Characterizing Result Errors in Internet Desktop Grids

Derrick Kondo, Filipe Araujo, Paul Malecot, Patricio Domingues, Luis Moura Silva, Gilles Fedak, Franck Cappello

Characterizing Result Errors in Internet Desktop Grids

Derrick Kondo[1] Filipe Araujo[2] Paul Malecot[1] Patricio Domingues[3]
Luis Moura Silva[2] Gilles Fedak[1] Franck Cappello[1]

[1]INRIA Futurs, France
[2]University of Coimbra, Portugal
[3]Polytechnic Institute of Leiria, Portugal
[dkondo, malecot, fedak, fci]@lri.fr,
[filipius, luis]@dei.uc.pt, patricio@estg.ipleiria.pt

Abstract

Desktop grids use the free resources in Intranet and Internet environments for large-scale computation and storage. While desktop grids offer a high return on investment, one critical issue is the validation of results returned by participating hosts. Several mechanisms for result validation have been previously proposed. However, the characterization of errors is poorly understood. To study error rates, we implemented and deployed a desktop grid application across several thousand hosts distributed over the Internet. We then analyzed the results to give quantitative, empirical characterization of errors rates. We find that in practice, error rates are widespread across hosts but occur relatively infrequently. Moreover, we find that error rates tend to not be stationary over time nor correlated between hosts. In light of these characterization results, we evaluated state-of-the-art error detection mechanisms and describe the trade-offs for using each mechanism. Finally, based on our empirical results, we conduct a benefit analysis of a recently proposed mechanism for error detection tailored for long-running applications. This mechanism is based on using the digest of intermediate checkpoints, and we show in theory and simulation that the relative benefit of this method compared to the state-of-the-art is as high as 45%.

1 Introduction

Desktop grids use the free resources in Intranet and Internet environments for large-scale computation and storage. For over 10 years, desktop grids have been one of the largest distributed systems in the world providing TeraFlops of computing power for applications from a wide range of scientific domains, including climate prediction [12], computational biology [26], and physics [1]. Despite the huge computational and storage power offered by desktop grids and their high return on investment, there are several challenges in using this volatile and shared platform effectively. One critical issue is validation of results computed by insecure and possibly malicious hosts. For example, in [28], the authors report that errors can be caused by both hardware or software errors (for example, CPU's corrupted by overclocking, or by incorrect modifications of the application or desktop grid software). (Zealous project participants often try to increase their ranking in the project's list of the most productive users.) Other times, errors can be caused by a malicious user who for example submit fabricated results [19]. For these reasons, effective error detections mechanisms are essential and several meth-

ods have been proposed previously [23, 29].

However, little is known about the nature of errors real systems. Yet, the trade-offs and efficacy among different error detection mechanisms is dependent on how errors occur in real systems. Thus, we investigate errors in a real system by focusing on the following critical questions:

1. What is the frequency and distribution of host error rates?
2. How stationary are host error rates?
3. How correlated are error rates between hosts?
4. In light of the error characterization, what is the efficacy of state-of-the-art error detection mechanisms and can new mechanisms be proposed?

To help answer those questions, we deployed an Internet desktop grid application across several thousand desktop hosts. The results of the application returned by the hosts were then validated. The invalid results were then analyzed to characterize quantitatively the error rates in a real desktop grid project, which we describe in the following sections.

The paper is organized as follows. In Section 2, we define the basic terminology used throughout the paper. In Section 3, we describe related work in terms of error characterization and detection. In Section 5.2, we study the stationarity of host error rates over time. In Section 5.3, we study the correlation of errors between hosts. In Section 6, based on our empirical results, we present a benefit analysis for a recently proposed mechanism for error detection based on intermediate checkpoints and tailored for long-running applications.

2 Background

At a high level, a typical desktop grid system consists of a server from which **workunits** of an **application** are distributed to a **worker** daemon running on each participating host. The workunits are then executed when the CPU is available, and upon completion, the **result** is return back to the server. We define a result **error** to be any result returned by a worker that is not the correct value or within the correct range of values. We call any host that has or will commit at least one error an **erroneous host** (whether intentionally or unintentionally).

