently estimate these probabilities in the setting of cooperative distributed computation.

Contributions. For the general setting of network supercomputing we consider the problem of estimating the probability of each participating processor performing a task correctly. The requirement here is that these estimates are computed efficiently in a distributed system of \(n\) workers without centralized control. The estimation is done with the help of “test tasks,”

* Department of Computer Science & Engineering, University of Connecticut, 371 Fairfield Way, Unit 4155, Storrs CT 06269, USA. Emails: [sedaaas]@engr.uconn.edu.
† University of British Columbia, Vancouver, BC V6T 1Z3, CANADA. Email: kishori@interchange.ubc.ca.
i.e., tasks whose results are known to a processor that needs to obtain these estimates. Each test task can be performed by any worker in constant time. For the initial set \( P \) of \( n \) processors, we assume that every processor \( i \in P \) is given a distinct set of test tasks \( T_{ti} \), for which it knows the correct results, and others do not. Processors communicate via a synchronous fully-connected message-passing system. We deal with failure models where workers can return incorrect results and can crash. In particular, the adversary assigns to each processor \( i \) the probability \( p_i \) of returning correct results (incorrect results are returned with probability \( 1 - p_i \)). We present a randomized decentralized algorithm that estimates, for each processor \( i \in P \), the probability \( \tilde{p}_i \) of performing correct tasks, and such estimates are computed by all processors. The estimates are calculated using the \((\epsilon, \delta)\)-approximation, for \( 0 < \epsilon < 1 \) and \( \delta > 0 \), that estimates the mean of a random variable. For the given \( \delta > 0 \) and \( \epsilon > 0 \) chosen by the user, the algorithm obtains estimates \( \tilde{p}_i \) that obey the following bound: \( \Pr[\tilde{p}_i(1 - \epsilon) \leq \tilde{p}_i \leq \tilde{p}_i(1 + \epsilon)] > 1 - \delta \). We analyze our algorithm and assess its time, work, and message complexities. In additional detail our contributions are as follows.

1. We formulate the following model of adversary. Given the initial set of processors \( P \), with \( |P| = n \), the adversary assigns arbitrary constant positive probability \( p_i \) of performing correct tasks correctly to each processor \( i \in P \). Additionally, the adversary can crash any set of processors, subject to one of the three constraints: 1) The adversary is constrained by a linear fraction, where \( |P - F| \geq hn \), with \( 1 < h < 1 - f \) and \( f \in (0, 1) \). 2) The adversary is constrained by a fractional polynomial, where \( |P - F| = \Omega(n^a) \), for a constant \( a \in (0, 1) \). 3) The adversary is constrained by a log-polynomial, where \( |P - F| = \Omega(\log^c n) \), for a constant \( c \geq 1 \). (Constraints (2) and (3) are as in [8].

2. We present a randomized algorithm for \( n \) processors to compute the probabilities \( p_i \). The algorithm works in synchronous rounds, where each processor asks some other processor to perform a test task and return the result. It then shares its knowledge of results with one randomly chosen processor. Once a processor accumulates a "sufficient" number of results, it becomes "enlightened." Enlightened processors then "profess" their knowledge by multicasting it to a random, exponentially growing subset of processors. When a processor receives a message telling it that "enough" gossip was done, it halts. The values that control "sufficient" numbers of results and "enough" gossiping are established in our analysis and are used as compile-time constants.

We consider the protocol, by which the "enlightened" processors "profess" their knowledge and reach termination, to be of independent interest. The protocol’s message complexity does not depend on crashes, and termination does not require explicit coordination. This addresses the challenge of termination when \( P - F \) can vary broadly in the considered three models.

3. Our analysis shows that in each model all live processors estimate the probability \( p_i \) for every processor \( i \in P - F \) using the \((\epsilon, \delta)\) approximation, \( \text{whp} \). Complexity results for the algorithm also hold \( \text{whp} \):

- For the linearly bounded model we show that work complexity \( W(n) \) is \( O(n \log n) \), message complexity \( M(n) \) is \( O(n \log^2 n) \), and time complexity \( T(n) \) is \( O(\log n) \).

- For the polynomially constrained model we show that \( W(n) = O(n \log n \log \log n) \), \( M(n) = O(n \log^2 n \log \log n) \), and \( T(n) = O(n^{1 - a} \log n \log \log n) \).

- For the poly-log constrained model we show that \( W(n) = O(n \log \log n) \), \( M(n) = O(n \log^2 n \log \log n) \), and \( T(n) = O(n) \).

The work complexity results show that the algorithm is efficient, e.g., if \( O(n \log \log n) \) real tasks are to be done after the estimation, then the estimation expense is amortized.

Finally we note that the \((\epsilon, \delta)\)-approximation is rarely seen in distributed computing literature, and we consider showing the relevance of this technique, and bringing it to the attention of researchers in distributed computing, to be among the contributions of this work.

Prior/Related Work. Earlier approaches explored ways of improving the quality of the results obtained from untrusted workers in the settings where an infallible master is coordinating the workers. Fernandez et al. [13], [12] and Konwar et al. [17] present algorithms that help the master determine correct results \( \text{whp} \), while minimizing work. Additionally, [17] provides efficient algorithms that can estimate the probability of processors returning incorrect results. However, they assume that this probability is the same for every processor. The failure models assume that some fraction of processors can exhibit faulty behavior. Another recent work by Christoforou et al. [5] pursues a game-theoretic approach. Fernandez et al. [14] studied the master-worker model with message loss and delay in addition to assuming that processors can return incorrect results; they give algorithms with exact bounds on work and expected work. Paquette and Pelc [18] consider a fault-prone system in which a decision has to be made on the basis of unreliable information, and design a deterministic strategy that leads to a correct decision \( \text{whp} \).

As already mentioned, our prior work [8], [9] introduced the decentralized approach that eliminates the master, and provided a synchronous algorithm that is able to perform all tasks \( \text{whp} \), while dealing with incorrect behaviors under a very strong assumption that the average probability of non-crashed processors returning incorrect results remains inferior to \( \frac{1}{2} \).

