Probabilistic Byzantine Tolerance for Cloud Computing

Luciana Arantes*, Roy Friedman †, Olivier Marin *, and Pierre Sens *
*Sorbonne Universités, UPMC Univ Paris 06,
CNRS, Inria, LIP6
F-75005, Paris, France
Email: firstname.lastname@lip6.fr
†Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
Email: roy@cs.technion.ac.il

***Abstract*—Preventing Byzantine failures in the context of cloud computing is costly. Traditional BFT protocols induce a fixed degree of replication for computations and are therefore wasteful. This paper explores probabilistic Byzantine tolerance, in which computation tasks are replicated on dynamic replication sets whose size is determined based on ensuring probabilistic thresholds of correctness. The probabilistic assessment of a trustworthy output by selecting reputable nodes allows a significant reduction in the number of nodes involved in each computation task. The paper further studies several reputation management policies, including the one used by BOINC as well as a couple of novel ones, in terms of their impact of the possible damage inflicted on the system by various Byzantine behavior strategies, and reports some encouraging insights.**

***Keywords*-Byzantine, reputation, cloud.**

I. Introduction

A. Background

Heavy computational tasks are often performed these days in cloud computing environments by splitting the computation into multiple tasks using platforms such as Apache Hadoop [1] and Spark [2], or using volunteer computing platforms such as BOINC [3]. Commonly, these platforms are structured around a scheduler whose role is to disseminate the computation tasks into available compute nodes, who are responsible for computing the results of these tasks and returning their results. Obviously, this involves an implicit assumption that the scheduling and routing costs of tasks to compute nodes is considerably cheaper than calculating them, as otherwise it would not make sense to do so. An illustration of such a typical computing environment appears in Figure 1.

When trying to make such systems resilient to Byzantine behavior, one is faced with the large replication costs of masking Byzantine failures. Specifically, under the assumption that there could be up to f Byzantine nodes in the system, each computation task must be executed by $3f + 1$ nodes in naive application of traditional Byzantine fault tolerance approaches [4], [5]. More sophisticated mechanisms that distinguish between execution and validation require between $f+1$ [6] to $2f+1$ [7], [8]. As an example, if 10 nodes might be Byzantine, this means that each computation task needs to be executed on 11 or 21 nodes, as the case may be.

The principle of elasticity in cloud computing is to adapt resource provisioning as a means to optimize the tradeoff between cost and performance. Conversely, tolerating Byzantine failures induces incompressible costs. Traditional BFT protocols require making strong assumptions on the number of Byzantine nodes that exist in the system assuming for instance some known and fixed bound f on the number of Byzantine nodes.

In this paper, we extend the direction proposed in [9] and explore an alternative design path to the above: instead of fixing f arbitrarily, we replicate computations so that the probability of obtaining a correct result is satisfactory for the application. By trading the provable correctness of each computation step for a probabilistic one, we reduce the amount of resources required by the system. Specifically, we assume that each node j has a given probability p_j of acting in a Byzantine manner during an arbitrary calculation of a computation task. We define the *reputation* r_j of j as $1-p_j$. In addition, rather than requiring absolute masking of Byzantine failures, we only require obtaining a correct answer to each computation task with probability above a given threshold. Consequently, each computation task needs to be replicated only over the minimal number of compute nodes that will ensure meeting this probabilistic threshold.

As a motivating example, when a computation is sent to a group of compute nodes S, if only part of them generate an incorrect answer, the scheduler can send the same computation task to additional nodes until the probability of obtaining the correct result is above the given threshold. Otherwise, if all replies are the same, only if all the nodes have chosen to return a false answer it will go undetected by the scheduler.

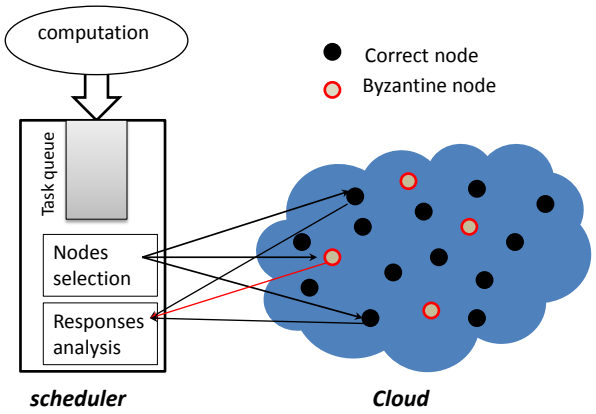


Fig. 1. Typical cloud computing environments

In case the probabilities of acting in a Byzantine manner are independent and identically distributed (IID) and equal to 0.1, then we can ensure that such an undetected Byzantine failure will occur with a probability of at most 0.0001 by replicating the computation on 4 nodes only. Hence, there is a great potential for reduction of resources compared to traditional approaches, propelling the exploration of this approach to Byzantine fault tolerance.

There are several scenarios where treating Byzantine behavior as a probabilistic event rather than a binary property of the nodes makes sense. One such scenario is when Byzantine behavior is a result of a heisenbug rather than an intrusion. Even when Byzantine behavior is caused by malice, if the scheduler picks nodes for replicated executions in a random manner, and assuming such nodes cannot control this choice, we can model the probability of one of these nodes being Byzantine as a probabilistic event. Finally, in order to avoid detection and in order to conserve resources, an intruder or attacker might prefer to return bad results only occasionally. Specific examples of the above are discussed later in this paper.

B. Contributions

In this paper, we explore the impact of probabilistic reliability, coupled with a reputation management mechanism, on mitigating the ability of Byzantine behavior to disrupt the computation in a cloud computing environment. In particular, we analyze the probability that a given reply is the correct answer for a given compute task when the task has been sent to a replication set of compute nodes with an assumed individual reputation level. Based on this model, we derive a corresponding scheduler's algorithm that greedily contacts minimal sets of compute nodes until obtaining enough confidence in a given value.

