Sabotage-Tolerance Mechanisms for Volunteer Computing Systems

Luis F. G. Sarmenta

Ateneo de Manila University
Loyola Heights, Quezon City
Metro Manila, Philippines
Email: lfgs@admu.edu.ph
URL: http://www.cag.lcs.mit.edu/bayanihan/

Abstract

In this paper, we address the new problem of protecting volunteer computing systems from malicious volunteers who submit erroneous results, by presenting *sabotage-tolerance* mechanisms that work without depending on checksums or cryptographic techniques. We first analyze the traditional technique of *voting*, and show how it reduces error rates exponentially with redundancy, but requires all work to be done several times, and does not work well when there are many saboteurs. We then present a new technique called *spot-checking* which reduces the error rate linearly (i.e., inversely) with the amount of work to be done, while only costing an extra fraction of the original time. Integrating these mechanisms, we then present the new idea of *credibility-based fault-tolerance*, wherein we estimate the conditional probability of results and workers being correct, based on the results of using voting, spot-checking, and other techniques, and then use these probability estimates to direct the use of further redundancy. Using this technique, we are able to attain mathematically guaranteeable levels of correctness, and do so with much smaller slowdown than possible with voting or spot-checking alone. Finally, we validate these new ideas with Monte Carlo simulations, and discuss other possible variations of these techniques.

Key words: peer-to-peer computing, distributed computing, metacomputing, internet computing, Java, network of workstations, collaborative computing

⋆ Draft submitted to *Future Generation Computer Systems*. The work described in this paper was done as part of the author's Ph.D. thesis, *Volunteer Computing* [8], at the Massachusetts Institute of Technology, Cambridge, MA, USA.

1 Introduction

In recent years, there has been a rapidly-growing interest in *volunteer computing systems*, which allow people from anywhere on the Internet to contribute their idle computer time towards solving large parallel problems. Probably the most popular examples of these are distributed.net, which gained fame in 1997 by solving the RSA RC5-56 challenge using thousands of volunteers' personal computers around the world [2], and SETI@home, which is currently employing hundreds of thousands of volunteer machines to search massive amounts of radio telescope data for signs of extraterrestrial intelligence [9]. A number of academic projects have also ventured to study and develop volunteer computing systems, including some, like our own Bayanihan [8], that promote web-based systems using Java [1,4]. Even the commercial sector has joined the fray, with a number of new startup companies seeking to put volunteer computing systems to commercial use, and pay volunteers for their computer time [3,5–7].

The key advantage of volunteer computing over other forms of metacomputing is its ease-of-use and accessibility to the general public. By making it easy for anyone – even casual users – on the Internet to join in a parallel computation, volunteer computing makes it possible to build very large global computing networks very quickly, as proven by the success of SETI@home and distributed.net. This same advantage, however, also creates a new problem: if we allow anyone to join a computation, how do we prevent *malicious volunteers* from *sabotaging* the computation by submitting bad results?

Traditional *fault-tolerance* techniques that work well against *random* faults, such as using parity and checksum schemes, will not be effective in this case because they cannot protect against *intentional* attacks by malicious volunteers – or *saboteurs* – who can disassemble the code, and figure out how to produce valid checksums for bad data. Thus, there is a need for new *sabotage-tolerance* mechanisms that work in the presence of malicious saboteurs without depending on checksums or cryptographic techniques.

In this paper, we present such techniques. We begin with the traditional technique of *voting*, and present the new mechanisms of *spot-checking*, *backtracking* and *blacklisting*. We then integrate these mechanisms by presenting the new idea of *credibility-based fault-tolerance*. In this technique, we estimate the *credibility* of results and workers as the probability of their being correct given the results of using voting, spot-checking, and other techniques. By then using these estimates to determine whether a piece of work needs to be repeated or is credible enough to be accepted, we are not only able to attain mathematically guaranteeable levels of correctness, but are also able to do so with much smaller slowdown than possible with traditional voting techniques. Finally, we validate these new ideas with Monte Carlo simulations, and discuss other possible variations and extensions of these techniques.

2 Models and Assumptions

Basic Model. In this paper, we assume a *work-pool-based master-worker model* of computation, which is used in practically all volunteer computing systems today, as well as in many grid systems, metacomputing systems, and other wide-area network-based parallel computing systems in general.[1] In this model, a *computation* is divided into a sequence of *batches*, each of which consists of N independent *work objects*. At the start of each batch, these work objects are placed in a *work pool* by the *master* node, and are then distributed to P different *worker* nodes who execute them in parallel and return their *results* to the master. When the master has collected the results for all the work objects, it generates the next batch of work objects and repeats the whole process until the computation is done.

To model sabotage, we assume that up to a certain *faulty fraction* f of the P workers are *saboteurs* who submit bad results for the work they are given. Without the use of sabotage-tolerance mechanisms, these bad results eventually get accepted at the end of each batch, and become *errors*. The average fraction of final accepted results that are errors is defined as the *error rate* (ε). This fraction is also equal to the probability of each *individual* final result being bad.

The goal of our sabotage-tolerance mechanisms is to reduce the error rate to an acceptably small value while minimizing the degradation of the system's performance.

Acceptable Error Rate. In reducing the error rate, we assume that there is a non-zero *acceptable error rate*, $\varepsilon_{\mathrm{acc}}$, below which occasional errors are deemed acceptable.