The \((\epsilon, \delta)\)-approximation has been applied to a wide range of difficult scientific problems. For example, it has been successfully applied for approximation of probabilistic inference in Bayesian networks [7], solving Ising model problems in statistical mechanics [15], estimation of convex bodies [11], and estimating the number of solutions to a DNF formula [16]. We refer the reader to [6] for a broader list of references.

Document structure. In Section II we present the model and measures of efficiency. Our algorithm is given in Section III. In Section IV we discuss the estimation techniques. In Section V we analyze the algorithm and derive complexity bounds. We conclude in Section VI with a discussion.
II. Model of Computation and Definitions

System model. There are \( n \) processors, each with a unique identifier (id) from set \( P = [n] \). We refer to the processor with \( i \) as processor \( i \). The system is synchronous and processors communicate by exchanging reliable messages. Computation is structured in terms of synchronous steps, where in each step a processor can send messages, receive messages, and/or perform local polynomial computation, where the local computation time is assumed to be negligible compared to message latency. Messages received by a processor in a given step include all messages sent to it in the previous step.

Tasks. Ultimately the cooperating processors must perform tasks. Each task can be performed locally by any processor. The tasks are (a) similar, meaning that any task can be done in constant time, (b) independent, meaning that each task can be performed independently of other tasks, and (c) idempotent, meaning that the tasks admit at-least-once semantics and can be performed concurrently. To avoid misrepresentation of results, we assume that once a processor performs a task, it unforgeably signs the result. This is not discussed further.

Models of adversity. Processors are dependable in that a processor may compute the results of tasks incorrectly and it may crash. Following a crash, a processor performs no further actions. Otherwise, each processor adheres to the protocol established by the algorithm it executes. Messages can be sent to crashed processors, but they are neither delivered nor a crashed processor responds. Thus a crash can be detected if an expected response does not arrive. We refer to non-crashed processors as live.

We consider an oblivious adversary that, prior to the computation, (a) assigns an arbitrary constant probability \( p_i > 0 \) of returning a correct result for each processor \( i \in P \), and (b) decides what processors to crash and when to crash them. For an execution of an algorithm, let \( F \) be the set of processors that adversary crashes; the number of processors that can crash is established by the following adversarial models.

Model \( F_p \), adversary constrained by a fraction of the processors in \( P \): \( |P - F| \geq h n \), where \( 1 < h < 1 - f \) and \( f \in (0, 1) \), such that, up to \( |f| P \) processors can be crashed.

Model \( F_{p, c} \), adversary constrained by a fractional polynomial: \( |P - F| = \Omega(n^c) \), for a constant \( c \in (0, 1) \).

Model \( F_{p, t} \), poly-log constrained adversary: \( |P - F| = \Omega(\log^t n) \), for a constant \( t \geq 1 \).

Measures of efficiency. We assess the efficiency of algorithms in terms of time \( T(n) \), work \( W(n) \), and message \( M(n) \) complexities. We use the conventional measures of time complexity, assessed as the maximum number of steps executed by any processor, and work complexity, assessed as the total number of steps executed by all \( n \) processors. We assess message complexity as the number of point-to-point messages sent during the execution. Lastly, we use the common definition of an event \( E \) occurring with high probability (wphp) to mean that \( \Pr[E] = 1 - O(n^{-\alpha}) \) for some constant \( \alpha > 0 \).

III. Algorithm Description

We now present our decentralized algorithm \( A_{est} \) that employs no master and instead uses a gossip-based approach to share information. The algorithm is structured in terms of the main loop that iterates through three stages: QUERY, RESPONSE, and GOSSIP. Each stage consists of three steps, Send, Receive, and Compute, that are executed synchronously by the processors. In the QUERY stage each processor sends, receives, and performs test tasks. During the RESPONSE stage the processor replies with the results for the test tasks, if any, and collects such results sent by other processors. If enough information is gathered, the processor becomes “enlightened.” In the GOSSIP stage each processor gossips the collected results to other processor, except that enlightened processors “profess” their results to an exponentially growing random set of processors. The processors then update their local knowledge based on the received messages, and, if sufficient information was propagated, compute the estimates for the probabilities \( p_j \) and halt. The pseudocode for algorithm \( A_{est} \) is given in Figure 1; the algorithm uses subroutine Estimation() to compute the probabilities, given in Figure 2. We next describe the algorithm in greater detail.

Inputs. Each processor \( i \) receives as inputs the number of processors \( n \), the estimation parameters \( c \) and \( \delta \), and the set of test tasks \( TT_i \) from its environment.

Output. Each processor \( i \) outputs the estimates of probabilities \( p_j \) for each \( j \in P \) in array \( \text{Estimate}_i[1..n] \). If a crash of processor \( j \) is detected, \( \text{Estimate}_i[j] \) is set to \(-1\).

Local knowledge and state variables. Every processor \( i \) maintains the following:

- Array \( R_i[1..n] \) stores results of test tasks, where element \( R_i[j] \) is a set of results of test tasks done by processor \( j \). Each \( R_i[j] \) is a set of tuples \((v, s, r)\) representing the correctness of the result \( v \in \{0, 1, -1\} \) computed by processor \( j \) on behalf of processor \( s \), in round \( r \). (This ensures that results computed by processor \( j \) in different rounds \( r \) and for different processors \( s \) are included.) The value \( v = 0 \) means that the result was computed incorrectly, \( v = 1 \) means that it was computed correctly, and \( v = -1 \) means that processor \( j \) has not returned a result, hence, per our model assumption, it crashed.
- \( r \) is the round (iteration) number that is used to timestamp the computed results.
- \( \ell \) controls the number of messages multicast by enlightened processors: the multicast is sent to \( 2^{\ell+1} - 1 \) destinations. The value of \( \ell \) is also used to “prioritize” processors, where higher values of \( \ell \) correspond to higher priority, with ties broken by the processor identifiers. That is, given two distinct processors \( i \) and \( j \) we say that processor \( j \) has higher priority than \( i \) if \((\ell_i, i) < (\ell_j, j)\), where \((\cdot, \cdot)\) is a lexicographic comparison, i.e., \((\ell_i, i) < (\ell_j, j)\) if and only if either (i) \( \ell_i < \ell_j \), or (ii) \( \ell_i = \ell_j \) and \( i < j \).
- \( \text{enlightened} \) is a boolean that determines whether the processor has enough information to start “professing” its knowledge by means of aggressive gossip.
procedure for processor $i$:

input $n$, /* $n$ is number of processors */ $\epsilon, \delta, r$ /* $\epsilon > 0$ and $\delta > 0$ are estimation parameters */ $TT_i$, /* the set of test tasks for $i$ */

output $\text{Estimate}_{\epsilon}[1..n]$ init $\perp$ /* array of estimates of $p_j$ for each $j \in P$ */