Workunits of an application are often organized in groups of workunits or **batches**. To achieve overall low rates for a batch of tasks, the individual error rate per host must be made small. Consider the following scenario described in [23] where a computation consists of 10 batches, each with 100 workunits. Assuming that any work unit error would cause the entire batch to fail, then to achieve an overall error rate of 0.01, the probability of a result being erroneous must be no greater than 1×10^{-5}. Many applications (for example, those from from computational biology [28] and physics [1]) require (low) bounds on error rates as the correctness of the computed results are essential for making accurate scientific conclusions.

3 Related Work

3.1 Characterizing Errors

To the best of our knowledge, there has been no previous study that gives quantitative estimates of error rates from empirical data. Several previous works [11, 15] study *failure* rates of executing tasks, where a failure is any event that causes a task's execution to terminate. However, the definition of failures in those studies is different from the notion of errors as it does not take into account result correctness; a workunit's execution could fail, but the result computed eventually could be entirely correct and would not be considered an error. While certain failures may be correlated to result errors, the relation has not been studied in detail.

Parameter	Definition
f	Fraction of hosts that commit at least one error
s	Error rate per host
φ	Probability that a worker returns an erroneous result
ε	Fraction of results that will be erroneous
m	Number of identical results before a vote is considered to be complete
q	Frequency of spot-checking
n	Amount of work contributed by the erroneous worker
c	Number of segments or equivalently checkpoints per task
R	Number of workers on which a checkpointed task is replicated
X	Random variable distributed geometrically with parameters p and v representing the number of task segments before an error occurs

Table 1. Parameter Definitions.

3.2 Validating Results

In this section, we discuss three of the most common state-of-the-art methods [25, 23, 24, 29, 28] for reducing error rates in desktop grids namely spot-checking, majority voting, and credibility-based techniques, and emphasize each of their advantages and highlight the assumptions on which they are based.

The **majority voting** method detects erroneous results by sending identical workunits to multiple workers. After the results are retrieved, the result that appears most often is assumed to be correct. In [23], the author determines the amount of redundancy for majority voting needed to achieve a bound on the frequency of voting errors given the probability that a worker returns a erroneous result. Let the error rate φ be the probability that a worker is erroneous and returns an erroneous result unit, and let ε be the percentage of final results (after voting) that are incorrect. Let m be the number of identical results out of $2m-1$ required before a vote is considered complete and a result is decided upon. Then the probability of a incorrect result being accepted after a majority vote is given by:

$$\varepsilon_{majv}(\varphi, m) = \sum_{j=m}^{2m-1} \left(\begin{array}{c} 2m-1 \\ j \end{array} \right) \varphi^j (1-\varphi)^{2m-1-j} \quad (1)$$

From Equation 1, the author shows that the error rate decreases exponentially. So voting is especially effective when the error rate is small. However, when the fault rate is relatively large, increasing redundancy does not significantly reduce the error rate; for example, when $\varphi = 20\%$, the error rate is still more than 1% when $m = 6$.

The redundancy of majority voting is $\frac{m}{1-f}$, where f is fraction of hosts can commit at least one error. Even if a workunit is replicated just twice, the performance of the entire system will be cut in half. Another potential drawback to this method is that it is susceptible to correlated failures, as the bounds computed for ε_{majv} assume that error occur independently among hosts; if hosts collude together often and conduct a coordinated attack, majority voting may not be beneficial.

Thus, voting is an effective method only when the error rate is independent among hosts, and the error rate is relatively small ($< 1\%$), or when there is an abundance of resources to limit the performance degradation resulting from duplicated workunits, or when the performance degradation is acceptable.

Another method for error detection is **spot-checking**, whereby a workunit with a known correct result is distributed at random to workers. The workers' results are then compared to the previously computed and verified result. Any discrepancies cause the corresponding worker to be **blacklisted**, i.e., any past or future results re-

turned from the erroneous host are discarded (perhaps unknowingly to the host).

Erroneous workunit computation was modelled as a Bernoulli process [23] to determine the error rate of spot-checking given the portion of work contributed by the host, and the rate at which incorrect results are returned. The model uses a work pool that is divided into equally sized batches and it assumes the following. First, an upper bound f on the percentage of the total number of workers that are corrupt can be determined. Seconds, the rate s at which a corrupt worker sends erroneous results is constant across erroneous workers, and does not fluctuate. Moreover, when $s < 1$, erroneous workers decide when to send in erroneous results independently of one another. When multiple workers do send in bad results for the same workunit, those results match. (Note that this is a worst case scenario and makes the probabilistic results stronger than in reality.)