For some applications, the acceptable error rate can be relatively high (1% or more). These include applications such as image or video rendering, where a few scattered erroneous pixels would not be noticeable to the human eye. These also include "self-correcting" applications such as genetic algorithms, and some statistics applications. Search applications with rare and easily verifiable true-positive results, such as the SETI@home application and the code cracking

[1] Furthermore, although most message-passing and shared-memory programs do not fall under this model, we have shown that these can also be implemented on work-pool-based systems by using a BSP-style programming model [8].

applications of distributed.net, can also tolerate a relatively high error rate. In these applications, all errors would be in the form of false negatives. These can cause us to take a longer time to find the solution, but if we have more good workers than bad, and we keep reassigning work until we find the desired solution, we would eventually find the solution with only a little slowdown – i.e., if we have a false negative rate of ε, then the expected number of times we would have to do all the work before finding the true positive solution is only $1/(1-\varepsilon)$. [2]

Of course, there are also other applications – including many traditional scientific computational applications – that cannot tolerate even a single error in a batch (or even in several batches). Even in these cases, however, we can still use our sabotage-tolerance mechanims by simply making the acceptable error rate, ε_{acc}, small enough to make the probability of getting *any* error at all acceptably small. For example, suppose that a computation has 10 batches of 100 work objects each, and that a single error in *any* of the 10 batches will cause the whole computation to fail. In this case, to reduce the probability of the *whole* computation failing, $P(\text{fail})$, to less than 1%, the acceptable error rate for each batch, ε_{acc}, which is equal to the probability of an *individual* result failing, must be at most $\varepsilon_{\text{acc}} = P(\text{fail})/(10 \times 100) = 0.001\% = 1 \times 10^{-5}$. Fortunately, although this error rate may seem small, we show in Sect. 4 how such low error rates can be achieved with only a small slowdown.

Performance. To measure the performance and efficiency of fault-tolerance mechanisms, we consider *redundancy* and *slowdown*. Redundancy is defined as the ratio of the average total number of work objects actually assigned to the workers in a batch vs. the original number of work entries. Slowdown is defined as a similar ratio between the *running times* of the computation with and without the use of the mechanism. In general, redundancy leads to an equivalent slowdown, but in some cases – especially when workers can join, leave, or get blacklisted in the middle of a batch – slowdown may be different from redundancy. If workers leave, for example, then the remaining workers must take over their work. This increases the slowdown, even though the total amount of work, and thus the redundancy is the same.

Saboteurs. For simplicity, we assume that each saboteur is a Bernoulli process with a constant probability s of submitting a bad result, known as the *sabotage rate*. In reality, saboteurs may communicate and collude with other saboteurs to try to increase their chances of getting their bad answers accepted. They may also change their sabotage rates in time in order to try to fool techniques such as credibility-based fault-tolerance. For now, we do not address these possibilities, and simply assume that we can employ other techniques such as checksums and randomization to make such attacks difficult. As discussed in

[2] This is computed as: $\sum_{i=1}^{i=\infty} P(\text{solution found in } i \text{ rounds}) \cdot i = \sum_{i=1}^{i=\infty} \varepsilon^{i-1} \cdot (1-\varepsilon) \cdot i$.

Sect. 6, however, if we can model these situations mathematically and derive the appropriate conditional probabilities, we can adapt the general idea of credibility-based fault-tolerance accordingly to cover these situations as well.

We also make the simplifying assumption that all workers, whether saboteurs or not, run at roughly the same speed so that they get roughly equal shares of work. Note, however, that under this assumption, we can still handle saboteurs who are faster than others by simply treating them as multiple saboteurs.

3 Basic Mechanisms

Aside from the mechanisms presented here, there are also other ways to protect a computation against sabotage. These include techniques that take advantage of authentication, encrypted computation, and obfuscation to make it difficult for saboteurs to understand the computational code and thus to produce valid-looking, but erroneous, results [8]. Such techniques, however, may be application-specific and may rely on heuristics that are difficult to analyze mathematically.

In this paper, therefore, we focus on mechanisms that rely on the more general principles of redundancy and randomization. Not only are these mechanisms more robust (i.e., they assume less about the application, and thus work for a broader range of applications and assumptions), but they can also be combined with other approaches to achieve even better reliability.

3.1 Majority Voting

The most basic such mechanism is the traditional mechanism of *majority voting*, wherein we do each piece of work several times, and decide which result to accept through a vote. As shown in Fig. 1, we can easily implement this scheme by using a modified *eager scheduling work pool*. Here, as in normal eager scheduling [1,8], the master continuously goes through the *work entries* in the work pool in round-robin fashion, until the *done* flags of all work entries are set. In this case, however, the *done* flag of each *work entry* is left unset until we collect m matching results for that work entry, thus implementing an *m-first* voting scheme.

Assuming that bad answers match (so that saboteurs can vote together), this voting scheme can be shown to have an expected redundancy of $m/(1-f)$, and an error rate that shrinks roughly exponentially in m, as shown in Fig-

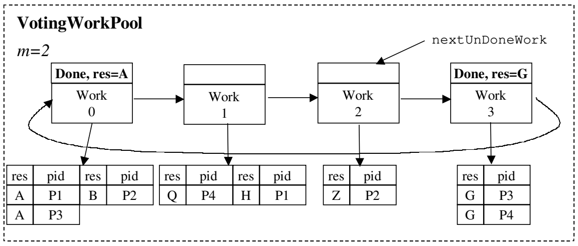

Fig. 1. Eager scheduling work pool with m-first majority voting.