Next, we use our model to explore the ability of Byzantine behavior to damage the system under two axes. First, trying to cause the system to waste as much resources as possible. This is by deliberately returning false answers, which forces the scheduler to allocate additional nodes to the same compute task. The main defence of the scheduler in this case is its reputation management scheme, in which it can dynamically adjust the reputation of each compute node based on whether this node returned a value that was deemed to be the correct one or not. We explore three such reputation management strategies, namely the one used by BOINC [3] and two novel ones introduced by us. We show that all three schemes serve as an effective tool in limiting the influence of Byzantine behavior on the resources consumed by the system over time. It is worth pointing out that our original strategies do better than the one used in BOINC.

Last, the other direction of Byzantine behavior we explore is trying to hurt the correctness of the computation result by returning false answers. Yet, in order to avoid detection and therefore to increase the likelihood of success, in this attack the Byzantine nodes only return bad results when all compute nodes chosen for the replication set of the same compute task are Byzantine. Here, we show that when the scheduler picks compute nodes in a uniform random independent manner, Byzantine nodes must contribute to a very large number of useful computation for each successful attempt to return a false value. Further, we identify a large and an important family of problems for which even such occasional success in returning a false undetected value cannot prevent the system from obtaining an overall correct result. This means that for these problems, Byzantine nodes help the system much more than they hurt it, so their overall impact on the system is positive!

In summary, our contribution is a study of the impact of reputation management on the potential damage caused by Byzantine behavior in cloud computing environments when the goal is probabilistic reliability. We provide a formal model for this analysis, and use it to explore the ability of various reputation management systems in mitigating the damage caused by Byzantine behavior such as the extra resources the system is forced to consume and the ability to drive it into computing wrong results. We hope that our encouraging findings will serve as a motivation for further exploration of this direction.

C. Paper Roadmap

The rest of this paper is organized as follows: We survey related work in Section II. The model assumptions and goals are specified in Section III. The formal analysis is presented in Section IV including the resulting scheduler's algorithm. We explore various reputation management strategies, including the one used by BOINC and two novel ones in Section V as well as the cost to the Byzantine processes when trying to hide their behavior in Section VI. Finally, we conclude with a discussion in Section VII.

II. Related Work

Probabilistic consensus protocols ensure correct execution with probabilistic termination [10]–[13]. Alternatively, they can be stopped after a finite amount of execution rounds in which case their termination is guaranteed, but their safety becomes probabilistic. Several papers study the trade-off between termination probability and total step complexity of randomized consensus algorithms [14]–[16].

At any event, these protocols whose aim is to ensure consensus among the nodes despite Byzantine failures require at least $3f + 1$ nodes (and some even more than that) [17], [18]. Notice that the typical definition of Byzantine tolerant consensus is that if all correct nodes propose the same value, then this value has to be chosen. Hence, in scientific computing in which the result of each computation is deterministic based on the input, as in our model, these protocols can indeed be used to completely mask Byzantine occurrence, yet at a cost of high resource utilization.

Non-probabilistic BFT protocols always ensure safety, yet their termination depends on making synchrony timing assumptions [4], [19], [20]. They also require that the number of nodes participating would be at least $3f + 1$.

The idea of separating ordering from execution was explored in [7]. It was shown that given a trusted ordering entity, e.g., one that is created using a traditional BFT protocol, specific operations can be replicated only $2f + 1$ times.

In the SETI@HOME project [21], jobs are replicated by default on 2 machines. The scheduler then compares the results and if they do not match, it sends the job to additional machines until enough replies agree (2 by default). In the more advanced BOINC project [3], there is also an *adaptive replication* scheme whose goal is to reduce the computation overhead. When enabled, BOINC maintains an estimate $E(H)$ of host H's recent error rate according to the following scheme: $E(H)$ is initialized to 0.1. It is multiplied by 0.95 when H reports a correct (replicated) result. It is incremented by 0.1 when H reports an incorrect (replicated) result. Notice that it takes a long time to earn a good reputation and a short time to lose it.

The adaptive replication policy then works as described below. When the scheduler needs to assign a job to a host, the scheduler decides whether to trust the host according to the following rule: Given a trust threshold parameter A, currently hard-coded to 0.05, if $E(H) > A$, do not trust the host. Otherwise, trust the host with probability $1 - \sqrt{E(H)/A}$. If the scheduler trusts the host, it prefers to send it non replicated jobs. Otherwise, the scheduler sends replicated jobs to it. We applied, as in SETI@HOME and BOINC, these mechanisms and their associated magic numbers in our work. However, we seek to systematically and rigorously study the impact of a given choice of replication strategy on the confidence that the scheduler can obtain in the results.

A close related work to ours is [22] where the authors propose a reputation-based scheduler. Similarly to our approach, each node has a probability of being Byzantine and tasks are replicated over a set of nodes that ensure a probability of returning a correct response, which should be above a threshold. However, in order to determine the size and composition of such a group, the scheduler relies not only on the reputation of the nodes but also on the majority-based voting criterion of verification. In our approach, on the other hand, tasks are just replicated over the minimum number of nodes that meet the threshold. Moreover, in that work, the probability that a majority of nodes return the same wrong answer is neglected. In our case, the probability that all nodes return the same wrong answer must be below a threshold.

In [6], a trusted scheduler routes computation tasks to $f+1$ nodes. It is assumed that the computations take a long time to compute, and therefore the replicas take periodic checkpoints of their state. A separate set of auditors occasionally verify that all $f + 1$ checkpoints are consistent using a consensus protocol, and if not, they rollback the computation to a previously consistent checkpoint and restart it on different nodes. Such a strategy is also used to detect which nodes were Byzantine and eliminate them from the pool of compute nodes. As we take a probabilistic correctness approach, we can replicate the computation on fewer nodes (for moderate and large values of f).

A similar idea under the name of *spot-checking* was discussed in [23]. In that work, the results returned from compute nodes of a volunteer computing platform are sporadically verified by trusted spot-checkers. Whenever a mismatch is revealed, the nodes that returned false answers are *blacklisted* and never chosen again.