Allowing the model to exclude coordinate attacks, let q be the frequency of spot-checking, and let n be the amount of work contributed by the erroneous worker. $(1 - qs)^n$ is probability that erroneous host is not discovered after processing n workunits. The rate which spot-checking with blacklisting will fail to catch bad results is given by:

$$\begin{aligned}\varepsilon_{scbl}(q, n, f, s) &= s * P(\text{erroneous worker} \mid \\ &\qquad \text{it passed previous spot-checks}) \\ &= \frac{sf(1-qs)^n}{(1-f) + f(1-qs)^n}\end{aligned} \tag{2}$$

The denominator gives the portion of workers (both good and bad) remaining after all spot-checks have occurred for the batch of workunits.

This shows that the error rate for spot-checking is inversely proportional to the number of workunits computed per worker; the error rate decreases linearly with n, and so batches should be chosen to be as large as possible to increase the chance that an erroneous host will be detected.

The advantage of this approach is that the amount of redundant computation is negligible (especially when compared to the majority voting method described previously, which uses redundant computation to validate results). In particular, the amount of redundancy of spot-checking is given by $\frac{1}{1-q}$.

The disadvantage of spot-checking is the difficulty of effectively blacklisting an erroneous host, when it can register under new identities at will or if there is high host churn as shown by [6]. Moreover, blacklisting may be harmful if it removes from the project workers that unintentionally and infrequently return invalid workunits. However, without blacklisting, the upper bound on the error is much higher and does not decrease inversely with n.

Another way of reducing workunit errors is to use conditional probabilities of errors, given the history of host result correctness. A system based on this principle is called a **credibility-based** system. Due to space limitations, we only describe the method at a high-level, and details can be found in [25]. The idea is based on the assumption that hosts that have computed many results with relatively few errors have a higher probability of errorless computation than hosts with a history of returning erroneous results. Workunits are assigned to hosts such that more attention is given to the workunits distributed to higher risk hosts.

To determine the credibility of each host, any error detection method such as majority voting, spot-checking, or various combinations of the two can be used. The credibilities are then used to compute the conditional probability of a result's correctness. As such, this method, like spot-checking, assumes that the error rate per host remain consistent over time as it uses the conditional probability of errors.

In summary, blacklisting is a method for preventing errors by removing hosts from the system. Majority voting is a method for detecting and correcting errors by replicating workunits on multiple hosts. Spot-checking is a method for detecting

erroneous hosts by sending workunits to workers randomly for which the correct result is known. Credibility-based systems (which could be based on majority voting or spot-checking or both) reduce the probability errors by assign workunits to hosts, based on their past performance.

4 Method

We studied the error rates of a real Internet desktop grid project called XtremLab [2, 18]. XtremLab uses the BOINC infrastructure [9, 5] to collect measurement data of desktop resources across the Internet. The XtremLab application currently gathers CPU availability information by continuously computing floating point and integer operations, and every 10 seconds, the application will write the number of operations completed to file. Every 10 minutes, the output file is uploaded to the XtremLab server.

In this study, we analyze the outputs of the XtremLab application to characterize the rate at which errors can occur in Internet-wide distributed computations. In particular, we collected traces between April 20, 2006 to July 20, 2006 from about 4400 hosts. From these hosts, we obtained over 1.3×10^8 measurements of CPU availability from 2.2×10^6 output files. We focused our analysis on about 600 hosts with more then 1 week worth of CPU time in order to ensure the statistic significance of our conclusions. (Note that when we increased the threshold to 2, 3, or 4 weeks, the conclusions of our analysis did not change.)

Errors in the application output are determined as follows. Output files uploaded by the workers are processed by a validator. The validator conducts both syntactical and semantics checks of the output files returned by each worker. The syntactical checks verify the format of the output file (for example, that the time stamps recorded were floating numbers, the correct number of measurements were made, and each line contains the correct number of data). The semantic checks verify the correctness of the data to ensure that the values reported fall in the range of feasible CPU availability values. Any output files that failed these checks were marked as erroneous, and we assume any output file that fails a syntactic or semantic check would correspond to an error of a workunit in a real desktop grid project.