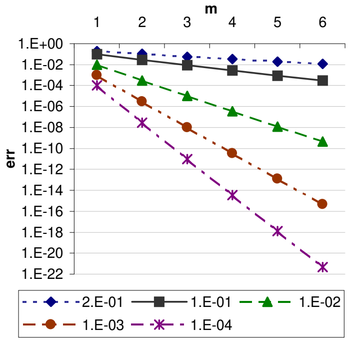

Fig. 2. Error rate of majority voting for various values of m and f.

ure 2.[3] This exponentially shrinking error rate means that voting works very well in systems with a small faulty fraction, f, and furthermore, that it gets increasingly better as f decreases. Thus, in systems with very low error rates to begin with, such as hardware systems, it does not take much redundancy to shrink the error to extremely low levels.

Unfortunately, however, voting also has its drawbacks. First, it is inefficient when f is not so small. As shown in Fig. 2, for example, at $f = 20\%$, doing all the work at least $m = 6$ times still leaves an error rate larger than 1%. Secondly, and more importantly, it has a minimum redundancy of 2, regardless of f and the target error rate, ε_{acc}. For these reasons, voting is only practical in cases where: (1) f is small (i.e., $f <\approx 1\%$), and (2) either (a) we have enough idle or spare nodes to take on the extra load without causing additional slowdown (as in the case of hardware-redundant triple modular redundancy systems), or (b) a slowdown of at least 2 (or in general m) is deemed to be an acceptable price to pay to reduce errors.

[3] More precisely, $\varepsilon_{\text{majv}}(f,m) = \sum_{j=m}^{2m-1} \binom{2m-1}{j} f^j (1-f)^{(2m-1-j)}$, which is bounded by $\frac{(4f(1-f))^m}{2(1-2f)\sqrt{\pi(m-1)}}$ [8,10].

3.2 Spot-Checking

In cases where either f is large, or our maximum acceptable error rate is not too small, we can use a novel alternative we call *spot-checking*. In spot-checking, the master node does not redo *all* the work objects two or more times, but instead randomly gives a worker a *spotter* work object whose correct result is already known or will be known by checking it in some manner afterwards. Then, if a worker is caught giving a bad result, the master *backtracks* through all the results received from that worker so far, and invalidates all of them. The master may also *blacklist* the caught saboteur so that it is prevented from submitting any more bad results in the future.

Because spot-checking does not involve replicating all the work objects, it has a much lower redundancy than voting. If we assume that the master spot-checks each worker with a Bernoulli probability q, called the *spot-check rate*, then the redundancy, on average, will just be $1/(1-q)$. For example, if $q = 10\%$, then 10% of the work the master gives would be spotter works. This means that on average, the master gives out $(1/(1-0.1)) = 1.11N$ work objects during the course of a batch with N original work objects.

3.2.1 Spot-checking with Blacklisting

Even with this low redundancy, however, spot-checking can still achieve very low error rates. To see this, consider the case where caught saboteurs are *blacklisted* and never allowed to return or do any more work (at least within the current batch). In this case, errors can only come from saboteurs that survive until the end of the batch. Assuming that a saboteur is given a total of n work objects, including spotter works, during a batch (where n is the saboteur's share in the total work, i.e., N/P, plus the $1/(1-q)$ redundancy of spot-checking and the extra load that the remaining workers have to take when a worker gets blacklisted), then the *average final error rate with spot-checking and blacklisting*, $\varepsilon_{\text{scbl}}$, can be computed as:

$$\varepsilon_{\text{scbl}}(q, n, f, s) = \frac{sf(1-qs)^n}{(1-f) + f(1-qs)^n} \tag{1}$$

where s is the sabotage rate of a saboteur, f is the fraction of the original population that were saboteurs, $(1-qs)^n$ is the probability of a saboteur surviving through n turns, and the denominator represents the fraction of the original worker population that survive to the end of the batch, including both good and bad workers.

Closer analysis of this function [8] shows that it has maximum at roughly

$\hat{s}^*_{\text{scbl}}(q,n) = \min\left(1, \frac{1}{q(n+1)}\right)$ and has a maximum value that can be bounded as follows:

$$\hat{\varepsilon}^*_{\text{scbl}}(q,n) < \frac{f}{1-f} \cdot \frac{1}{qne} \tag{2}$$

Intuitively, this means that if a saboteur knows n in advance, then it should set its sabotage rate to be $\hat{s}^*_{\text{scbl}}$, since a higher sabotage rate would lead to a saboteur being caught too quickly, while having a lower sabotage rate would lead to fewer errors in the end. Even if saboteurs optimize their sabotage rates in this way, however, Eq. 2 says that the average error rate cannot be larger than $\hat{\varepsilon}_{\text{scbl}}$. That is, spot-checking reduces worst-case average error rate *linearly* with n (for a constant f). Thus, to reduce the error rate, it is to the master's advantage to make the batches longer so that n is larger.

3.2.2 Spot-checking without Blacklisting

Unfortunately, it may not always be possible to enforce blacklisting. Although we can blacklist saboteurs based on email address, it is not too hard for a saboteur to create a new email address and volunteer as a "new" person. Blacklisting by IP address would not work either because many people use ISPs that give them a dynamic address that changes every time they dial up. Requiring more verifiable forms of identification such as home address and a telephone number can turn away saboteurs, but would probably turn away many well-meaning volunteers as well.

It is thus useful to consider the effectivity of spot-checking when blacklisting cannot be enforced. Unfortunately, in these cases, saboteurs can increase the error rate significantly by leaving after doing only a limited number of work objects, l, and then rejoining under a new identity. We can show that this changes the upper bound on the worst-case average error rate to f/ql [8]. This is significantly worse than before, because unlike Eq. 2, this does not shrink inversely with n, and thus cannot be expected to shrink with the length of a batch. The best thing that a master can do in this case, is to try to force saboteurs to stay longer (i.e., increase l) by making it hard for them to forge a new identity or by imposing delays.