The closest work to ours is [9], who define the concept of *iterative redundancy*, which improves on the concept of *progressive redundancy* proposed in [24]. In progressive redundancy, when trying to obtain a threshold of t agreeing results, a compute task is sent to t compute nodes. If all replies are the same, then the single result is deemed correct. Otherwise, when there are multiple results, the compute task is sent to additional $t - l$ nodes until one result obtains the required threshold of support, where l is the number of occurrences of the most common value returned. In contrast to our approach, the goal of iterative redundancy is that one value will have at least t more supporters than any other value. It also starts by sending the compute task to t compute nodes and increasing the number of contacted nodes iteratively until the threshold is met. Yet, in iterative redundancy, the threshold t is determined based on the probabilistic reliability that such an answer will be correct given the assumed average reliability of nodes in the system.

There are several major differences between the work in [9] and ours. First, our model allows for various compute nodes to have different levels of reliability (reputation [25], [26]). Second, we explore several strategies of reputation

management and the corresponding malicious behavior that Byzantine nodes may employ against them. In particular, we investigate the performance of the reputation management strategy of the popular BOINC system in the face of Byzantine compute nodes as well as the performance penalty for Byzantine compute nodes who try to hide and only return (the same) wrong answer when all chosen nodes are Byzantine. Let us also note that the schedular algorithm in [9] is similar to ours, but specified in terms of their assumptions, analysis, and goals.

III. System and Threat Models

We consider a typical cloud IaaS architecture, i.e., one in which computing tasks continuously arrive and need to be scheduled on a large pool of available compute nodes (physical or VMs), similar to the one depicted in Figure 1. To that end, we assume a trusted node that act as a scheduler for these computing tasks. In particular, the scheduler is assumed to be fault-tolerant, always available, and always obey their prescribed protocol.

The communication in the system is performed by sending and receiving messages over a communication network. The network is assumed to be authenticated and reliable, with a bounded communication latency. That is, a message is only received if it was indeed sent by some node, and the receiver always knows who the true sender of a message is.

Unlike the scheduler, the compute nodes may occasionally act in a *Byzantine* manner. That is, while executing a computing task, each compute node j may return an incorrect answer (or not answer at all) with probability p_j. We refer to the probability $r_j = 1 - p_j$ that j returns a correct answer as the *reputation* of j. Notice that r_j's may change overtime.

Based on the above, whenever a scheduler node receives a compute task, it sends it to multiple compute nodes. When the replies arrive, the scheduler compares them. If they all agree, then the scheduler knows that this is the correct answer with a certainty that depends on the reputations of the nodes chosen. Otherwise, if some replies do not return within the deadline, the scheduler knows that these nodes are faulty and sends the same compute task to additional nodes. Similarly, if the replies do not match, then the scheduler knows that at least some of the nodes acted in a Byzantine manner and may send the compute task to additional nodes until it has enough probabilistic confidence in one of the replies.

We further assume that each compute task i has a normalized compute time T_i and that each compute node j has a known computing speed C_j. Hence, when there are no failures, a task i that is scheduled to be computed on a node j completes its execution on j within time T_i/C_j.

The number of nodes needed to execute each compute task in order to gain a certain confidence level in the reply is the main topic of this paper.

IV. Probabilistic Byzantine Tolerance Based on Reputation

In this section, unless specified otherwise, we assume that failure probabilities are independent.

A. Basic Formal Analysis

When the scheduler sends a compute task to a set S of compute nodes, the probability that all of them are correct is given by

$$P_C = \prod_{j \in S} r_i \tag{1}$$

Similarly, the probability that all are Byzantine is

$$P_B = \prod_{j \in S} (1 - r_j) \tag{2}$$

Further, the probability that a specific subset S_1 of S is Byzantine and all others are correct is

$$P_{SB} = \prod_{j \in S_1} (1 - r_j) \cdot \prod_{j \in (S \setminus S_1)} r_j \tag{3}$$

while the probability that all nodes in S_1 are correct and all others are Byzantine is

$$P_{SC} = \prod_{j \in S_1} r_j \cdot \prod_{j \in (S \setminus S_1)} (1 - r_j) \tag{4}$$

In particular, the probability of having at least one correct node is $1 - P_B$ and the probability of having at least one Byzantine node is $1 - P_C$. The probability of having exactly i correct answers is

$$\sum_{S_i \subset S} \prod_{j \in S_i} r_j \cdot \prod_{j \in (S \setminus S_i)} (1 - r_j),$$

where S_i denotes subsets of S of size i. In the particular case in which all compute nodes have identical reputation and uniform choosing probability, we get

$$\binom{|S|}{i} (r_j)^i \cdot (1 - r_j)^{n-i}.$$

where $n = |S|$. The goal of the scheduler is to send the task to enough nodes such that the chances of not detecting a false answer is below a given threshold, denoted T_B. The latter may occur only if all the chosen nodes are Byzantine since, in this case, they may all return the same false answer. Hence, S needs to be chosen such that P_B is bounded by the required threshold T_B. For example, if $p_j = 0.1$ ($r_j = 0.9$) for each compute node j and the threshold is 0.0001, then S should include at least 4 compute nodes. Notice that in this case, the a-priori chance of obtaining a correct answer from all nodes in S is $0.9^4 = 0.6561$. Further, with probability $4 \cdot 0.9^3 \cdot 0.1 = 0.2916$, there are exactly 3 correct answers, etc.

Similarly, the probability that the correct answer will be returned by at least i nodes is the summation of the probabilities of having k correct answers for all $i \leq k \leq n$. In the above example, the probability of having at least 3 correct replies is therefore $0.6561 + 0.2916 = 0.9477$, etc.

B. When All Replies are the Same

Once the results are returned by the chosen nodes, the scheduler can compute the following probabilities in order to decide whether to accept any of the values or to submit the compute task to additional compute nodes in order to increase its trust in the correctness of the reply. For example, suppose that all replies included the same value v. This means that either all compute nodes were correct or all compute nodes were Byzantine. Clearly, the situation in which all nodes are Byzantine is the worst, since the scheduler cannot detect that the result is incorrect.