$R_i[1..n]$ init $\emptyset$ /* set of collected result indicators $\langle \text{res}, \text{src}, \text{rnd} \rangle$ */

int $r$ init 0 /* round number */

int $\ell$ init /* specifies the number of processor messages to be sent per iteration */

bool enlightened init false /* indicates whether the processor is "enlightened" */

while true do

QUERY STAGE

Send:
1. Let $q$ be a randomly selected processor id from $P$
2. Let $t$ be a randomly selected task from $TT_i$
3. Send $\langle t, i, r \rangle$ to processor $q$

Receive:
4. Let $M = \{ m : m = \langle \text{task}, id \rangle \}$ be the set of received messages

Compute:
5. if $|M| > \lceil \log n \rceil$ then
6. $M \leftarrow$ random selection of $\lceil \log n \rceil$ elements from $M$
7. Let $V = \{ \langle \text{val}, id \rangle : m \in M \land \text{val} = \text{result of } m.\text{task} \land id = m.\text{id} \}$

RESPONSE STAGE

Send:
8. for each $w \in V$
9. Send $\langle w.\text{val}, i \rangle$

Receive:
10. if message $\langle \text{val} \rangle$ is received from $q$ chosen in QUERY STAGE then
11. if $\text{val}$ is the correct result for task $t$ chosen in QUERY STAGE then
12. $R_i[q] \leftarrow R_i[q] \cup \langle t, i, r \rangle$ /* test task was computed correctly */
13. else
14. $R_i[q] \leftarrow R_i[q] \cup \langle 0, i, r \rangle$ /* test task was computed incorrectly */
15. else /* no response from processor $q$ */
16. $R_i[q] \leftarrow R_i[q] \cup \langle -1, i, r \rangle$ /* $-1$ is used to record a crash */

Compute:
17. if $\forall j \in P : (\sum_{s \in R_i[j]} I_{\langle t, i, r \rangle} (x.\text{res}) \geq T_1) /* sufficient no. of correct results */$
18. $\forall \exists x \in R_i[j] : x.\text{res} = -1$ /* or $j$ crashed */
19. enlightened $\leftarrow$ true /* processor becomes enlightened */

GOSSIP STAGE

Send:
20. if enlightened then /* gossip aggressively */
21. Let $D$ be a set of $2^{\lceil \log n \rceil}$ processor ids randomly selected from $P$
22. Send $\langle \text{profess}, R_i[\cdot], \ell, i \rangle$ to processors in $D$
23. $\ell \leftarrow \ell + 1$
24. else
25. Let $q$ be a randomly selected processor id from $P$
26. Send $\langle \text{share}, R_i[\cdot], \ell, i \rangle$ to processor $q$

Receive:
27. Let $M = \{ m : m = \langle \text{type}, R, \ell, id \rangle \}$ be the set of received messages
28. if $\exists m \in M : m.\text{type} = \text{profess}$ then
29. enlightened $\leftarrow$ true /* processor becomes enlightened */
30. if $\exists m \in M : \langle \ell, i \rangle \prec (m.\ell, m.\text{id})$ then
31. $\ell \leftarrow 0$

Compute:
32. for each $j \in P$
33. $R_i[j] \leftarrow R_i[j] \cup \bigcup_{m \in M} m.\text{R}[j]$
34. if $\exists m \in M : m.\ell \geq \lceil \log n \rceil$ then
35. $\text{Estimate}(R_i[\cdot], \text{Estimate}_{\epsilon}[\cdot])$ /* Compute the estimates and store in \text{Estimate}_{\epsilon}[1..n] */
36. halt
37. $r \leftarrow r + 1$

Fig. 1. Algorithm $A_{est}$ at processor $i$ for $i \in P$.

Control flow. We refer to each iteration of the main while-loop as the round. The loop is synchronous, but each processor exits the loop based on its local state, thus the loop may not terminate simultaneously; to model this we let the loop iterate forever and include an explicit halt for each processor $i$. Next we detail each of the three stages within a round. Recall that each stage is comprised of three steps.

QUERY stage:

Send step: Processor $i \in P$ selects at random a target processor $q \in P$ and a task $t \in TT_i$ and sends the request containing task $t$ to $q$.

Receive step: The processor receives the requested tasks sent to it in the preceding step (if any).

Compute step: If the number of tasks requested is less
The subroutine is used for calculating the estimation of the algorithm that although the algorithm performs the estimation (if any) from processor \(q\) tasks to the respective requesters. If the result for the test task is correct then processor \(i\) adds \((1, i, r)\) to \(R_i[q]\), otherwise it adds \((0, i, r)\). If, however, it does not receive a message from \(q\) then it adds \((-1, i, r)\) to \(R_i[q]\), where \(-1\) indicates that processor \(q\) crashed.

**Compute step**: Processor \(i\) uses the values in \(R_i[]\) to check whether it gathered a certain number of results (in the analysis we will show that this is sufficient for computing the \((\epsilon, \delta)\)-approximation). If so, the processor becomes enlightened. This is done with the help of the function call \(I_1(x, \text{res})\) in line 17. The function \(I_A : \mathbb{N} \rightarrow \{0, 1\}\) is the indicator function, such that \(I_A(x)\), for any set \(A \subseteq \mathbb{N}\), returns value 1 if \(x \in A\) and 0 otherwise (this is also used in the analysis).

**Gossip stage**: If processor \(i\) is enlightened, it aggressively gossips its knowledge by professing it to an exponentially growing random set of processors. The size of the set is governed by the exponent \(\ell\) that is incremented in each round. Otherwise the processor shares its knowledge with one randomly chosen processor.

