Brief Announcement:
Decentralized Network Supercomputing in the Presence of Malicious and Crash-Prone Workers*

Seda Davtyan
Computer Science & Engineering
University of Connecticut
sedat@engr.uconn.edu

Kishori M. Konwar
Immunology and Microbiology
University of British Columbia
kishori@interchange.ubc.ca

Alexander A. Shvartsman
Computer Science & Engineering
University of Connecticut
aas@cse.uconn.edu

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, our prior work introduced the decentralized approach, and provided a synchronous algorithm that is able to perform all tasks with high probability (whp), while dealing with malicious behaviors under a rather strong assumption that the average probability of live (non-crashed) processors returning bogus results remains inferior to 1/2 during the computation. There the adversary is severely limited in its ability to crash processors that normally return correct results. This work develops an efficient synchronous decentralized algorithm that is able to deal with a much stronger adversary. We consider a failure model with crashes, where given the initial set of processors \( P \), an adversary is able to crash any subset \( F \) of processors, where \(|F| \leq f \cdot n\), for a constant \( f \) (0 < \( f < 1 \)), under the constraint that there exists a subset \( H \subseteq P - F \), with \(|H| = \Omega(n)\), called the hardened set, such that the average probability of a processor in \( H \) returning a bogus result is inferior to 1/2. Here any processor may return bogus results, and \( H \) may be much smaller than \( P - F \), while the average probability of processors in \( P - F \) returning a bogus result may be greater than 1/2. We develop an efficient randomized algorithm for \( n \) processors and \( t \) tasks (\( n \leq t \)), where each live processor is able to determine locally when all tasks are performed, and obtain the results of all tasks. We prove that in \( \Theta(\frac{1}{\varepsilon^2} \log n) \) rounds all live workers know the results of all tasks whp, and that these results are correct whp. The work complexity of the algorithm is \( \Theta(t \log n) \), the message complexity is \( \Theta(n \log n) \), and the bit complexity is \( O(tn \log^3 n) \).

Categories and Subject Descriptors: F.2.0 [Theory of Computation]: ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY – General

General Terms: Algorithms, Reliability

Keywords: Distributed Algorithms, Fault-Tolerance, Randomized Algorithms, Communication, Internet Supercomputing

1. INTRODUCTION

Internet supercomputing comes at a cost substantially lower than acquiring a supercomputer, or building a cluster of powerful machines [1, 2, 3]. The promise of scalable network supercomputing depends on the availability of efficient decentralized algorithms—algorithms that do not depend on centralized control—able to deal with computers that may return bogus results and/or crash. A phenomenon of increasing concern is that workers may return incorrect results due to unintended failures, or may claim to have performed assigned work so as to obtain incentives, such as earning a higher rank in the system. A typical Internet supercomputer consists of a master server and a large number of computers called workers that perform computation on behalf of the master. Earlier approaches explored ways of improving the quality of the results obtained from untrusted workers in the master-worker setting, e.g., [5, 6, 7] present work-efficient algorithms where the master can determine the correct results whp.

Our prior work [4] removes the assumption of an infallible and bandwidth-unlimited master processor. However, in [4] the adversary is severely limited in its ability to crash processors that normally return correct results. In this work we aim to provide a decentralized solution that is able to deal with a much stronger adversary.

2. THE PROBLEM AND SYSTEM MODEL

We consider the problem of performing \( t \) tasks in a distributed system of \( n \) workers (\( n \leq t \)) without centralized control. The tasks are constant-time, independent, and admit at-least-once execution semantics. We assume that the workers can obtain the tasks from some repository. The fully-connected message-passing system is synchronous and the workers communicate using authenticated messages. The crash-prone workers can return incorrect results. The weakly adaptive adversary decides on the subset of processors to crash prior to the beginning of the computation. The computation is structured in rounds, where in each round a processor sends and receives messages and performs a local polynomial computation, where the local computation time is negligible compared to message latency.

Let \( P \) be the initial set of processors. For each \( i \in P \), we define \( p_i \) to be the probability of processor \( i \) returning incorrect results, independently of other processors. We define \( H \) to be a hardened set in an execution of a specific algorithm, if no processors in \( H \) crash, and \( \frac{1}{\varepsilon} \sum_{i \in H} p_i < \frac{1}{2} - \varepsilon \), for some constant \( \varepsilon > 0 \), i.e., the average probability...
of processors in $H$ returning incorrect results is inferior to $\frac{1}{2}$; we use $\varepsilon$ to prevent the average probability of misbehavior becoming arbitrarily close to $\frac{1}{2}$ as $n$ grows arbitrarily large.

We consider a linearly bounded adversary that can crash any subset $F$ of processors, such that $|F| \leq f \cdot n$, for a constant $f$, where $0 < f < 1$, provided that each execution has a hardened set of processors $H \subseteq P - F$, with $|H| > h \cdot n$, where $0 < h < 1 - f$. Note that, unlike in [4], the average probability of non-crashed processors returning bogus results can become greater than $1/2$.