We define by α the probability that all replies are incorrect and by β the probability that all replies are the same. Hence, we are interested in the probability $P(\alpha|\beta) = \frac{P(\beta|\alpha)P(\alpha)}{P(\beta)}$ (from Bayes theorem). Since Byzantine nodes can do whatever they want, we have no expression for $P(\beta|\alpha)$, but it can be upper bounded by 1. Also, $P(\alpha) = P_B$ while $P_C \leq P(\beta) \leq (P_C + P_B)$ (again, it is not equal to $P_C + P_B$ since the Byzantine nodes do not necessarily return the same answer even when all selected nodes are Byzantine). This gives a bound on $P(\alpha|\beta)$:

$$P(\alpha|\beta) < \frac{P_B}{P_C} \leq T_B \tag{5}$$

Using the numbers and assumptions of the example above, we get that the probability that all replies are incorrect is still close to 0.0001.

C. Multiple Answers

If the scheduler receives more than one answer, then obviously at most one answer is correct and any other answer was generated by Byzantine nodes. Suppose one of the replies is v_1 and denote by S_1 the set of compute nodes that returned v_1. The a-priori probability that S_1 includes correct nodes (and therefore v_1 is correct) and all other nodes are Byzantine is P_{SC} (formula 4). Denote by γ the event in which all nodes in S_1 are correct and all others acted in a Byzantine manner. Denote by δ the event in which either all nodes in S_1 are correct and all other nodes are Byzantine or all nodes in S_1 are Byzantine (and we do not know anything about the other nodes). The probability that v_1 is the correct value is the same as the probability that S_1 are correct and is expressed by $P(\gamma|\delta) = \frac{P(\delta|\gamma)P(\gamma)}{P(\delta)}$. Obviously, $P(\delta|\gamma) = 1$ and $P(\gamma) = P_{SC}$. Further,

$$P(\delta) = \prod_{j \in S_1} (1 - r_j) + P_{SC}$$

and therefore

$$P(\gamma|\delta) = \frac{P_{SC}}{\prod_{j \in S_1}(1 - r_j) + P_{SC}} \tag{6}$$

Hence, in this scenario, the scheduler needs to send the compute task to additional nodes until one value meets the correctness threshold.

Let us note that since the chance of a split vote, in which there are two answers each having a similar number of supports, is very low and becomes negligible as the size of the set of compute nodes performing the compute task is increased. For example, with 4 nodes as above, the chance of an equally split vote is only 0.0486 and rapidly diminishes with additional nodes.

If we consider b Byzantine nodes with the same reputation r_B and that every correct node has the same reputation r, the probability that $P(\gamma|\delta)$ is above a correctness threshold T_C is given by the following formula:

$$\frac{r^{|S_1|}(1 - r_B)^b}{(1 - r)^{|S_1|} + r^{|S_1|}(1 - r_B)^b} > T_C.$$

We can deduce that

$$|S_1| > \frac{log(T_C/(1 - T_C)) - b \cdot log(1 - r_B)}{log(r/(1 - r))} \tag{7}$$

Note that $T_C = 1 - T_B$. Finally, it is possible that some replies did not arrive at all. Since we assume failure independence in this section, the scheduler needs to send the compute task to additional nodes until enough nodes return an answer whose probability of being correct is above the given threshold.

D. The Schedular's Algorithm

In this section, we assume that the scheduler's main goal is to conserve resources in expectation while obtaining the minimal thresholds for correct values. This leads to a scheduling algorithm as listed in Algorithm 1, executed for each compute task. The algorithm takes as input a task CT and the threshold value T_B. Specifically, the scheduler selects a minimal set of nodes R_1 for which if all of them return the same reply, then the probability that it is the incorrect result is below the T_B threshold. To that end, it can use the formula (5) for $P(\alpha|\beta)$, by setting $S = R_1$ for P_C and P_B in formulae (1) and (2) respectively.

Let us denote by M_1 the set of nodes in R_1 that returned the value v that was most popular among the values returned by nodes in R_1. If all results are the same ($M_1 = R_1$), then v is chosen (lines 8 and 9). Otherwise, the schedular finds a second disjoint subset R_2 such that if all nodes in R_2 return the same value as the one returned by nodes in M_1, then the probability that this value is the correct one is above the required correctness threshold, $T_C = 1 - T_B$ (line 14). To that end, it uses the formula (6) for computing $P(\gamma|\delta)$, setting $S = R_1 \cup R_2$ and $S_1 = M_1 \cup R_2$.

If now all nodes in R_2 return the same result as the ones in M_1, then this value is chosen and the protocol terminates (lines 11 and 12). Otherwise, we define M_2 to be the set of nodes from $R_1 \cup R_2$ (line 4) that returned the most popular value and the scheduler searches for a set R_3 such that if all its members return the same result as the ones in M_2, then

Algorithm 1: Scheduler's Algorithm

input : Compute task *CT*; Threshold T_B

$R \leftarrow \emptyset$;
Choose a minimal set of nodes R_1 s.t. `cond1`(R_1,T_B);
for $i \leftarrow 1$ **to** ∞ **do**
  Set $R \leftarrow R \cup R_i$;
  Send *CT* to all members of R_i;
  Wait for replies from R_i;
  Let v be a most frequent value in all replies received and let M_i be the set of all nodes that returned it;
  if *($i == 1$) **and** ($M_i == R$)* **then**
    return v;
  end
  if `cond2`*(R, M_i,* T*)* **then**
    return v;
  end
  Choose a minimal set R_{i+1} s.t. $R_{i+1} \cap R == \emptyset$ and `cond2`($R \cup R_{i+1}$, $M_i \cup R_{i+1}$, T_B);
end

`cond1`*(set S, threshold th)*
  calculate P_B and P_C over S according to formulae (2) and (1);
  return ($P_B/P_C \leq th$);

`cond2`*(set S, set S_1, threshold th)*
  calculate $P(\gamma|\delta)$ over S and S_1 according to formula (6);
  return ($P(\gamma|\delta) > (1 - th)$);

this value will be correct with probability above the required threshold T_C, etc.