**Receive step**: Processor \(i\) receives messages. If it receives a profess message, it also becomes enlightened. Additionally, if a profess message is received from a processor with a higher priority (as determined by the lexicographic comparison in line 30) the processor sets \(\ell\) to 0.

**Compute step**: Processor \(i\) updates its knowledge in \(R_i[]\) by including the information gathered from the received messages. If processor \(i\) receives a message \(m\) such that \(m, \ell \geq \lceil \log n \rceil\), then it calls the \(\text{Estimation()}\) procedure to compute the needed probability estimates and halts. Otherwise processor \(i\) increments \(r\) and moves to the next round.

```

Fig. 2. Estimation of the probabilities for each \(j \in P\).

\textbf{Estimation()} subroutine: The subroutine is used for calculating an estimate \(\hat{p}_j\) of probability \(p_j\) for every processor \(j \in P\) and storing the result in \(\text{Estimate}[j]\). For a processor \(j\) whose crash was detected (as the result of the lack of message response), we set \(\text{Estimate}[j] = -1\). In the next section we discuss the rationale behind the estimation computation and the choice of parameters \(\Gamma\) and \(\Gamma_1\). The estimate \(\hat{p}_j\) is calculated as follows. First the tuples in \(R[j]\) are sorted according to the round number, then the sum of the first \(N\) result correctness indicators (recall that 1 means correct, 0 means incorrect) is computed for the largest \(N\) such that the sum remains inferior to \(\Gamma_1\). The estimate \(\hat{p}_j\) is then computed as \(\frac{1}{N}\).

**IV. Estimation of the \(p_i\)**

Getting an \((\epsilon, \delta)\)-approximation \(\hat{p}_i\) for \(p_i\), for any \(\epsilon, \delta > 0\), where \(\text{Pr}[p(t - 1) \leq \hat{p}_i \leq p(t + 1)] > 1 - \delta\), might sound like a straight forward problem solvable by collecting a sufficient number of samples and selecting the majority as the outcome. However, such a solution is programmable if we know the required number of samples \(a priori\). In fact this number will be dependent on the value of \(p_i\), \(\epsilon\) and \(\delta\). Since the value of \(p_i\) is unknown, we want the algorithm to terminate as early as possible, once the useful computations are done, without reliance on the value of \(p_i\) as either an input or a bound. The algorithm should be able to detect if sufficient number of samples are collected on the fly to arrive at an \((\epsilon, \delta)\)-approximation. Below we explain this with an example.

Suppose we have a random variable \(X\), where \(X \in \{0, 1\}\) such that \(\text{Pr}[X = 0] = p\) and \(\text{Pr}[X = 1] = 1 - p = q\). Consider the independent and identically distributed (iid) random variables \(X_1, X_2, \ldots, X_m\) whose distribution is that of \(X\). Therefore, \(\mathbb{E}[X] = \mathbb{E}[X_1] = \ldots = \mathbb{E}[X_m] = q\). Suppose we want to use the unbiased estimator \(\frac{S_m}{m}\) of \(q\), where \(S_m = \sum_{i=1}^m X_i\). An estimator \(T(X_1, X_2, \ldots, X_m)\) of a parameter \(\theta\) is called unbiased estimator of \(\theta\) if \(\mathbb{E}[T(X_1, X_2, \ldots, X_m)] = \theta\) [4]. Let us choose \(m = c \log n\), for some \(c > 0\), in an attempt to have a reasonable number of trials.

We next state the well-known Chernoff bounds.

**Lemma 1 (Chernoff Bounds)**: Let \(X_1, X_2, \ldots, X_n\) be iid independent Bernoulli random variables with \(\text{Pr}[X_1 = 1] = p_i\) and \(\text{Pr}[X_i = 0] = 1 - p_i\), then it holds for \(X = \sum_{i=1}^n X_i\) and \(\mu = \mathbb{E}[X] = \sum_{i=1}^n p_i\) that for all \(\delta > 0\), (i) \(\text{Pr}[X \geq (1 + \delta)\mu] \leq e^{-\frac{\delta^2 \mu}{2}}\), and (ii) \(\text{Pr}[X \leq (1 - \delta)\mu] \leq e^{-\frac{\delta^2 \mu}{2}}\).

By a simple application of a slight variation of the Chernoff bounds we can show that for \(\delta > 0\)

\[
\text{Pr}\left[\frac{S_m}{m} \geq (1 + \delta)\mu\right] \leq e^{-\frac{\delta^2 \mu}{2}} \leq e^{-\frac{\delta^2 \mathbb{E}[X]}{2}} \leq n^{-\frac{c^2 \log n}{4}}
\]

A similar relation can be shown for the case where \(\text{Pr}\left[\frac{S_m}{m} \leq (1 - \delta)\mu\right] \leq n^{-\frac{c^2 \log n}{4}}\). Observe that unless we have some prior information about the value of \(q\) (or \(p\)), other than the trivial bounds \(0 \leq p \leq 1\), we may not know what \(c\) to choose to determine the number of repetitions for obtaining the desired accuracy for the estimation of \(q\). Thus it is desirable to have an algorithm that has an online rule for stopping the computation.

Subroutine \(\text{Estimation()}\) in Figure 2 is used for calculating an \((\epsilon, \delta)\)-approximation of \(p_i\) as described above. Now we elaborate on the technical aspects of \((\epsilon, \delta)\)-approximation and
Estimate sorted in ascending order of the rounds to consider the by looking at the history of the results stored in the list 

SRA

choose the constant 

\[ \sim \]

test tasks required to compute 

\[ E \]

Theorem 1 (SRA) 

Pr

input parameters:

\((\epsilon, \delta)\) with \(0 < \epsilon < 1, \delta > 0\)

1: Let \( \Gamma = 4\alpha \log \left(\frac{2}{n}\right)/\epsilon^2 \)

2: Let \( \Gamma_1 = 1 + (1 + \epsilon)\Gamma \)

3: initialize \( N \leftarrow 0, S \leftarrow 0 \)