4 Credibility-based Fault-Tolerance

Although useful alone, voting and spot-checking can also be combined to increase reliability. One way to do so is to simply apply the two mechanisms independently on the same system. If blacklisting is enforced, then this simple hybrid approach effectively allows us to take the linearly-reduced error rate due to spot-checking and exponentially reduce it by voting. This allows us to achieve error rates that are orders-of-magnitude smaller than those of voting or spot-checking alone for the same slowdown [8].

Another, even better way of combining voting, spot-checking, as well as other mechanisms we may develop in the future, is through a new generalizable technique we call *credibility-based fault-tolerance.*

Here, the key idea is to compute the *credibility* of each tentative result as the conditional probability that the result is correct, based on voting (i.e., the more workers agreeing on a result, the higher its credibility), spot-checking (i.e., the higher the number of spot-checks passed by the workers who produced these results, the higher the credibility of the workers and the results), and other factors (e.g., human knowledge that some worker machines are more trustworthy than others). While the credibility of a result is below a certain *credibility threshold* ϑ, which we set to $\vartheta = 1 - \varepsilon_{\text{acc}}$, we continue to have it recomputed by other workers, and continue to spot-check workers. When the credibility threshold is reached (which can happen either because we gather enough matching results, or the workers pass enough spot-checks, or both), then we accept the result as final. By waiting for the threshold to be reached in this way, we guarantee that on average, the error rate will not exceed $1 - \vartheta = \varepsilon_{\text{acc}}$. At the same time, we allow the system to automatically combine voting and spot-checking, and efficiently trade-off redundancy for more reliability, using only as much redundancy as necessary to reach the threshold. The result is mathematically guaranteeable correctness with "minimal" (i.e., given what we can know and control) slowdown.

4.1 Implementation

We can implement this idea by attaching *credibility values* to different objects in the system, as shown in Fig. 1, where the *credibility* of some object X, written $Cr(X)$, is defined as an estimate of the conditional probability, given the current observed state of the system, that object X is, or will give, a good result. As shown, we have four different types of credibility: that of *workers* (Cr_P), *results* (Cr_R), *result groups* (Cr_G), and *work entries* (Cr_W). The credibility of a worker depends on its observed behavior such as the number of spot-checks it has passed, as well as other assumptions such as the upper bound on the faulty fraction, f. In general, we give less credibility to new workers who have not yet been spot-checked enough, and more credibility to those who have passed many spot-checks and are thus less likely to be saboteurs or have high sabotage rates. The credibility of a worker determines the credibility of its results, which in turn determine the credibility of the

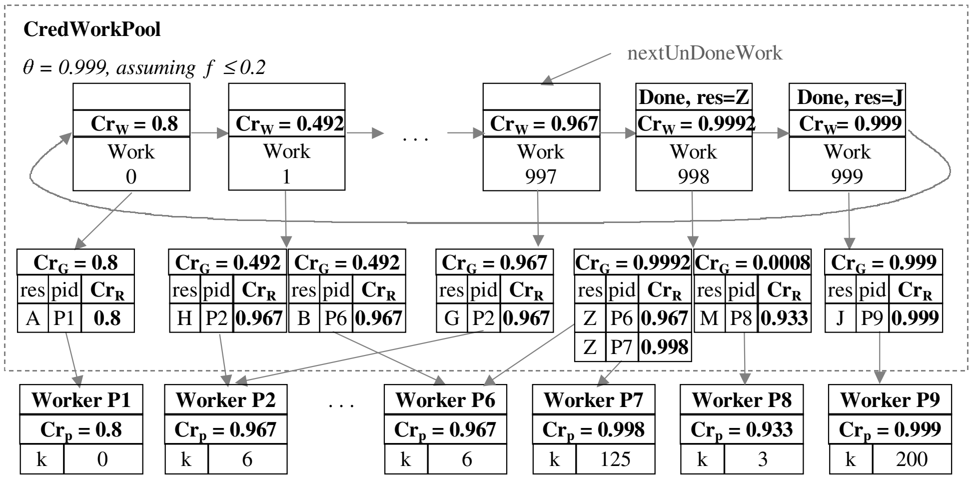

Fig. 3. A credibility-enhanced eager scheduling work pool (using Eq. 6 and Eq. 7).

result groups in which they respectively belong. The credibility of a result group (which is composed of matching results for a work entry) is computed as the conditional probability that its results are correct, given their credibilities, and the credibilities of other results in other result groups for the same work entry. Finally, the credibility of the *best* result group in a work entry gives us the credibility of the work entry itself, and gives us an estimate of the probability that we will get a correct result for that work entry if we accept its currently best result.

In the course of running a parallel batch, the credibilities of the objects in the system change as workers pass spot-checks or get caught, and as matching results are received for the same work entry. Assuming there are enough good workers, the credibility of each work entry W eventually reaches the desired threshold value ϑ. When this happens, the work entry is marked done and the server stops reassigning it to workers. When all the work entries reach the desired threshold ϑ, the batch ends. At this point, assuming that our credibilities are *good* estimates of the conditional probabilities they represent, the expected fraction of final results that will be correct should then be at least ϑ, and the error rate would thus be at most ε_{acc}.

4.2 Calculating Credibility

A key trick in this technique is computing the credibility values correctly. In general, there are many possible *credibility metrics*, corresponding to different ways of observing the current state of the system, as well as different ways of computing or estimating the conditional probability of correctness based on observations. In this section, we present particular metrics that we have found to be effective.