Notice that while the for loop in the scheduler's algorithm is not bounded, in practice, it terminates with high probability. For preventing it from terminating, different values should be returned often enough such that the correctness threshold T_C is never reached, which would mean that continuously nodes need to act in a Byzantine fashion. However, the probability of not terminating diminishes exponentially with the size of the set of contacted nodes (R in the algorithm).

Figure 2 shows the average number of steps of the scheduler's algorithm when the threshold T_B varies. The fraction of Byzantine nodes in the system is fixed to 15% and all the nodes have the same reputation r. If the latter is high ($r = 0.99$), then the algorithm quickly converges in just 1.345 steps, i.e, within a very small number of iterations.

V. On the Cost of Byzantine Behavior

A. Impact of reputation strategies

We first study the impact of reputation update policies on the correctness threshold T_C and the size of required same value set, S_1.

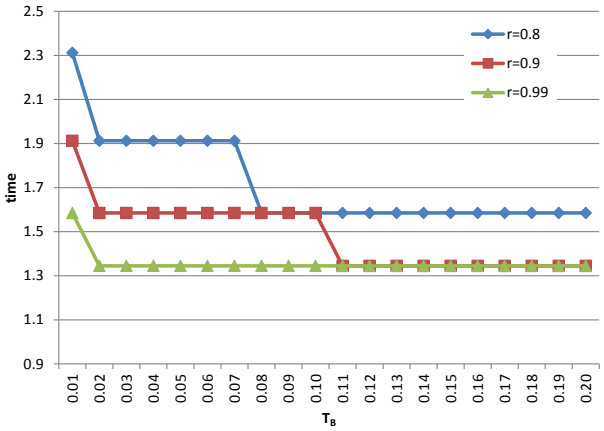


Fig. 2. Expected convergence time of scheduler's algorithm (in number of iterations required for convergence). The algorithm quickly converges when the reputation of nodes increases.

a) Reputation strategies: Considering formula (7), the chosen strategy to increase and decrease nodes' reputation has an impact on the size of S_1. Thus, for evaluation sake, we consider the following three strategies:

- The **BOINC** strategy presented in Section II where reputations are computed as 1 - error rate.
- The **symmetrical** strategy, where the reputation of a node is increased (respectively, decreased) by X% each time it returns a correct (respectively, wrong) value.
- The **asymmetrical** strategy, where Byzantine nodes are more punished than the others: the reputation of a node is increased by X% each time it returns a correct value and is decreased by 2X% when it returns a wrong value.

The threshold T_C is equal to 0.97 and in order to avoid that Byzantine nodes' reputation slows down too fast, we consider that these nodes return a bad value with probability $p = 0.5$. For the symmetrical and asymmetrical strategies, X was set to 20%.

Exploiting the three described update reputation strategies, Figure 3 gives the evaluation of the reputation of both the correct and Byzantine nodes at each time step. BOINC punishes compute nodes for Byzantine behavior severely and rewards their trust for good behaviour very slowly. On the other hand, the reputation of correct nodes in BOINC grows much slower than in the symmetric and asymmetric strategy. Ultimately, the goal of the reputation system is to obtain efficient resource utilization, which is explored in the next section.

b) Cost of one Byzantine behavior: We first consider only one Byzantine node ($b = 1$). Figure 4 shows the evolution of the size of S_1, i.e., as each strategy updates differently the reputation of correct and Byzantine nodes, we have evaluated how the size of S_1 decreases over the time.

We can observe in the figure that, as expected, strategies have different impact in S_1's size. BOINC, which strongly punishes the Byzantine node, is less efficient because the

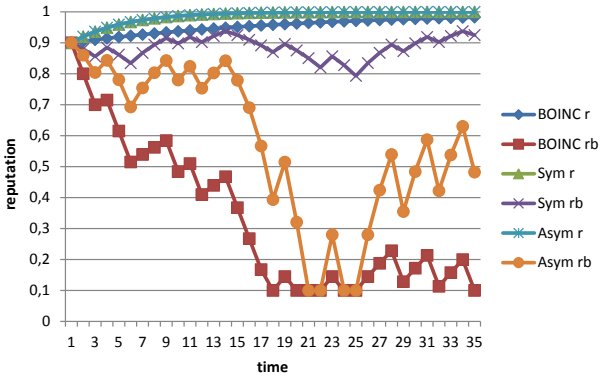


Fig. 3. Reputation strategies evolution. BOINC and asymmetric punish nodes severally, while asymmetric and symmetric better reward good behavior.

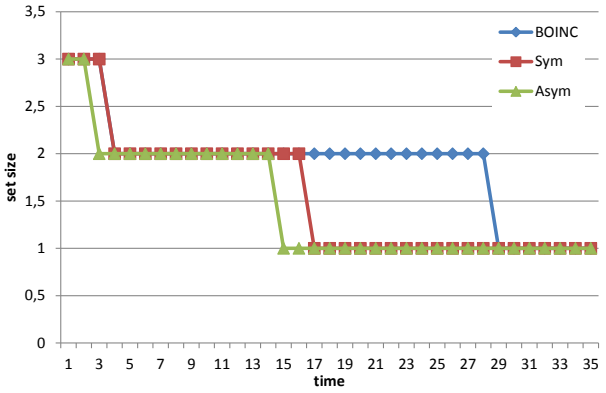


Fig. 4. Evolution of size of S1 for b =1 and T_C = 0.97 for the 3 reputation strategies. By keeping the highest difference between the reputation of correct and Byzantine nodes, the asymmetric strategy quicker converges to the smallest value of S1.