4: while \( S < \Gamma_1 \) do \( N \leftarrow N + 1; S \leftarrow S + Z_N \)

5: output: \( \hat{\mu}_Z \leftarrow \frac{N}{S} \)

Fig. 3. The Stopping Rule Algorithm (SRA) for estimating \( \mu_Z \).

determine the value of \( \delta \) for our analysis to hold \( \text{whp} \). For every processor \( i \in P - F \) we further bound the number of test tasks required to compute \( \bar{p}_i \).

The idea behind subroutine Estimation() is based on the Stopping Rule Algorithm (SRA) of Dagum et al. [6]. For completeness we reproduce in Figure 3 this well-known algorithm for estimating the mean of a random variable with support in \([0, 1]\), with \((\epsilon, \delta)\)-approximation. Let \( Z \) be a random variable distributed in the interval \([0, 1]\) with mean \( \mu_Z \). Let \( Z_1, Z_2, \ldots \) be independently and identically distributed according to \( Z \) variables. We say the estimate \( \hat{\mu}_Z \) is an \((\epsilon, \delta)\)-approximation of \( \mu_Z \) if \( \Pr[|\mu_Z - \hat{\mu}_Z| \leq \epsilon \mu_Z] \geq 1 - \delta \).

Let us define \( \lambda = (e - 2) \approx 0.72 \) and \( \Gamma = 4\alpha \log \left(\frac{2}{n}\right)/\epsilon^2 \). Now, Theorem 1 (slightly modified, from [6]) tells us that SRA provides us with an \((\epsilon, \delta)\)-approximation with the number of trials within \( \frac{1}{\epsilon^2} \) \( \text{whp} \), where \( \Gamma_1 = 1 + (1 + \epsilon)\Gamma \).

Theorem 1 (Stopping Rule Theorem): Let \( Z \) be a random variable in \([0, 1]\) with \( \mu_Z = \mathbb{E}[Z] > 0 \). Let \( \mu_Z \) be the estimate produced and let \( N_Z \) be the number of experiments that SRA runs with respect to \( Z \) on inputs \( e \) and \( \delta \). Then, \( \Pr[|\mu_Z - \hat{\mu}_Z| \leq \epsilon \mu_Z] \geq 1 - \delta \).

SRA computes an \((\epsilon, \delta)\)-approximation with an optimal number of samplings, within a constant factor [6], thus SRA-based method provides substantial computational savings.

First, we want to show that \( \Pr[N_Z > \left(1 + \frac{\epsilon}{2}\right)c \log n] \leq \frac{1}{n^7} \) for some \( c > 0 \) and \( \alpha > 0 \). Let us choose a \( \delta = \frac{1}{n^6} \), for some \( \alpha > 0 \), then for every \( \epsilon > 0 \) and \( \Gamma_1 = 1 + (1 + \epsilon)\Gamma \) we have

\[
\Gamma = 4\alpha \log \left(\frac{2}{\sqrt{n}}\right)/\epsilon^2 = 4\alpha \log \left(n^{\alpha}\right)/\epsilon^2 = \frac{4\alpha n^{\alpha}}{\epsilon^2}.
\]

Also, we have \( \Gamma_1 \leq (1 + \epsilon)\frac{n^{\alpha}}{\epsilon^2} \) for some \( \alpha' > 0 \). Now, using the Stopping Rule Theorem (Theorem 1) we have

\[
\frac{1}{n^7} \geq \Pr[N_Z > \left(1 + \frac{\epsilon}{2}\right)c \log n] \geq \Pr[N_Z > \left(1 + \epsilon\right)^2 \frac{4\alpha n^{\alpha}}{\epsilon^2}].
\]

We define the random variable \( X = \sum_{i=1}^{n} \) to estimate the total number of times processor \( w \) is selected in round \( r \). In line 21 processor \( q \) chooses a destination for the message uniformly at random, and hence \( \Pr[X_i = 1] = \frac{1}{n} \). Let \( \mu = \mathbb{E}[X] = \sum_{i=1}^{n} X_i = \frac{1}{n} \epsilon n \log n = c \log n, \) then by applying Chernoff bound, for some \( 1 > \delta > 0 \) we have:

\[
\Pr[X \leq (1 - \delta)\mu] \leq e^{-\frac{\epsilon^2}{2}} \leq e^{-\frac{(\log n)^2}{2n}} \leq \frac{1}{\sqrt{n^7}} \leq \frac{1}{n^7}.
\]

where \( \beta > 0 \). Hence, \( \Pr[X \leq 1] \leq \frac{1}{n^7} \). Let \( \mathbb{E}_w \) denote the fact that processor \( w \) receives a message from processor \( q \) in round \( r \), and let \( \mathbb{E}_w \) be the complement of that event. By Boole’s inequality we have \( \Pr[\mathbb{E}_w^c] \leq \sum_{i=1}^{w} \Pr[\mathbb{E}_i] \leq \frac{1}{n^7}, \) where \( \gamma = \log n - 1 > 0 \). Hence each processor \( w \in P - F \) receives at least one message from processor \( q \) in round \( r \) \( \text{whp} \), i.e., \( \Pr[\mathbb{E}_w^c] \geq 1 - \Pr[\mathbb{E}_w^c] = 1 - \Pr[\mathbb{E}_w] \geq 1 - \frac{1}{n^7}. \)

V. Complexity Analysis

Here we analyze the performance of algorithm \( A_{ext} \). We first show that if a processor becomes enlightened then every live processor terminates quickly.

Lemma 2: In algorithm \( A_{ext} \), subroutine Estimation() computes an \((1, 1/n^6)\)-approximation, for some constant \( \alpha > 0 \), of \( p_i \) for any \( i \in P \), and the number of responses from each live processor \( i \) sufficient for the estimation is \( O(\log n) \), \( \text{whp} \).

Proof: According to GOSSIP stage of the algorithm if processor \( q \) is enlightened then it starts sending profess messages. W.l.o.g. we assume that \( q \) is the processor with the highest priority among all enlightened processors. According to Compute step of GOSSIP stage (line 34) every processor halts once it receives a profess message \( m \) from some processor such that \( m.l \geq [\log n] \). Since processor \( q \) has the highest priority, once enlightened, it does not reset its \( l \) to 0, and hence in \( \Theta(\log n) \) rounds of the algorithm processor \( q \) sends \( n = cn \log n \) profess messages, where \( c \geq 1 \) is a constant. Let \( r \) be the round when processor \( q \) sends \( n \) profess messages. We want to prove that in round \( r \) every processor receives a profess message from \( q \) \( \text{whp} \). Let us assume that there exists processor \( w \) that does not receive a profess message from processor \( q \) in round \( r \). We prove that \( \text{whp} \) such a processor does not exist. Since \( n \) profess messages are sent in round \( r \), there were \( n \) random selections of processors from set \( P \) in line 21 by processor \( q \); let \( i \) be the index of one such selection. Let \( X_i \) be a Bernoulli random variable such that \( X_i = 1 \) if processor \( w \) was chosen by processor \( q \) and \( X_i = 0 \) otherwise.