To date, there has been little specific data about error rates in Internet desktop environments, and we believe that the above detection method gives a first-order approximation of the error rates for a real Internet desktop grid project. However, there are limitations in our methodology, which include the following. First, the method measures the error rates of only a single, compute-intensive application. While we believe this application is representative of most Internet desktop grid applications in terms of its high ratio of computation compared to communication, applications with different IO or computation patterns could potentially differ in error rates. Nonetheless, we believe this is the first study of a real project to give quantitative estimates of error rates. Second, the method will not be able to detect all possible errors, for example those errors that cause workunits to be correct both syntactically and semantically. Third, we cannot determine the exact cause of errors. For example, if an output file is corrupt, we cannot determine whether the cause was due to hardware malfunction, software code modifications, a malicious user, or any other cause. Nevertheless, we do not believe the errors are due to network failures during transfers of output files. This is because BOINC has a protocol to recover from failures (of either the worker of server) during file transmission that ensures the integrity of files transfers [9]. In this protocol, the length of the transfer is transmitted before the actual file, and so the server is able to determine if the transfer was completed. If a failure occurs during transmission, the worker will retry sending the file later from where it left off before the failure. Thus, we believe most errors occurred on the host machine itself.

Detecting all possible causes of errors in large-scale distributed systems and being able to detect all possible symptoms of those causes is a challenging and open issue, and we will refine our validation process in future work.

5 Error Characterization

5.1 Distribution of Error Rates

The effectiveness of different methods by which errors are detected is heavily dependent on the distribution of errors among hosts. We measured the fraction of workunits with errors per host, and show the cumulative distribution of these fractions in Figure 1. The mean error rate was 0.002, and the max error rate was 0.098.

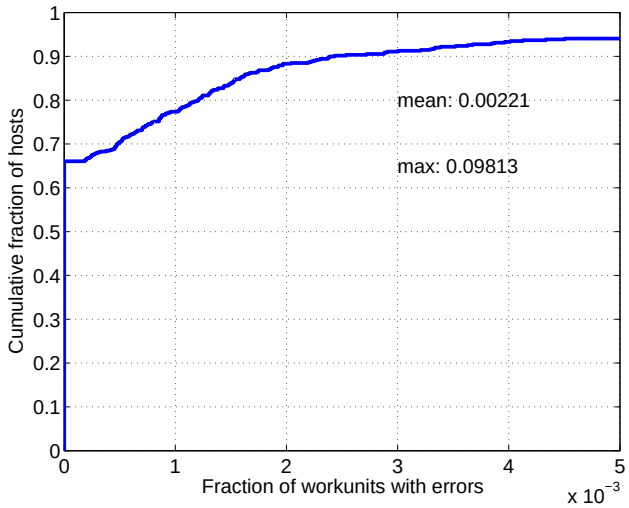


Figure 1. Error Rates of of Hosts in Entire Platform

We find that a remarkably high percentage of hosts (about 35%) returned at least one corrupt result in the 3 month time frame. Given that the overall error rate is low and a significant fraction of hosts result in at least one error, blacklisting all erroneous may not be an efficient way of preventing errors.

An error rate of 0.002 may seem so low that error correction, detection and prevention are moot, but consider the scenario in Section 2 again where the desired overall error rate is 0.01. In that case, the probability of a result being erroneous must be no greater than 1×10^{-5}. If $f \times s = 0.002$ as shown in Figure 1, we can simply replicate each workunit twice to achieve an error rate less then $1 \times 10^{-}5$. That is, if $m = 2$, then the error rate given by Equation 1 is $\epsilon_{majv} = .000004$, and the redundancy is about 2.00.

While spot-checking can achieve a similar error rate of about $1 \times 10^{-}5$, spot-checking requires a large number of workunits to be processed before achieving it. For example, to achieve a similar error rate of 1×10^{-5} via spot-checking where $q = .10$, $f = 0.35$ (from Figure 1), $s = 0.003$ (as shown in Table 2), it requires that the number of workunits (n) processed by each worker be greater than 5300 by Equation 2. While redundancy is lower at 1.11 compared to majority voting, if each workunit requires 1 day of CPU time (which is a conservative estimate as shown here [10]), it would require at least 14.5 years of CPU time *per worker* before the desired rate could be achieved. Even if we increase q to 0.25 (and redundancy is 1.33), spot-checking requires $n = 3500$ (or at least 9.5 years of CPU time per worker assuming a workunit is 1 day of CPU time in length).