4.2.1 Credibility of Workers and their Results

Without Spot-checking. Without spot-checking, the credibility of a worker, and thus of its results, must be taken solely from assumptions that we are willing to make. In most cases, if we can assume a bound f on the faulty fraction of the worker population, then we simply let $Cr_P(P) = 1 - f$ for all workers, since f is the probability that a worker chosen at random would be bad. In some cases, we can assign some workers different credibilities based on human knowledge – e.g., workers from trusted domains get high credibility, while those from a domain that has had saboteurs in the past get low credibility.

With Spot-checking and Blacklisting. If we have spot-checking, then we can use the number of spot-checks passed by a worker, k, to estimate how likely a worker is to give a good result. Intuitively, the more spot-checks a worker passes, the more confident we can be that the worker is a good worker, or at least does not have a very high sabotage rate. (Note that we do not need to consider the credibility of workers who are spot-checked and caught, since these are removed from the system.)

If we have blacklisting, then we can compute the credibility of a worker $Cr_P(P)$ from k as one minus the conditional probability of receiving a bad result from a worker, given that the worker has survived k spot-checks. This probability is similar to that in Eq. 1, and can be computed and bounded as follows:

$$P(\text{result from } P \text{ is bad} \mid P \text{ survived } k \text{ spot-checks})$$

$$= \frac{sf(1-s)^k}{(1-f) + f(1-s)^k} \tag{3}$$

$$< \frac{f}{1-f} \cdot \frac{1}{ke} \text{ (for any } s\text{)} \tag{4}$$

This gives us the following credibility metric for spot-checking with blacklisting:

$$Cr_P(P)_{\text{scbl}} = 1 - \frac{f}{1-f} \cdot \frac{1}{ke} \tag{5}$$

which is a strict *lower bound* on the conditional probability of a worker P giving a *good* result.

Note that this equation does not apply to workers that have not yet been spot-checked, i.e., whose k is 0. In this case, we can just set $Cr_P(P) = 1 - f$. Alternatively, we can choose to just ignore results from workers that have not yet been spot-checked.

Without Blacklisting. Unfortunately, deriving a worker's credibility in the case when there is no blacklisting is not as straightforward. In general, the probability of errors is higher, so we need to assign lower credibilities to workers. Deriving an exact conditional probability like Eq. 5, however, is difficult, since saboteurs can leave and come back in under new identities, creating many different possible cases to consider. Thus, we take a different approach.

First, we note that if we assume that workers who leave or get caught rejoin immediately, then the faulty fraction of the worker population stays constant at around f. This implies that the probability of a randomly chosen worker being bad is around f, and thus the probability of a randomly chosen answer being bad is $f \cdot s$, where s is the sabotage rate of the saboteurs. Unfortunately, however, we do not know s. We can, however, derive a reasonable *estimate*, $\hat{s}$, based on k, and use that instead. One such estimate is $\hat{s} = 1/k$, which we can intuitively arrive at by noting that a saboteur with a sabotage rate of $1/k$ would have an average survival period of k spot-checks. Using this estimate gives us the following credibility metric for spot-checking with no blacklisting:

$$Cr_P(P)_{\text{scnb}} = 1 - \frac{f}{k} \tag{6}$$

As shown in Sect. 5.2, this metric proves to work well in simulations, where it always achieved the desired final error rate $1 - \vartheta$, without overly sacrificing performance.

Credibility of Results. For now, we will simply assume that the credibility of a result R, $Cr_R(R)$, is simply equal to $Cr_P(R.\text{solver})$ where $R.\text{solver}$ is the worker which produced the result. In general, however, it is possible to distinguish it from the solver's credibility. For example, we may give results received later lower credibility to guard against saboteurs who give good results at the beginning to earn credibility, but then start giving more bad results later on, once they know their credibility is high already.

4.2.2 Credibility of Result Groups and Work Entries

If a work entry W has only one result R_1 so far, then $Cr_W(W)$ is simply $Cr_R(R)$ of the result, which, under our assumptions, is equal to the credibility $Cr_P(R_1.\text{solver})$. If a work entry has several results, then we divide the results into g groups, G_a, for $1 \leq a \leq g$, with m_a members respectively, and then compute the credibility for each group based on the conditional probability of correctness, given the current combination of results received so far. This can be computed as:

$$Cr_G(G_a) \tag{7}$$

$$= \frac{P(G_a \text{ good})P(\text{all others bad})}{P(\text{get } g \text{ groups, where each } G_a \text{ has } m_a \text{ members})}$$

$$= \frac{P(G_a \text{ good}) \prod_{i \neq a} P(G_i \text{ bad})}{\prod_{j=1}^{g} P(G_j \text{ bad}) + \sum_{j=1}^{g} P(G_j \text{ good}) \prod_{i \neq j} P(G_i \text{ bad})}$$

where $P(G_a \text{ good})$ is the probability of all the results in G_a being good, computed as $\prod_{i=1}^{m_a} Cr_R(R_{ai})$ for all results R_{ai} in group G_a, and correspondingly, $P(G_a \text{ bad})$ is the probability of all the results in G_a being bad, given as $\prod_{i=1}^{m_a}(1 - Cr_R(R_{ai}))$. Figure 3 show some examples of how Eq. 7 is used (note especially works 1 and 998).