We define the random variable \( X = \sum_{i=1}^{n} \) to estimate the total number of processor \( w \) in round \( r \). In line 21 processor \( q \) chooses a destination for the message uniformly at random, and hence \( \Pr[X_i = 1] = \frac{1}{n} \). Let \( \mu = \mathbb{E}[X] = \sum_{i=1}^{n} X_i = \frac{1}{n} \epsilon n \log n = c \log n, \) then by applying Chernoff bound, for some \( 1 > \delta > 0 \) we have:

\[
\Pr[X \leq (1 - \delta)\mu] \leq e^{-\frac{\epsilon^2}{2}} \leq e^{-\frac{(\log n)^2}{2n}} \leq \frac{1}{\sqrt{n^7}} \leq \frac{1}{n^7}.
\]

where \( \beta > 0 \). Hence, \( \Pr[X \leq 1] \leq \frac{1}{n^7} \). Let \( \mathbb{E}_w \) denote the fact that processor \( w \) receives a message from processor \( q \) in round \( r \), and let \( \mathbb{E}_w \) be the complement of that event. By Boole’s inequality we have \( \Pr[\mathbb{E}_w^c] \leq \sum_{i=1}^{w} \Pr[\mathbb{E}_i] \leq \frac{1}{n^7}, \) where \( \gamma = \log n - 1 > 0 \). Hence each processor \( w \in P - F \) receives at least one profess message from processor \( q \) in round \( r \) \( \text{whp} \), i.e., \( \Pr[\mathbb{E}_w^c] = \Pr[\mathbb{E}_w] = 1 - \Pr[\mathbb{E}_w^c] \geq 1 - \frac{1}{n^7}. \)
Therefore, every live processor terminates in $O(\log n)$ rounds of the algorithm $\whp$.

Next lemma shows that if a processor $q \in P - F$ is enlightened, then in each subsequent round $O(n \log n)$ messages are sent $\whp$.

**Lemma 4:** In the $\text{Send}$ step of GOSSIP stage of $A_{est}$ $O(n \log n)$ messages are sent in every round $\whp$.

**Proof:** We use induction on the round number, showing that in every round there can be at most $kn \log n$ messages for a sufficiently large constant such that $k > 8$. Unless stated otherwise, hereafter by messages we mean messages of processor type that are being sent in $\text{Send}$ step of GOSSIP stage.

The base case is the first round, say round $t_0$, in which some set of processors sets their $\text{enlightened}$ variable to true. There can be at most $n$ such processors, and according to our algorithm, after $\text{enlightened}$ is set to true for a processor, it starts with $\ell = 0$ and sends $\frac{1}{2} \log n$ messages, and hence, $O(n \log n)$ messages are sent during round $t_0$. Let $M_t$ be the set of messages sent by all processors in round $t$. Note that in round $t$ we have $|M_t| \equiv m_t \leq kn \log n$.

**Induction hypothesis:** In round $t > t_0$ we have $m_t \leq kn \log n$.

**Induction step:** We want to show that in round $t + 1$ we have $m_{t+1} \leq kn \log n$.

Consider the processors at the beginning of round $t + 1$. Observe that any message from a processor with a higher priority to the processor with a lower priority will reset $\ell = 0$ at the later processor.

Let $d_i$ denote the number of messages sent by the processor $i \in P$ in $\text{Send}$ step of GOSSIP stage of round $t$. By the construction of the algorithm $d_i = 2^d - 1 \lceil \log n \rceil$ where $d$ is the level of a processor and $0 \leq d \leq \lceil \log n \rceil$. Note that any distinct processors $i, j \in P$ can be at different levels ($\ell_i \neq \ell_j$). Let us assume that the processor $id_i$'s are ranked in the descending order of the $d_i$'s. Hereafter when we refer to the $i$th processor we mean the processor with ranking $i$, based on $d_i$.

We define random variable $X_i^t$ for each processor $i \in P$. After all messages are sent and received in round $t$ we let $X_i^t = 0$ if processor $i$ received a message from a processor $j$ with higher priority, and $X_i^t = 1$ otherwise. Let us further denote by $p_i = \Pr(X_i^t = 1)$, note that $p_0 = 1$ since the processor 0 has the highest priority. Therefore, $p_0 = 1; p_1 = (1 - \frac{1}{n})d_0; p_2 = (1 - \frac{1}{n})d_0 + d_1; \ldots; p_i = (1 - \frac{1}{n})\sum_{j=0}^{i-1}d_j$.

We define $X^t = \sum_{i=0}^{n-1} d_iX_i^t$ as a random variable that counts the number of messages that are sent during round $t + 1$. Clearly, $X^{t+1} \leq 2 \sum_{i=0}^{n-1} d_iX_i^t + n \log n$. The expected number of messages sent in round $t + 1$ is bounded by:

$$2E \sum_{i=0}^{n-1} d_iX_i^t + n \log n = 2 \sum_{i=0}^{n-1} d_iE[X_i^t] + n \log n = 2(d_0 + \sum_{i=1}^{n-1} d_i \sum_{j=0}^{i-1} d_j) + n \log n$$

where $c = c(n) = 1 - \frac{1}{n}$. Consider the descending arrangement of $d_i$'s grouped in blocks of consecutive terms as $\tilde{d}_0, \tilde{d}_1, \ldots, \tilde{d}_h, \tilde{d}_{h+1}, \ldots, \tilde{d}_{n-1}$, where each group includes a maximum number of $d_i$'s such that $\sum_{i=0}^{k-1} d_i < n \log n$, with a possible exception for the last block, where $j = 0, 1, \ldots, s$, $k_0 = 0$, and $k_{s+1} = n$. We note that at the minimum the first grouping of $d_i$'s is within the constant factor of $n \log n$, otherwise the total number of messages sent is less than $kn \log n$ and the inductive step holds for round $t + 1$. Using such blocking and the fact $c < 1$ and $d_i \geq 0$ we have