Next, we focused on characterizing the hosts that returned at least one corrupt result. Figure 2 shows the cumulation distribution of the fraction of workunits with errors for all hosts with at least one corrupt result. We observe that about 80% of the hosts have error rates of .005 or less. The mean error rate over this set of hosts is 0.0065, and the maximum is 0.098.

Figure 3 shows the skew of the distribution of errors among those erroneous hosts. In particular, we sort the hosts by the total number of errors they committed, and the blue, solid plot in Figure 3, shows the cumulation fraction of errors. For example, the point (0.10, 0.70) shows that the

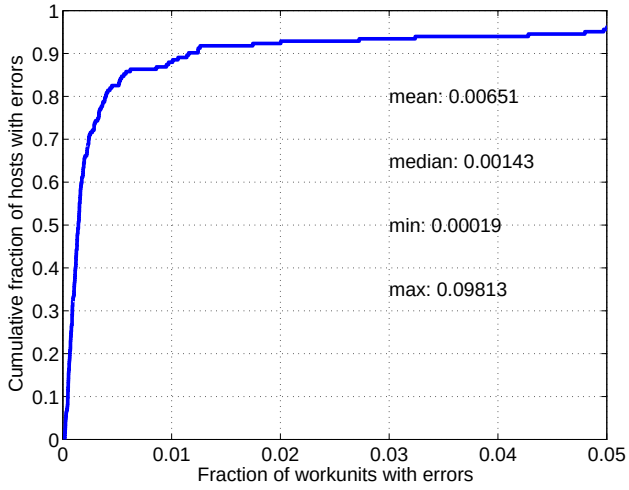


Figure 2. Error Rates of Erroneous Hosts Only

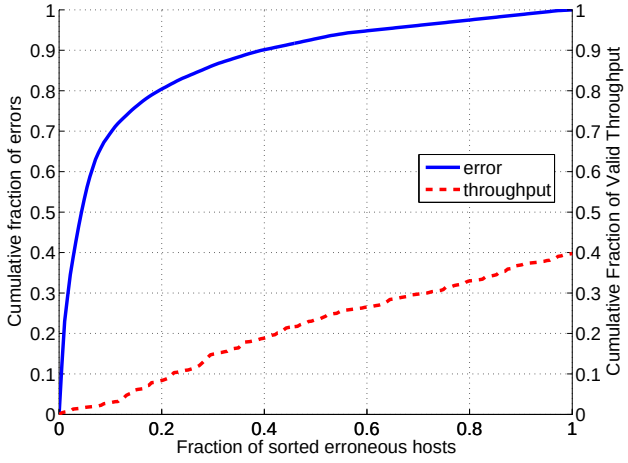


Figure 3. Cumulative Error Rates and Effect on Throughput

top 0.10 of erroneous hosts commit 0.70 of the errors. Moreover, the remaining 0.90 of the hosts cause only 0.30 of the errors. We refer to the former and latter groups as **frequent** and **infrequent offenders**, respectively.

Figure 3 also shows the effect on throughput if the top fraction of hosts are blacklisted, assuming than an error is detected immediately and that after the error is detected, all workunits that had been completed previously by the host are discarded. If all hosts that commit errors are blacklisted, then clearly throughput is negatively affected and reduced by about 0.40. Nevertheless, blacklisting could be a useful technique if it is applied to the top offending hosts. In particular, if the top 0.10 of hosts are blacklisted, this would cause less than a 0.05 reduction on the valid throughput of the system while reducing errors by 0.70. One implication of these results is that an effective strategy to reduce errors could focus on eliminating the small fraction of frequent offenders in order to reduce the majority of errors without having a negative effect on overall throughput.

So we also evaluated majority voting and spot-checking in light of the previous result, by dividing the hosts into two groups, frequent and infrequent offenders based on the knee of the curve shown in Figure 3, but a similar problem described earlier occurs. The error for majority voting ε_{majv} is given by Equation 1, where $\varphi = f_{all} \times s_{frequent} \times f_{frequent} + f \times s_{infrequent} \times f_{infrequent}$. f_{all} is simply the fraction of workers that could result in at least one error (0.35). $s_{frequent}$ (0.0335) and $s_{infrequent}$ (0.001) (see Table 2) are the error rates for frequent and infrequent offenders, respectively. $f_{frequent}$ (0.10) and $f_{infrequent}$ (0.90) are the fraction of erroneous workers in the frequent and infrequent groups respectively.