4.3 Using Credibility

4.3.1 Voting and Spot-checking Combined

Although credibility-based fault-tolerance can be used with voting alone or spot-checking alone, it is best used to integrate voting and spot-checking together. In this case, we start with all workers effectively having a credibility of $1 - f$ and start collecting results. If the credibility threshold ϑ is low enough, and the batch is long, then by the time we go around the circular work pool, the workers may have already gained enough credibility by passing spot-checks to make their results acceptable. In these cases, we do not need to do voting and we can reach our desired error rates with only $1/(1 - q)$ redundancy due to the spotter works. If ϑ is high, then spot-checking would not be enough, so we start reassigning work, collecting redundant results, and voting.

If we did not use spot-checking, we would eventually reach the threshold after a slowdown proportional to roughly $\log(1 - \vartheta)/\log(1 - Cr_P(P)) = \log_f \varepsilon_{\text{acc}}$. Spot-checking, however, effectively reduces the base of this logarithm, $1 - Cr_P(P)$, linearly in time, and thus allows us to reach the desired threshold in much less time than with voting alone. In Sect. 5.2, for example, we show that at $f = 20\%$ and $q = 10\%$, $N = 10000$, and $P = 200$, we can reach an error rate of 1×10^{-6} with an average slowdown of only around 3 compared to m-first voting's 32.

Another advantage of using credibility is that it works well even if we cannot enforce blacklisting. By using the credibility metric from Eq. 6, we effectively neutralize the effect of saboteurs who only do a few pieces of work and then rejoin under a new identity. As shown in Sect. 5.2, there is now no advantage to doing so, and in fact, it seems that there is now more incentive for a saboteur to stay on for longer periods.

4.3.2 Using Voting for Spot-checking

Although using credibility with voting and spot-checking already works quite well, we can gain even more performance by using voting *for* spot-checking.

So far, we have assumed that a master spot-checks a worker by giving it a piece of work whose correct result is already known. Since this implies that either the master itself, or one of a few fully-trusted workers, must do the work to determine the correct result, we generally assume that q needs to be small (i.e., less than 10%). Since k is roughly qn, this limits the rate at which credibilities increase and thus limits performance.

Fortunately, we can attain much better performance by using credibility-based voting as a spot-checking mechanism. That is, whenever one of a work entry's result groups reaches the threshold (such that the work entry can be considered done), we increment the k value of the solvers of the results in the winning group, while we treat those in the losing groups as if they had failed a spot-check (i.e., we remove them from the system and invalidate their other results).

If we assume that we have to do all the work at least twice, which implies that *all* results returned by a worker would have to participate in a vote, then using these votes to spot-check a worker implies that a worker will get spot-checked at least $k = n$ times – i.e., $1/q$ times more than before. This implies a corresponding decrease in the error rate and a corresponding increase in the credibility of good workers, which in turn allows the voting to go even faster.

Note that this technique is only made possible by using credibility-based voting to begin with. Naïvely using traditional majority voting to spot-check workers would be dangerous because the chance of saboteurs outvoting good workers and thus getting them blacklisted would be significant, especially if f is not small. Credibility-based voting works because it guarantees that we do not vote until the probability that the vote will be right is high enough. Thus, it limits the probability of good workers being outvoted to a very small value. Note, however, that some "bootstrapping" is required here. That is, we cannot start using voting for spot-checking until the result groups actually start reaching the threshold and voting. This implies that: (1) spot-checking by voting is only beneficial when the redundancy is already at least 2, and (2) we need to maintain normal spot-checking (at least for the first few batches) to allow the workers to gain enough credibility to reach the threshold early enough.

5 Simulation Results

5.1 The Simulator

To verify our theoretical results, we have developed a Monte Carlo simulator that simulates the behavior of an eager scheduling work pool in the presence of saboteurs and various fault-tolerance mechanisms [8]. To simulate the workers and saboteurs, we create a list of P *worker entries* and randomly select fP of them to be saboteurs. We then simulate a computation done by these workers by going through the list in round-robin manner, each time simulating the action of the current worker contacting the master to return a result (for the work object it received in its *previous* turn) and to get new work. This assumes, as in Sect. 2, that all workers have exactly the same speed, so that the work is equally distributed among the workers, and each worker gets to take a turn before any other worker is allowed to take a second turn.

For our experiments, we ran 100 runs of simulated computation, each consisting of a sequence of 10 batches of $N = 10000$ work objects each, done by $P = 200$ workers. These numbers were chosen to be small enough to be simulatable in a reasonable amount of time, but large enough to provide good precision (i.e., the smallest measurable error rate is 1×10^{-7}) and to prevent blacklisting from killing all the saboteurs too early. In addition, the work-per-worker ratio, $N/P = 50$, was chosen to be large enough to show the effects of spot-checking, while still being representative of potential real applications. Also, having the computation go through 10 batches allows us to see the benefits of letting good workers gain higher credibility over time. When doing blacklisting, we only do *batch-limited blacklisting*, which means that we allow blacklisted nodes to return at the start of the *next* batch. However, these return with a different worker ID and a clear record. Specifically, a returning saboteur's k is set back to 0 and its credibility is correspondingly reset.

5.2 Results

Figures 4 to 9 show the experimental results we get from running our Monte Carlo simulator.