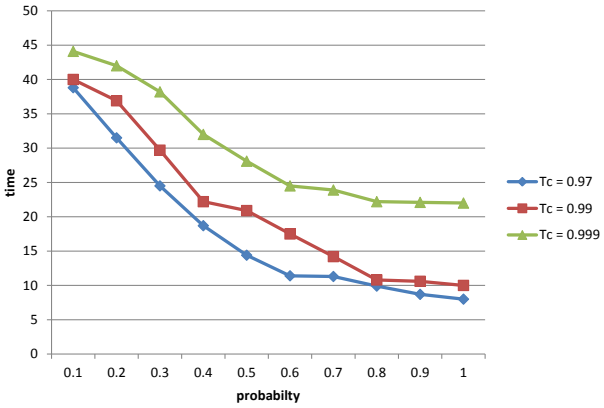


Fig. 5. Convergence time to reach the minimum size of S1 ($|S1| = 1$) when the probability of the Byzantine node to return a wrong value varies. A smart Byzantine node with sporadic wrong value computation highly increases the convergence time.

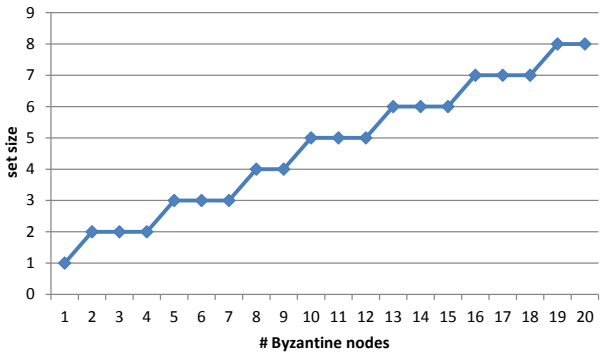


Fig. 6. Size of S_1 for T_C = 0.97 when b varies. When the system is stable, the adding of several Byzantine nodes increases the number of nodes needed to be contacted.

reputation of correct nodes increases slowly. However, in the asymmetric strategy, the size converges faster to small values than in the other strategies since the Byzantine node is more severely punished when it returns a wrong value, compared to the other two strategies. In summary, the asymmetric strategy converges to the smallest set size faster due to its combined harsh punishment for Byzantine behavior with favorable rewards for good behavior. It is, therefore, the best strategy.

We denote a "smart" Byzantine node, the one which, aiming at keeping a better reputation, does not always return a wrong value. Hence, in order to evaluate the impact of a "smart" Byzantine node on the asymmetric strategy, the best one, we have also measured the time for S_1 to be of size 1, which characterizes the convergence time, i.e., the period during which the Byzantine node can slow down the system. After that, the reputation value of the Byzantine node will be too small to have any impact on the system. To this end, we have varied the probability p of the Byzantine node of returning a wrong value and consider 3 correctness threshold values. The results are given in Figure 5.

With a reliable threshold of $T_C = 0.97$, p has a high impact on the convergence time. For instance, a "smart" Byzantine node which only returns one wrong value within 10 times ($p = 0.1$), increases 5 times the convergence time compared to a Byzantine node which always returns a wrong value ($p = 1$). On the other hand, for greater threshold value, p has a lower impact. For instance, if we set $T_C = 0.999$, the same "smart" Byzantine node only doubles the convergence time whereas with $T_C = 0.9999$ (not illustrated in the figure) the smart node only increases by 10% the convergence time.

c) Cost of several Byzantine behaviors: Figure 6 shows the size of S_1 when the number of Byzantine nodes varies (x-axis) for a threshold $T_C = 0.97$ in the asynchronous strategy. Considering a stable system ($|S_1| = 1$) where $r = 0.99856$ and $r_b = 0.9$, the number of Byzantine nodes is increased. Each Byzantine node arrives with a reputation $r_b = 0.9$ and we observe an increase by one of S_1 every 3 Byzantine nodes. We can thus conclude that the initial reputation value of nodes has a direct impact on the increase of S_1 size. For instance, if we consider an initial reputation value equal to 0.7, the size of S_1 is increased every 5 Byzantine nodes insertions.

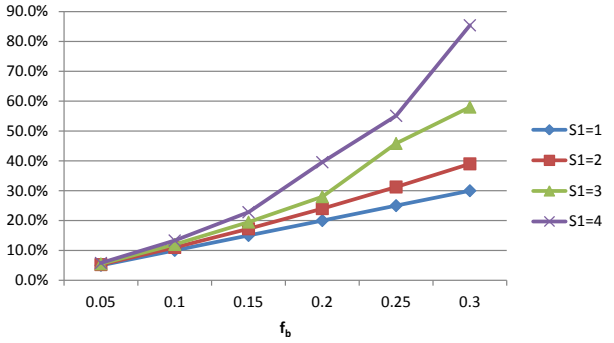


Fig. 7. Overhead according to the fraction of Byzantine nodes. When the proportion of Byzantine nodes is high, the probability of having wrong responses in larger sets increases, inducing a higher overhead.

B. Additional cost of Byzantine nodes

We now express the cost to obtain a given number of correct responses. We denote $N_{act}(s)$ the number of nodes necessary to be contacted in order to obtain s responses from correct nodes. $N_{act}(s)$ is at least equal to s plus possibly additional nodes, if some of the first s responses concern Byzantine nodes. Considering f_b as the fraction of Byzantine nodes of the system, the number of responses returned by Byzantine nodes, N_{Byz}, in a set of size s, is given by:

$$N_{Byz}(s) = \sum_{i=1}^{s} i.\binom{s}{i}.f_b^i$$

Then, N_{act} for a set of size s is given by the following formula:

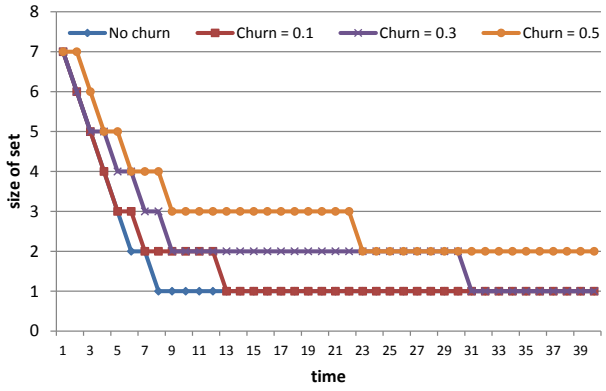
$$N_{act}(s) = s + N_{act}(\lfloor N_{Byz}(s) \rfloor) \quad (8)$$

Figure 7 shows the percentage of overhead, i.e., the cost of contacting additional nodes (=$(N_{act}(s) - s)/s$), when f_b varies for different sizes of response set s in the asynchronous strategy. An overhead of 50% means that, in average, $1.5s$ nodes must be contacted in order to obtain s values returned by correct nodes.