We plot ε_{majv} as a function of m in Figure 4. We find that the error rate ε_{majv} decreases exponentially with m, beginning at about 1×10^{-5} for $m = 2$.

We also compute the error rate for spot-checking with blacklisting when dividing the hosts in terms of frequent and infrequent offenders. The error rate ε_{scbk} is given by the sum of the error rates for each grouping, $\varepsilon_{scbk,frequent}$ and $\varepsilon_{scbk,infrequent}$. $\varepsilon_{scbk,infrequent}$ is given by substituting $f_{all} \times f_{frequent}$ for f and $s_{frequent}$ for s in Equation 2. $\varepsilon_{scbk,infrequent}$ can be calculated similarly.

Then we plot in Figure 5 $\varepsilon_{scbk,frequent}$, $\varepsilon_{scbk,infrequent}$, and ε_{scbk} as a function of n (the number of workunits that must be computed by each worker) where $q = 0.10$. The plot for the frequent offenders decreases exponentially; this is because the error rate for the hosts is relatively high, and so after a series of workunit computations, the erroneous hosts are rapidly detected. The plot for the infrequent offenders decreases very little even as n increases significantly. This is because the error rates for the infrequent offenders are relatively low, and thus, increasing n does not improve detection nor reduce error significantly. The effect of the net error rate ε is that it initially decreases rapidly for n in the range [0, 1000]. Thereafter, the error rate decreases little. We also looked at much larger ranges, and the decrease in error rate was relatively small.

The conclusion is that spot-checking acts as a low-pass filter in the sense that hosts with high error rates can be easily detected (and can then be blacklisted); however, hosts with low error rates remain in the system. Specifically, spot-checking can reduce error rates down to about 1×10^3 quickly and efficiently. To achieve lower error rates, one must use majority voting. In the next section, we show that spot-checking may have other difficulties in real-world systems.

5.2 Stationarity of Error Rates

Intuitively, a process is stationary if its statistical properties do not change with time. In particular, a stationary process will have a constant mean. In this section, we investigate how stationary the mean of the host error rate s is over time, and describe the implications for error detection mechanisms given our findings.

We measured the stationarity of error rates by determining the change in mean error rates over 96 hours periods for each host. That is, for every 96 hours of wall-clock time during which the worker had been active, we determined the mean error rate on each host, and measured the change

