Reliably Executing Tasks in the Presence of Malicious Processors

Antonio Fernández[1], Chryssis Georgiou[2], Luis López[1], and Agustín Santos[1]

[1] LADyR, GSyC, Universidad Rey Juan Carlos, Móstoles, Spain
[2] Dept. of Computer Science, University of Cyprus, CY-1678 Nicosia, Cyprus

Problem and Motivation. The demand for processing large amounts of data has increased over the last decade. As traditional one-processor machines have limited computational power, distributed systems consisting of multitude of cooperating processing units are used instead. An example of such a massive distributed cooperative computation is the SETI@Home project [5]. As the search for extraterrestrial intelligence involves the analysis of gigabytes of raw data that a fixed-size collection of machines would not be able to effectively carry out, the data are distributed to millions of voluntary machines around the world. A machine acts as a server and sends data (aka tasks) to these client computers, which they process and report back the result of the task computation. This gives rise to a crucial problem: how can we prevent malicious clients from damaging the outcome of the overall computation?

In this work we abstract this problem in the form of a distributed system consisting of a master fail-free processor and a collection of processors (workers) that can execute tasks; worker processors might act maliciously. Since each task returns a value, we want the master to accept only correct values with high probability. Furthermore, we assume that the service provided by the workers is not free (as opposed to the SETI@Home project). For each task that a worker executes, the master computer is charged with a work-unit. Therefore, considering a single task assigned to several workers, our goal is to have the master computer to accept the correct value of the task with high probability, with the smallest possible amount of work. We explore two ways of bounding the number of faulty processors and evaluate an algorithm that the master can run. Our preliminary analytical results show that it is possible to obtain high probability of correct acceptance with reasonable amount of work.

Prior/Related work. The problem we consider in this work can be viewed as a special case of the voting problem [1] where a deciding agent decides on a value based on values generated and sent by processing nodes; the goal is for the agent to reach the correct decision with high probability. The voting problem has been considered in the presence of malicious voters (e.g., [6]), and optimal strategies have been identified that maximize the probability of correct decision. However, to the best of our knowledge, no prior work involving malicious voters consider minimizing the amount of work while restricting the probability of incorrect decisions of the voting procedure.

Do-All is the abstract problem of having a collection of processors to cooperatively perform a collection of independent tasks in the presence of failures [2]. This problem has been widely studied the last two decades in a variety of computation and failure models. Recently, this problem was studied under Byzantine processors [4]. Although the idea of executing tasks in the presence of malicious nodes is the same, the model and the problem we consider in this work are different.

Model and Algorithm. We assume there is a *master* processor M which has a task that has to be executed. This task returns a value, which M wants to reliably obtain. M is not capable of executing the task itself, so a set P of n (powerful) processors (called *workers*), $P = \{1, ..., n\}$, is made available to M. We assume that the workers might fail, and their faulty behavior is not restricted (e.g., send incorrect value, do not send any value, etc). We want to minimize the (expected) number of workers that have to run the task in order to obtain a failure probability of no more than ρ. The workers are continuously waiting for M to give them a task to execute, they execute a task if they are assigned one, and return the computed value. However, processors can be slow, and messages can get lost or arrive late. In order to introduce these assumptions in the model, we consider that there is a known probability d of M receiving the reply from a given worker on time. We also consider two types of known bounds on the number of faulty processors. We either assume that (i) there is a fixed bound $f < n/2$ on the maximum number of processors that can fail, or (ii) there is a probability $p < 1/2$ of any processor to be faulty.

We explore the following algorithm. First M chooses uniformly at random a subset S from the set P. We consider two ways to do so: either M defines a probability q and chooses each worker with that probability or it fixes a value s and chooses exactly s workers. Then, M sends the task to the processors in S and waits T time for the replies (T is a value set by M based on the value of d). M has a threshold τ and accepts as correct any value v received from at least τ workers within time T. If there is no such value, the algorithm has failed (it is repeated). Note that the (expected) work of an execution of this algorithm is exactly the (expected) size of the set S.

Using Chernoff bounds we show that our algorithm guarantees a failure probability of no more than ρ for: (a) $q = \frac{3(\ln 2 - \ln \rho)}{(1-2p)^2 npd}$ and $\tau = 2np(1-p)qd$ when we use parameters p and q, with expected work $E[|S|] = nq$, (b) $s = \lceil \frac{3(\ln 2 - \ln \rho)}{(1-2p)^2 pd} \rceil$ and $\tau = 2sp(1-p)d$ when we use parameters p and s, with work $|S| = s$, (c) $q = \frac{3(\ln 2 - \ln \rho)n^2}{(n-2f)^2 fd}$ and $\tau = \frac{2f(n-f)qd}{n}$ when we use parameters f and q, with expected work $E[|S|] = nq$, and (d) $s = \lceil \frac{3(\ln 2 - \ln \rho)n^3}{(n-2f)^2 fd} \rceil$ and $\tau = \frac{2f(n-f)sd}{n^2}$ when we use parameters f and s, with work $|S| = s$. (For $p < 1/6$ the values for $p = 1/6$ should be used. For $f < n/6$ the values for $f = n/6$ should be used.) More details can be found in [3].

References

1. D. Blough and G. Sullivan. A comparison for voting strategies for fault-tolerant distributed systems. In *SRDS'90*, pp.136–145, 1990.
2. C. Dwork, J. Halpern, and O. Waarts. Performing work efficiently in the presence of faults. *Siam J. on Computing*, 27(5):1457–1491, 1998.
3. A. Fernández, C. Georgiou, L. López, and A. Santos. Reliably executing tasks in the presence of malicious processors. Technical Report, 2005. (`http://gsyc.info/publicaciones/tr`.)

4. A. Fernández, C. Georgiou, A. Russell, and A. Shvartsman. The Do-All problem with Byzantine processor failures. *Theoretical Computer Science*, 333(3):433–454, 2005.
5. E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky. SETI@Home:Massively distributed computing for SETI. *Computing in Science and Engineering*, 3(1):78–83, 2001.
6. M. Paquette and A. Pelc. Optimal decision strategies in Byzantine environments. In *SIROCCO'04*, pp.245–254, 2004.