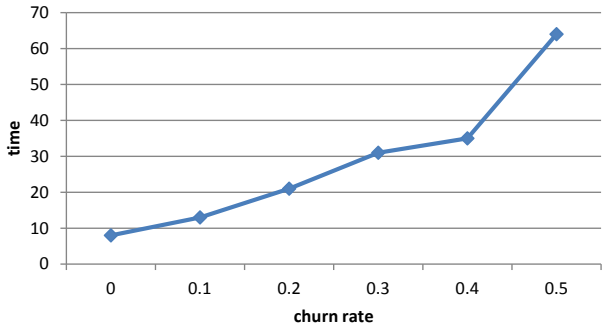
C. The impact of churn

Churn may have a strong impact in the evaluation of the size of S_1 since new correct (resp., Byzantine) nodes which arrive in the system can have an initial reputation value underestimated (resp., overestimated), when compared to the reputation of those nodes which are already in the system. We have thus evaluated the impact of nodes churn on the size of S_1 by considering the asymmetrical strategy. We set the threshold T_C to 0.97 and the number of Byzantine nodes b to 5. Churn was simulated by increasing the proportion of new nodes that arrive in the system at each time. Their initial reputation was fixed to 0.9. The results are given in figures 8.

We can observe that even a low churn of 10% (churn = 0.1 in the figures) has a high impact in the convergence time:



(a) Evolution of size of S_1 with churn.



(b) Convergence time with churn.

Fig. 8. Impact of churn with $T_C = 0.97$. As expected, high churn delays convergence since the system has less chance to learn and adjust the reputation of nodes.

the time to reach a set of size 1 is 1.625 times longer than with no churn. For high levels of churn (churn $\geq$ 0.3), the convergence time increases drastically. Figure 8(b) shows the convergence time in relation to the churn rate.

VI. On the Cost of Hiding Byzantine Behavior

Byzantine nodes whose main goal is to hurt the correctness of the computation and avoid the discovery of their behavior as much as possible, may act as follows: Whenever all compute nodes chosen for a given compute task are Byzantine, then they all collaborate to return the same false value. Otherwise, they all return the correct result for the compute task. Note that here we move to a model in which Byzantine behavior is deterministic and dependant on the choice of compute node. Since in this model an incorrect answer is never detected, the reputation of all nodes is always the same.

We analyze the case in which the number of Byzantine nodes is a fraction of all other nodes, and the scheduler picks compute nodes uniformly at random and independently for each compute task. In this case, when the fraction of Byzantine nodes is f_b, where $0 \leq f_b \leq 1$, whenever the scheduler chooses a set S of compute nodes for a given task, the probability that all of them will be Byzantine is $f_b^{|S|}$. Hence, in a long running system, in expectation, once every $\frac{1}{f_b^{|S|}}$ all nodes will be Byzantine, returning an undetectable

false answer. In all other times, a correct answer will be returned.

Given the way the schedular chooses nodes, each Byzantine node participates in $\frac{1}{f_b^{|S|}}$ computations for each time in which all other nodes are also Byzantine. At this time, all of them can return a bogus answer without computing the real task, and thus they do not waste any computational resources. However, in the other times, Byzantine nodes need to compute the task in order to know the correct answer returned by correct nodes. Assume that all computation tasks consume similar resources denoted by C. The amount of computation resources that each Byzantine node consumes in order to avert a single computation result is

$$(\frac{1}{f_b^{|S|}} - 1) \cdot C.$$

Another way of looking at this problem is that, with such a hiding strategy, Byzantine nodes actually contribute with a significant amount of useful work for valid computations. That is, for each compute task that they sabotage, they do useful work for $(\frac{1}{f_b^{|S|}} - 1)$ computations. For example, in the case that f_b is 0.2 and $|S| = 4$, they contribute to 624 valid computations for each disrupted one, which suggests that this strategy is highly inefficient for the Byzantine nodes.

In fact, for a large family of important (iterative) problems, we show below that using the hiding strategy, the worst damage Byzantine nodes can do is to slightly delay the computation. However, while doing so, they in fact spent more resources with useful work for the computation than the amount of extra resources necessary to correct their wrongdoing.

Specifically, the idea is that in many iterative computing problems, there is a bound on the difference between the partial result obtained in a given iteration and the partial result that is expected in the following iteration. Further, if the convergence is guaranteed regardless of the exact value with which each iteration started, that is, the iterative method is *globally convergent*, then the worst damage that such coordinated Byzantine behavior may causes is a setback of at most a single (or very few) iterations. In particular, most iterative methods for systems of linear equations are globally converging. In case the method is only *locally convergent*, such as *Newton's method*, small deviations made by Byzantine nodes will typically not prevent convergence whenever the initial guess enabled convergence.

On the other hand, large deviations can usually be detected and ruled out as they noticeably diverge from the values expected by the rate of convergence of the iterative method, which again prevents hiding Byzantine nodes from employing them. In the above example, this means that, for each iteration it disrupts, a Byzantine node contributes to 624 valid iterations and, therefore, its damage is at most the equivalent of executing a small number of valid iterations. In other words, in the above examples, when nodes are selected uniformly at random by the scheduler for each iteration independently, hiding Byzantine nodes contribute to much more useful work than the damage they do and, even if they can slightly prolong the computation process, they cannot halt it.