$$\sum_{j=0}^{n-1} d_i(c^{\sum_{j=0}^{i-1} d_j} - 1) \leq \sum_{j=0}^{k-1} d_i + \sum_{j=k_1}^{k_{s-1}} d_i c^{\sum_{j=0}^{k_{s-1}-1} d_j} + \ldots + \sum_{j=k_{s-1}}^{n-1} d_i c^{\sum_{j=0}^{k_{s-1}-1} d_j} \leq \sum_{j=0}^{k-1} d_i + \sum_{j=k_1}^{k_{s-1}} d_i c^{k_{s-1}} + \ldots + \sum_{j=k_{s-1}}^{n-1} d_i c^{k_{s-1}} \leq \sum_{j=0}^{k-1} d_i + \sum_{j=0}^{k_{s-1}} d_i c^{k_{s-1}} + \ldots + \sum_{j=0}^{n-1} d_i c^{k_{s-1}} \leq 2n \log n$$

since $c(n)^{n \log n} \to \frac{1}{n} \text{ as } n \to \infty$. Therefore, we have $E[X^{t+1}] \leq 2E \sum_{i=0}^{n-1} d_iX_i^t + n \log n \leq 7n \log n$.

By Chernoff bound with negative dependencies for some $\delta > 0$ we have

$$\Pr(X^{t+1} \geq (1 + \delta)E[X^{t+1}]) \leq e^{-\frac{\delta^2}{2E[X^{t+1}]}} \leq e^{-\frac{\delta^2}{2n \log n}} \leq \frac{1}{e^{\delta^2}}$$

where $\beta$ is some positive constant.

To simplify the presentation we proceed by defining the $\text{estimability}$ property, that tells us whether enough samples have been gathered.

**Definition 1:** ($\text{Estimability}$) We say that probability $p_j$ is $\text{estimable}$ for $j \in P$ in round $r$ of algorithm $A_{est}$ if, at the end of round $r$ we have $\sum_{x \in \mathcal{E}} \mathbb{E}_{X_r}(X^{t+1}) \geq 1$, or for some processor $i \in P - F$, $\exists x \in R_i[j]$ such that $x \cdot \mathbb{E}_{X_r}(X^{t+1}) = -1$.

In the previous section we showed that the number of responses sufficient to estimate $p_i$ with $(e^{1/n})$-approximation using subroutine Estimation() is $O(\log n)$. (In the sequel we let $\delta$ stand for $\frac{1}{n^2}$.) We next assess the number of rounds required for a processor $i \in P - F$ to become enlightened, that is the number of rounds required for $i$ either to collect sufficient responses for every processor $j \in P$ or to possess the result $-1$ from $j$, indicating that it crashed. The analysis follows along the lines of the analysis done in our earlier papers [8], [9]; except that here we argue about random selection of processors versus tasks in our prior work. Due to paucity of space we refer the kind reader to [8], [9] when appropriate to avoid a restatement of our results.

In $\text{Compute}$ step of QUERY stage a processor does at most $\lceil \log n \rceil$ tasks. Thus, it is possible that a live processor will not respond to a request to perform a test task. In this aspect the algorithm differs from the approach in [8], [9] where if a task is selected by a live processor, then it is consequently executed. Fact 1 below (a rewording after [3]) shows that $\whp$ no processor receives more than $\lceil \log n \rceil$ requests in one round.

**Fact 1:** If $n$ balls are uniformly randomly placed into $n$ bins with probability at least $1 - \frac{1}{n^2}$, for some $c > 0$, the fullest bin has $(1 + o(1))\frac{\log n}{\log \log n}$ balls.

We now analyze our algorithm in the three adversarial models. Let $F_i$ be the set of processors crashed before round $r$.

**Analysis for model $\mathcal{F}_{i}$**. Here $|F_i|$ is bounded as in model $\mathcal{F}$ of [9] with at most $hn \log n$ processor crashes for a constant $h \in (0, 1)$. Next two lemmas can be respectively proved along the lines of the proofs of Lemmas 5.1 and 5.2 in [9].

**Lemma 5:** In $O(\log n)$ rounds of algorithm $A_{est}$, $p_j$ is $\text{estimable}$ for every processor $j \in P$, $\whp$. 

Lemma 6: Once $p_j$ is estimable for every processor $j \in P$ then whp at least one processor in $P-F$ becomes enlightened in $O(\log n)$ rounds of algorithm $A_{est}$.

Analysis for model $F_{fp}$. Here we have $|F| \leq n-n^a$. For the purpose of analysis we divide an execution of the algorithm into two epochs: epoch $a$ consists of all rounds $r$ where $|F_r|$ is at most linear in $n$, so that the number of live processors is at least $c'n$ for some suitable constant $c'$; epoch $b$ consists of all rounds $r$, starting with the first round $r'$ (it can be round 1) when the number of live processors drops below some $c'n$ and becomes $c'n^a$ for some suitable constant $c''$. For the small number of failures in epoch $a$, we anchor the analysis to model $F_r$. For epoch $b$ we arrive at the following lemmas that can be respectively proved by arguing along the lines of Lemmas 4 and 7 in [8].

Lemma 7: In $O(n^{1-a} \log n)$ rounds of epoch $b$ $p_j$ is estimable for every processor $j \in P$, whp.

Lemma 8: Once $p_j$ is estimable for every processor $j \in P$, then at least one processor in $P-F$ is enlightened in $O(n^{1-a} \log n \log n)$ rounds of epoch $c$.

Analysis for model $F_{pl}$. Here we have $|F| \leq n-\text{poly log } n$, thus $|P-F| = \Omega(\text{poly log } n)$. For executions in $F_{pl}$, let $|P-F|$ be at least $b \log^2 n$, for specific constants $b$ and $c$ satisfying the model constraints. For the purpose of analysis we divide an execution of the algorithm into two epochs: epoch $b'$ consists of all rounds $r$ where $|F_r|$ remains bounded as in model $F_{fp}$ (for reference, this epoch combines epoch $a$ and epoch $b$ defined above); epoch $c$ consists of all rounds $r$, starting with the first round $r''$ (it can is round 1) when the number of live processors drops below $a_1 n^n$ and becomes $a_2 \log^{a_3} n$ for some suitable constants $a_1$, $a_2$, and $a_3$ (here $a_2 \geq b$ and $a_3 \geq c$). Note that either epoch may be empty.