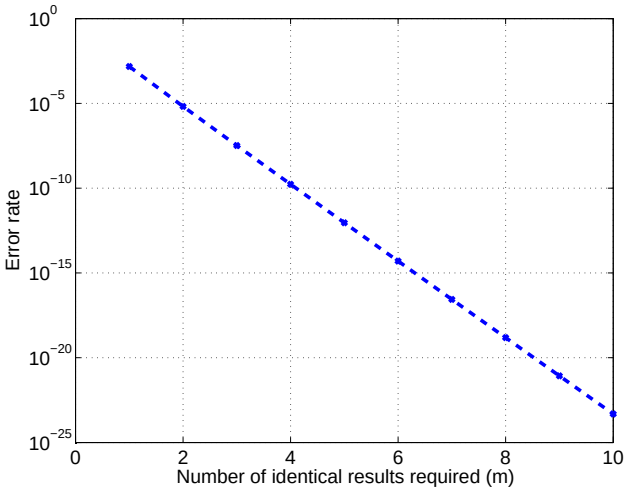


Figure 4. Error Rate With Majority Voting

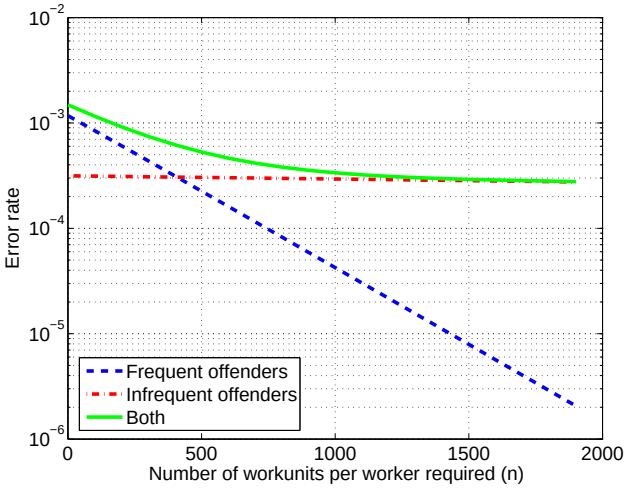


Figure 5. Error Rate With Spot-Checking and Blacklisting

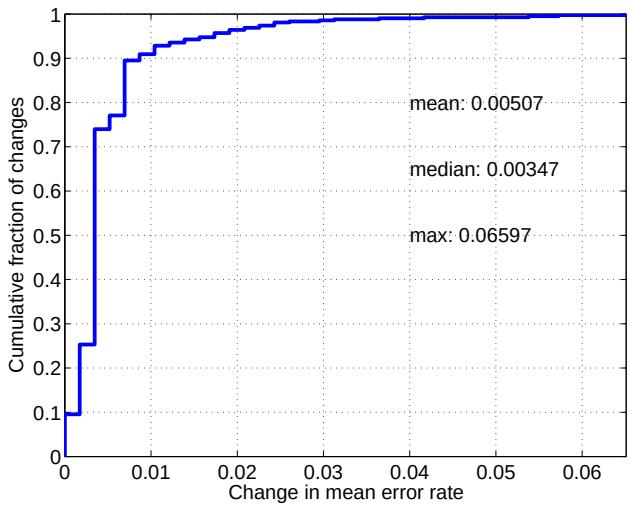


Figure 6. Error Rate Stationarity

	Statistic		
Host Group	μ	σ	σ/μ
All erroneous	0.0034	0.018	3.48
Top 10% erroneous	0.0335	0.030	0.89
Bottom 90% erroneous	0.001	0.002	2.01

Table 2. Statistics for Host Error Rates over 96 hour Periods.

in error rates from one period to the next. (We also tried 12, 24, 48 hour periods, but found similar results.) After close inspection of the results, we found that hosts often have long periods with no errors, and that when errors occurred, they occurred sporadically. Figure 6 shows the cumulative distribution function of error rate changes over all hosts. Because hosts often had relatively long periods without any errors, we excluded the data when the error rate for the current and previous interval was 0. Otherwise, including such data, would "skew" the distribution; that is, we would observe a CDF where most changes from one period to the next would be zero, but this would only because be errors occur infrequently and sporadically.

We found that only about 10% of the error rates were within 25% of the mean error rate of erroneous hosts. (We also graphed the CDF for the top 10% and bottom 90% of erroneous hosts, but found similar patterns.) Moreover, the mean change in error rate was 0.00507 (or about 0.77 of the mean error rate of erroneous hosts), and the median was 0.00347 (or about 0.533 of the mean error rate of erroneous hosts). This result shows that workunit errors are not very stationary, and in fact, the error rate fluctuates significantly over time.

We also computed statistics for *host* error rates over 96 hour periods. This characterizes s as defined in Section 3. Table 2 shows the mean, standard deviation, and coefficient of variation (which is the standard deviation divided by the mean) for all hosts, the top 10% of erroneous hosts, and the bottom 90% of erroneous hosts. We find that even for relatively long 96 hour periods, the host error rate is quite variable. In particular, the coefficients of variation for all hosts, the top 10%, and the bottom 90% are 3.48, 0.89, and 2.01 respectively.

To investigate the seasonality of errorless periods, we determined whether the set of hosts that err from time period to time period are usually the same hosts or different. In particular, we determined the erroneous host turnover rate as follows. For a specific time period, we determine which set of hosts erred, and then compared this set with the set of the hosts that erred in the following time period. The erroneous host turnover fraction is then the fraction of hosts in the first set that do not appear in the second set. We computed the erroneous host turnover fraction for time periods of 1 week, 2 weeks, and 4 weeks (see Figure 7). For example, the first segment at about 0.62 cor-

The reason that the mean for all hosts is different from the mean shown in Figure 2 is that here we calculate the mean using the mean of the individual hosts averages over time, whereas the mean in Figure 2 is calculated by dividing the total number of errors over all hosts by the total number of workunits completed. So the mean calculated here is taken over the mean of each host and takes into account variations over time.

responding to the 1 week period between April 27 and May 4 means that only 0.62 of the hosts that erred between April 20 and April 27 also erred between April 27 and May 4. (Note that the trace period began on April 20, 2006. Thus, the plots depicted in Figure 7 begin on April 27th, May 4th, and May 18th, respectively. Moreover, the trace period ended on July 20th, 2006. Thus, the plots end on July 20, and July 30th, as we only considered whole time periods for comparison.)