Figure 4 plots the resulting slowdown and error rate from majority voting given different values of the initial faulty fraction f (assuming a sabotage rate of 1). (This graph is like Fig. 2 turned on its side, except that m is replaced by slowdown, and the values of f are different.) As shown, when f is large, majority voting requires a lot of redundancy to achieve even relatively large error rates. Extending the line for $f = 0.2$ theoretically, we find that it would

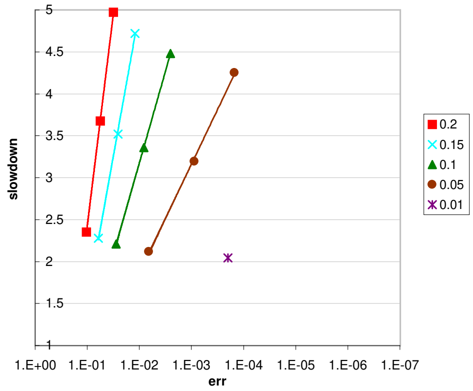

Fig. 4. Majority voting. Slowdown vs. maximum final error rate at various values of f and $m = \{2, 3, 4\}$.

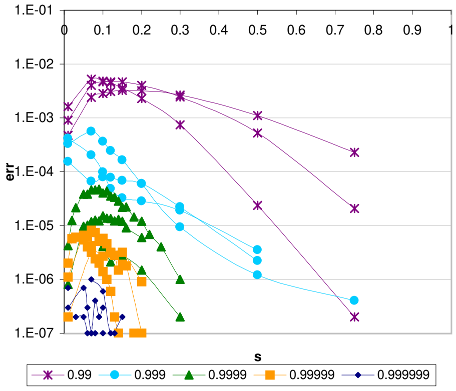

Fig. 5. Credibility-based voting with spot-checking and blacklisting. Error rate vs. s at $f = \{0.2,\ 0.1,\ 0.05\}$.

take a slowdown of more than 32 to achieve a final error rate of 1×10^{-6}. Note, however, that the slope becomes less steep as f becomes smaller. (Only one point for $f = 0.01$ is shown because the other points resulted in no errors in our experiments.)

Figure 5 shows the results of using credibility-based voting and spot-checking with blacklisting, using the credibility metric $Cr_P(P)_{\mathrm{scbl}}$ from Eq. 5. Here, each group of points corresponding to a credibility level is divided into three curves corresponding to $f = 0.2,\ 0.1$ and 0.05, respectively. Most significantly, this plot shows that, as intended, the average error rate never goes above $1 - \vartheta$, regardless of s and f.

One thing that is not shown in Fig. 5 is that while the maximum error rate remains roughly the same (as limited by $1 - \vartheta$), more and more redundancy is being needed to guarantee the bounds on the error rate. Figure 6 shows the slowdown incurred in achieving the *maximum* error rate for a particular value of f and ϑ. Note how the slopes of the lines here are much better than those in simple majority voting, thus allowing us to achieve lower error rates in less time. For example, whereas majority voting would have required a slowdown

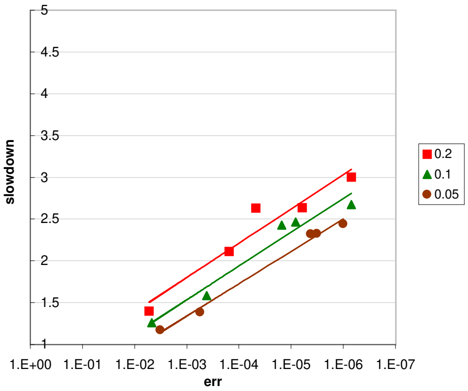

Fig. 6. Credibility-based voting with spot-checking and blacklisting. Slowdown vs. maximum final error rate at $\vartheta = 0.99, \ldots, 0.99999$ at various values of f.

of more than 32 to achieve an error rate of 1×10^{-6} for $f = 0.2$, here we only need around 3. Also note that in some cases, spot-checking can be enough to reduce f down to the threshold $1 - \vartheta$ without requiring voting, as shown by the points with slowdown less than 2.

Figure 7 shows how credibility-based fault-tolerance works even in cases without blacklisting, wherein saboteurs come back under a new identity after they are caught, or after doing l work objects without being caught.[4] In this case, we use the credibility metric $Cr_P(P)_{\mathrm{scnb}}$ from Eq. 6, and measure the error rate at various values of s for $f = 0.2$ and $\vartheta = 0.9999$. As shown, even without blacklisting, we successfully guarantee that the error rate never exceeds $1 - \vartheta = 1 \times 10^{-4}$, regardless of l. Interestingly, although error rates start high at $l = 1$ and decrease with l as predicted in Sect. 3.2.2, at some point above $l = 120$, the error rates get dramatically larger, and stay roughly constant. It is not clear why this happens, but we suspect that it is because saboteurs who stay until the next batch gain are able to carry over their credibility record and cause more errors in succeeding batches.

Finally, Figs. 8 and 9 show the results of using credibility-based voting to spot-check workers. Figure 8 shows how it guarantees that the error rate threshold is reached (in fact, it seems that error rates from this scheme tend to be smaller overall), and Fig. 9 shows the slowdown. As shown, the slope here is even better than that of the case with blacklisting. Here, we can now achieve an error rate of less than 1×10^{-6} from $f = 0.2$, with just a little over 2.5 redundancy. Comparing this with majority voting as shown in Fig. 4, this shows that for the same slowdown, we get an error rate which is almost 10^5 times better.

[4] We assume pessimistically that a saboteur knows when it is caught. If a master does not tell a saboteur that it has been caught but simply ignores its results, then we expect to get better error rates.