Next we consider epoch $c$. If the algorithm terminates in round $r''$, the first round of the epoch, the costs remain the same as the costs analyzed for $F_{fp}$ above. If it does not yet terminate, it incurs additional costs associated with the processors in $P-F_{r''}$, where $|P-F_{r''}| \leq b \log^2 n$.

For epoch $c$ we arrive at the following lemmas that can be respectively proved by arguing along the lines of Lemmas 8 and 9 in [8].

Lemma 9: In $O(n)$ rounds of epoch $c$ $p_j$ is estimable for every $j \in P$, whp.

Lemma 10: Once $p_j$ is estimable for every $j \in P$, then at least one processor in $P-F$ becomes enlightened in $O(n)$ rounds of epoch $c$, whp.

Next we assess time complexity $T(n)$, work complexity $W(n)$, and message complexity $M(n)$. $\text{poly log } n$.

Theorem 2: Algorithm $A_{est}$ computes for every processor $i \in P-F$ an $(\epsilon, \delta)$-approximation of $p_i$, for the given $\delta > 0$ and $\epsilon > 0$, with the following complexities:

- Model $F_r$: $T(n) = O(\log n)$, work $W(n) = O(n \log n)$, and message complexity $M(n) = O(n^{1-\epsilon} n \log n)$.
- Model $F_{fp}$: $T(n) = O(n^{-1/2} \log n)$, work $W(n) = O(n \log n \log n)$, and message $M(n) = O(n \log^2 n \log n)$.
- Model $F_{pl}$: $T(n) = O(n)$, work $W(n) = O(n \text{poly log } n)$, and message $M(n) = O(n \log^2 n \text{ poly log } n)$.

Proof sketch. Combining the results of Lemma 4, and the fact that $O(n \log n)$ messages are sent in every round of $A_{est}$ with Lemmas 5 and 6 we get results for model $F_r$; with Lemmas 7 and 8 we get results for model $F_{fp}$ and with Lemmas 9 and 10 we get results for model $F_{pl}$.

VI. CONCLUSION

We presented a synchronous decentralized algorithm that assesses reliability of processors in the context of cooperative distributed computing. Specifically, we estimate the probabilities of processors performing their tasks correctly as an $(\epsilon, \delta)$-approximation. Our randomized algorithm is also able to deal with processor crashes. We established time, work, and message complexity analyses that demonstrate the efficiency of the algorithm with high probability guarantees. The analysis was performed in three different models that differ in the extent of crashes occurring during its execution. We note that when our algorithm is used as a precursor to network supercomputing, its costs are completely amortized if there is a polylog number of tasks per processor.

REFERENCES

[1] Distributed.net. http://www.distributed.net/
[2] Setatohome. http://setatohome.ssl.berkeley.edu/
[3] Y. Azar, A.Z. Broder, A.R. Karlin, and E. Upfal. Balanced allocations. SIAM Journal of Computing, 29(1):180–200, 1999.
[4] George Casella and Roger L. Berger. Statistical Inference. Duxbury Advanced Series, second edition, 2001.
[5] E. Christoforou, A. Fernandez, Ch. Georgiou, and M. Mosteiro. Algorithmic mechanisms for internet supercomputing under unreliable communication. In NCA, pages 275–280, 2011.
[6] P. Dagu, 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] P. Dagu and M. Luby. An optimal approximation algorithm for Bayesian inference. Artificial Intelligence Journal, 93(1-2):1–27, 1997.
[8] S. Davtyan, K. M. Konwar, A. Russell, and A. A. Shvartsman. Dealing with undependable workers in decentralized network supercomputing. In ICDCN 2013, pages 27–41, Mumbai, India, 2013.
[9] S. Davtyan, K. M. Konwar, and A. A. Shvartsman. Robust network supercomputing without centralized control. In Proc. of 15th Intl Conf. on Principles of Distributed Systems, OPODIS, pages 435–450, 2011.
[10] S. Davtyan, K. M. Konwar, and A. A. Shvartsman. Decentralized network supercomputing in the presence of malicious and crash-prone workers. In ACM PODC 2012, pages 231–232, Madeira, Portugal, 2012.
[11] M. Dyer, A.M. Frieze, and R. Kannan. An optimal approximation algorithm for Bayesian inference. Journal of the ACM, 38(1):1–17, 1991.
[12] A. Fernandez, C. Georgiou, L. Lopez, and A. Santos. Reliably executing tasks in the presence of malicious processors. Technical Report RoSaC-2005-9, Grupo de Sistemas y Comunicaciones, Univ. Rey Juan Carlos, 2005. http://gsyc.escet.urjc.es/publicaciones/tr/RoSaC-2005-9.pdf.
[13] A. Fernandez, C. Georgiou, L. Lopez, and A. Santos. Reliably executing tasks in the presence of untrusted entities. In SRDS, pages 39–50, 2006.
[14] A. Fernandez, C. Georgiou, L. Lopez, and A. Santos. Reliable internet-based master-worker computing in the presence of malicious workers. Parallel Processing Letters, 22(1), 2011.
[15] M. Jerum and A. Sinclair. Polynomial-time approximation algorithms for the ising model. SIAM Journal of Computing, 22:1087–1116, 1993.
[16] R. Karp, M. Luby, and N. Madras. Monte Carlo approximation algorithms for enumerative problems. J. of Algorithms, 10:429–448, 1989.
[17] K. M. Konwar, S. Rajasekaran, and A. A. Shvartsman. Robust network supercomputing with malicious processes. In Proc. of 17th Intl Symp. on Distributed Computing (DISC), pages 474–488, 2006.
[18] M. Paquette and A. Pelc. Optimal decision strategies in Byzantine environments. Parallel and Distrib. Computing, 66(3):419–427, 2006.