3. ALGORITHMIC

We developed a randomized algorithm solving our cooperation problem for $n$ workers and $t = n$ tasks. The algorithm naturally generalizes for $t \geq n$ tasks, where each processor deals with fixed groups of $\lfloor t/n \rfloor$ tasks.

One of the main challenges in our algorithm is for every processor to find a subset of processors $S$ that maximizes the total average probability of computing results correctly, i.e., given the existence of the hardened set $H$, we want to select a subset $S$, where $H \subseteq S \subseteq P$, that maximizes $\sum_{s \in S} (1 - p_s)$, subject to $\frac{1}{n^2} \sum_{s \in S} p_s < \frac{1}{2} - \varepsilon$.

In solving the stated optimization problem, that we call SELECT; we assume that the probabilities $\{p_s\}$ are known. This assumption is made for simplicity only, as it is easily removed using our algorithms in [7], where $\{p_s\}$ are efficiently estimated with arbitrary accuracy. We developed a linear time algorithm for SELECT that uses a greedy approach. First, the processor identifiers are sorted in the decreasing order of $\{1 - p_i\}$. Then we iteratively select processors in the sorted order, keeping in mind the constraint on the average probability of returning result incorrectly for the subset $S \subseteq P$ of selected processors.

Now we detail the main algorithm that works in synchronous rounds. In every round a processor performs a random task and communicates its cumulative knowledge to one other randomly chosen processor. The number of rounds performed by the algorithm is an external (computation) parameter $K$. We prove that $K = \frac{1}{2} 10 L$ rounds are sufficient to obtain the high probability guarantee, for a certain $L$ that is shown to be $\Theta(\log n)$.

Each worker $i$ maintains two arrays of size linear in $n$. Array $R_i(1..n)$ is used to accumulate knowledge from different processors. Each element $R_i(j)$ is a set of results (initially empty) for task $j$, containing triples $(v_t, i, r)$ representing the result $v_t$ computed for task $j$ by processor $i$ in round $r$. The second array, $\text{Results}_i(1..n)$, stores the final results.

The algorithm iterates through three stages, Receive, Compute, and Send, where a single iteration comprises one round.

Receive Stage: Each processor receives messages sent during the previous round. The messages consist of the sender’s collection of the results. Upon receiving the messages a processor updates its own local copy of the sets by taking a union (this excludes duplicated triples).

Compute Stage: If the round count is less than $\frac{1}{5} L$, then processor $i$ randomly selects task $j$, computes the result $v_j$, and adds the triple $(v_j, i, r)$ to $R_i(j)$.

If the round count reached $K$, each processor $i$ goes over $R_i(j)$ for each task $j$ and extracts the set of processors that calculated the results. Among these processors, it then selects a subset of processors $S$ using algorithm SELECT. The result is then chosen to correspond to the plurality of the results calculated by processors in $S$ (in the analysis we prove that in fact a majority exists). The results are stored locally in array $\text{Results}_i(1..n)$. The processor halts.

Send Stage: Each processor $i$ chooses a target processor $k$ randomly from $P$ and sends to $k$ its results $R_i(1..n)$.

4. ALGORITHM ANALYSIS

For $t = n$ we prove that in $\Theta(\log n)$ rounds whp every task is performed $\Theta(\log n)$ times, possibly by different workers. Moreover, we prove that if a task has been performed $\Theta(\log n)$ times then whp in $\Theta(\log n)$ rounds of the algorithm each worker will acquire the results for every task.

We proved the following theorem that states our main result: The algorithm computes all $n$ tasks correctly at every processor in $\Theta(\log n)$ rounds whp.

We further show that for $t \geq n$ the time complexity of the algorithm is $\Theta(\frac{2}{3} \log n)$, the work complexity of the algorithm is $\Theta(t \log n)$, the message complexity of the algorithm is $\Theta(n \log n)$ and the bit complexity is $O(t n \log^2 n)$. The space complexity of the algorithm is $\Theta(t n \log^2 n)$.

5. SIMULATIONS

We developed a simulation of the algorithm for $t = n$. We let $L = \frac{10}{k} \log n$, for constant $k > 0$. We carried out simulations for up to $n = 1000$ processors. For each chosen $(n, k)$ pair we ran the simulation 100 times. We assumed that initially the average probability of returning incorrect result is inferior to $0.3$. We let $60\%$ of the processors return incorrect results with probability $0.1$, we denote this set of processors by $P_1$, and the remaining $40\%$ return incorrect results with probability $0.6$, this set is denoted by $P_2$; here $P = P_1 \cup P_2$. We let $\frac{2}{3}$ of the processors in $P_1$ crash, thus “ruining” the preset probabilistic balance between $P_1$ and $P_2$. Using procedure SELECT we then chose a subset of processors $S$, with $H \subseteq S$, such that the average probability of returning incorrect result stays inferior to $0.3$. The empirical data shows that even for the modest values of $k$ in $\{2, 3, 4\}$ after $\Theta(\log n)$ rounds the correct results of every task are known to all workers with very few exceptions.

6. REFERENCES