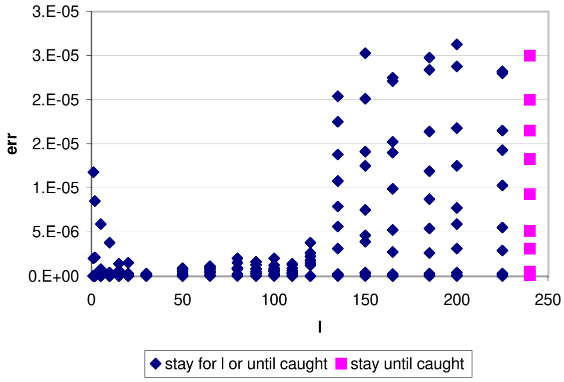

Fig. 7. Credibility-based voting with spot-checking, no blacklisting. Error rate vs. length-of-stay l at $f = 0.2$ and $\vartheta = 0.9999$.

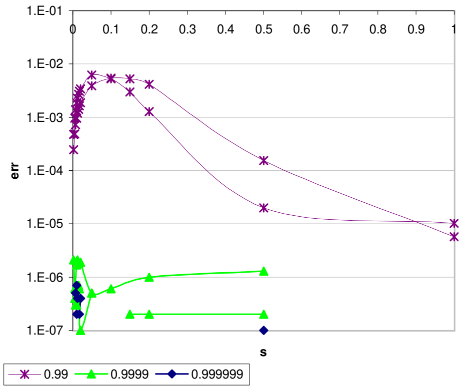

Fig. 8. Credibility-based voting with spot-checking by voting, no blacklisting. Error rate vs. s at $f = 0.2, 0.1, 0.05$ for various thresholds ϑ, assuming saboteur stays until caught and then rejoins immediately.

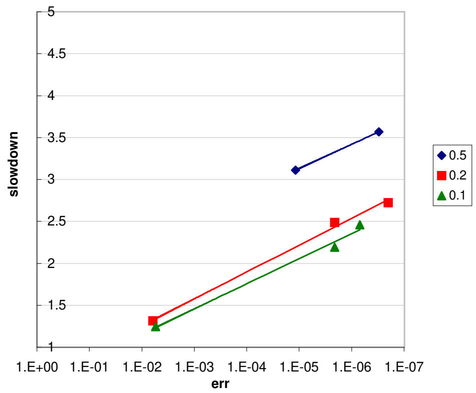

Fig. 9. Credibility-based voting with spot-checking by voting, no blacklisting. Slowdown vs. maximum final error rate at $\vartheta = 0.99$, 0.9999, and 0.999999 at various values of f, assuming saboteur stays until caught and then rejoins immediately.

6 Conclusion

In this paper, we have proposed new mechanisms for addressing the largely unstudied problem of sabotage-tolerance, and have demonstrated the potential effectivity of these mechanisms through mathematical analysis and simulation. A logical next step for research at this point, is then to implement and apply these techniques to real systems, and start benefitting from them. This should not be too difficult because the master-worker model to which these mechanisms apply is widely used today not only in volunteer computing systems but in other metacomputing and grid computing systems as well.

In the process of applying these mechanisms, questions may arise with respect to assumptions or implementation details. Some variations that we can explore in further research, for example, include:

- **Handling cases where saboteurs can collude on when to vote together.** This would imply a change in $P(G_a \text{ bad})$ in Eq. 7.
- **Incorporating the use of checksums.** A worker which submits a result that fails a checksum would be treated as if it had been spot-checked and caught submitting a bad result.
- **Incorporating the use of encrypted computation, obfuscation, and other techniques.** This would decrease the probability that a saboteur can successfully make a bad answer look good. This would thus increase the credibility of workers and results. By using a different key for each worker, we can also make it harder for colluding saboteurs to produce matching bad results. This would change $P(G_a \text{ bad})$ accordingly, since the probability that groups would be formed by bad results would now be very small.
- **Handling saboteurs that are not Bernoulli processes.** These include saboteurs who might start with giving good answers, but then start giving bad answers when their credibility has become high enough. One possibility here is to increase the spot-check rate when necesary. Another possibility is to compute the credibility of a worker based only on its more recent spot-checks.

In the light of these and other questions, one of the most significant contributions of this paper is the *generality* of the idea of credibility-based fault tolerance. That is, we can apply this technique with other mechanisms or under other assumptions as long as we can derive the net effect of the new mechanisms or assumptions on the conditional probabilities of results being correct. Thus, through this paper, we not only present concrete mechanisms which can already be applied today, but also open up the field for the development of other mechanisms as well.

References

[1] A. Baratloo, M. Karaul, Z. Kedem, and P. Wyckoff, Charlotte: Metacomputing on the Web, in Proc. 9th Intl. Conf. on Parallel and Distributed Computing Systems (1996).
`http://cs.nyu.edu/milan/charlotte/`

[2] A. L. Beberg, J. Lawson, D. McNett, `distributed.net` home page.
`http://www.distributed.net`

[3] Entropia home page.
`http://www.entropia.com/`

[4] P. Cappello, B.O. Christiansen, M.F. Ionescu, M.O. Neary, K.E. Schauser, and D. Wu, Javelin: Internet-Based Parallel Computing Using Java, in *Proc. ACM Workshop on Java for Science and Engineering Computation* (Las Vegas, 1997).
`http://javelin.cs.ucsb.edu/`

[5] Parabon home page. URL: `http://www.parabon.com/`

[6] Popular Power home page.
`http://www.popularpower.com/`

[7] Process Tree home page.
`http://www.processtree.com/`

[8] L.F.G. Sarmenta, *Volunteer Computing*, Ph.D. thesis. Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA, March, 2001.

[9] SETI@home.
`http://setiathome.ssl.berkeley.edu/`

[10] Y. A. Zuev, The Estimation of Efficiency of Voting Procedures, in *Theory of Probability and its Applications*, Vol. 42, No. 1, March, 1997.
`http://www.siam.org/journals/tvp/42-1/97594.html`