VII. Conclusion

The main contribution of this paper is a study of probabilistic reliability in Byzantine cloud computing environments. In particular, we have found promising evidence that reputation-based replication substantially helps in mitigating Byzantine behaviors and its impact on the correctness of the computation in such environments.

Assuming an environment where most compute nodes are trustworthy, the approach is simple. It replicates task executions on a varying number of nodes to ensure that a consistent answer from all nodes has a significant probability of being correct. When no returned value obtains enough support to be deemed correct, additional nodes are contacted. Yet, in order to conserve resources, if all these additional nodes return the most popular returned value up to that point then the probability that this value is the correct one is above the required correctness threshold.

Iterations of this process converge fast towards a trustworthy answer, with the additional advantage that they detect incorrect nodes with a significant degree of accuracy and actively discourage malicious behaviors. Nodes with consistently incorrect responses will quickly get discarded. Nodes that respond correctly most of the time in order to acquire a good reputation before injecting wrong data incur a very high effort/reward ratio, and end up actually taking a positive part in the system's computation. Further, we have also identified important sets of problems in which such nodes cannot disrupt the system from eventually reaching a correct answer. The worst damage they can do is a slight slowdown and, in fact, they end up helping the system more than disrupting it.

We have also investigated the effectiveness of multiple reputation management strategies, including the one employed by BOINC as well as a couple of novel ones. We found that all three are effective, but our new method, nicknamed asymmetric was the best in terms of its impact on the consumed resources and convergence times.

Given the encouraging results of this work, we plan to extend this direction by taking the computational power of nodes into account when forming the replication sets. The goal will be to identify beneficial trade-offs between the total computation time and the trustworthiness of the result. More precisely, the scheduler algorithm consists of multiple iterations that end when some value obtains the reliability threshold. Choosing very fast nodes can reduce the time to compute each iteration. However, if these nodes are less reliable, then additional iterations may be needed. Hence, when the goal is the expected compute time until a reliable

answer is obtained, one should look for an optimal tradeoff point between the positive impact of high compute power and the negative influence of lower reliability. We should point out that in cloud computing environment, usually, the higher the compute power of a node, the higher the cost of reserving it. Thus, there exists also a tradeoff between cost and reliability to be exploited.

Acknowledgements: This work was supported by a grant from CAMPUS France and the Israeli Ministry of Science and Technology PHC-Maimonide, grant #31807PC.

REFERENCES

[1] Apache hadoop project - http://hadoop.apache.org.

[2] Apache spark project - http://spark.apache.org.

[3] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[4] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[5] L. Lamport. Byzantine paxos by refinement. In *Proceeding of the 25th International Conference on Distributed Computing (DISC)*, pages 211–224, 2011.

[6] A. Agbaria and R. Friedman. A Replication- and Checkpoint-Based Approach for Anomaly-Based Intrusion Detection and Recovery. In *33rd IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 137–143, June 2005.

[7] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating Agreement from Execution for Byzantine Fault Tolerant Services. In *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 253–267, 2003.

[8] Christian Cachin, Dan Dobre, and Marko Vukolic. Separating data and control: Asynchronous BFT storage with 2t + 1 data replicas. In *Stabilization, Safety, and Security of Distributed Systems - 16th International Symposium, SSS 2014, Paderborn, Germany, September 28 - October 1, 2014. Proceedings*, pages 1–17, 2014.

[9] Y. Brun, G. Edwards, J. Y. Bang, and N. Medvidovic. Smart Redundancy for Distributed Computation. In *Proc. of the 31st IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 665–676, 2011.

[10] Michael O. Rabin. Randomized byzantine generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, SFCS '83, pages 403–409, Washington, DC, USA, 1983. IEEE Computer Society.

[11] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC, pages 154–162, 1984.

[12] Sam Toueg. Randomized byzantine agreements. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC, pages 163–178, 1984.

[13] Bruce M. Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *ACM Trans. Algorithms*, 6(4):68:1–68:28, September 2010.

[14] Hagit Attiya and Keren Censor-Hillel. Lower bounds for randomized consensus under a weak adversary. *SIAM J. Comput.*, 39(8):3885–3904, 2010.

[15] Hagit Attiya and Keren Censor. Tight bounds for asynchronous randomized consensus. *J. ACM*, 55(5):20:1–20:26, November 2008.

[16] James Aspnes. Lower bounds for distributed coin-flipping and randomized consensus. *Journal of the ACM*, 45:559–568, 1997.

[17] R. Friedman, A. Mostefaoui, and M. Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46–56, Jan 2005.

[18] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous byzantine consensus with $t < n/3$ and $o(n^2)$ messages. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 2–9, New York, NY, USA, 2014.

[19] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[20] Rui Garcia, Rodrigo Rodrigues, and Nuno Preguiça. Efficient middleware for byzantine fault tolerant database replication. In *Proceedings of the 6th ACM Conference on Computer Systems*, EuroSys, pages 107–122, 2011.

[21] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: An experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, November 2002.

[22] Jason D. Sonnek, Mukesh Nathan, Abhishek Chandra, and Jon B. Weissman. Reputation-based scheduling on unreliable distributed infrastructures. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), 4-7 July 2006, Lisboa, Portugal*, page 30, 2006.

[23] L.F.G. Sarmenta. Sabotage-Tolerance Mechanisms for Volunteer Computing Systems. *Future Generation Computing Systems*, 14(4):561–572, 2002.

[24] A. Bondavalli, S. Chiaradonna, F.D. Giandomenico, and J. Xu. An Adaptive Approach to Achieving Hardware and Software Fault Tolerance in a Distributed Computing Environment. *Journal of Systems Architecture*, 47(9):763–781.

[25] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Comput. Surv.*, 42(1):1:1–1:31, December 2009.

[26] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, December 2000.