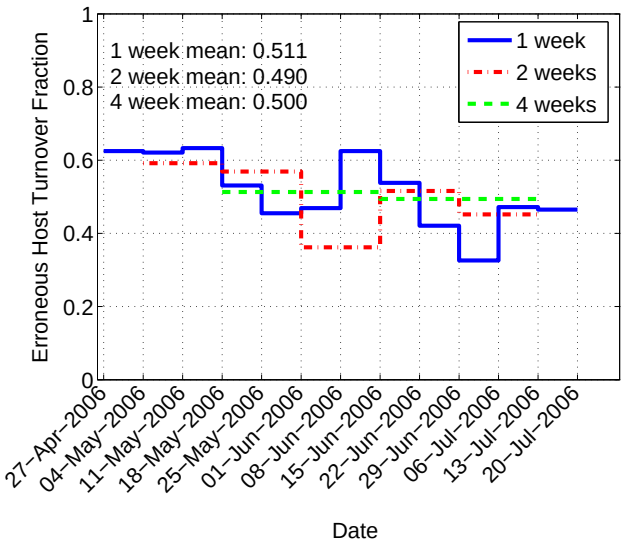


Figure 7. Turnover Rate of Erroneous Hosts

We find that for the 1 week and 2 week periods, the turnover rate fluctuates between ~ 0.60 and ~ 0.35. For the 4 week period, the turnover rate is about 0.50. On average, the turnover rate is about 0.50 for all periods, meaning that from time period to time period, 0.50 of the erred hosts will be newly erred hosts. That is, the 0.50 of erred hosts had *not* erred in the previous period. (We also determined the turnover rates for the frequent and infrequent offenders separately and the results are reported in the Appendix.)

One explanation for the lack of stationarity is that desktop grids exhibit much host churn, as users (and their hosts) often participate in a project for a while and then leave. In [6], the authors computed host lifetime by considering the time interval between entry and its last communication to the project. The host was considered "dead" if it had not communicated to the project for at least 1 month. They found that churn in Internet desktop grids was on average 91 days. Another explanation described in [28] is that a source of errors is overclocking of CPU's, and errors may be caused by non-stationary fluctuations in CPU temperature, as long running processes cause the CPU to overheat and emit calculation errors.

One implication is that mechanisms that depend on the consistency of error rates, such as spot-checking and credibility-based methods, may not be as effective as majority voting. Spot-checking depends partly on the consistency of error rates over time. Given the high variability in error rates and the intermittent periods without any errors, a host could pass a series of spot-checks, and thereafter or in between spot-checks, the host could produce a high rate of error. Conversely, an infrequent offender could have a burst of errors, be identified as an erroneous host via spot-checking, and then blacklisted. If this occurs with many infrequent offenders, this could potentially have a negative impact on throughput as shown in Figure 3.

The same is true for credibility-based systems. A host with variable error rates could build a high credibility, and then suddenly, cause high error rates. For example, suppose a host built a high credibility by returning errorless results for an entire 96 hour period (shown possible and likely by Figure 7). Then, the credibility-based system would conclude that any workunit sent to that host would be errorless. However, the the host after the 96 hour period could return erroneous results at a rate of 0.065 (as shown by Figure 6), which the credibility-based system would not detect, as it assumes consistency of host error rates (in this case 0). Thus, the estimated bounds resulting from spot-checking or credibility-based methods may not be accurate in real-world systems.

By contrast, majority voting is not as susceptible to fluctuations in error rates, as the error rate (and confidence bounds on the error rate) decrease exponentially with the number of votes. If $m = 2$, the expected error rate is 1.2×10^5 with a standard deviation of 8.2×10^7. Alternatively, if we assume the near-worst case scenario where hosts have a relatively high failure rate of 0.0214 ($= 0.0034 + 0.018$), we can still reliably achieve an error rate less than 1×10^{-5} by replicating each workunit $m = 4$ times, resulting in a redundancy of about 4. Nevertheless, the effectiveness of majority voting could be hampered by correlated errors, which we investigate in the next section.

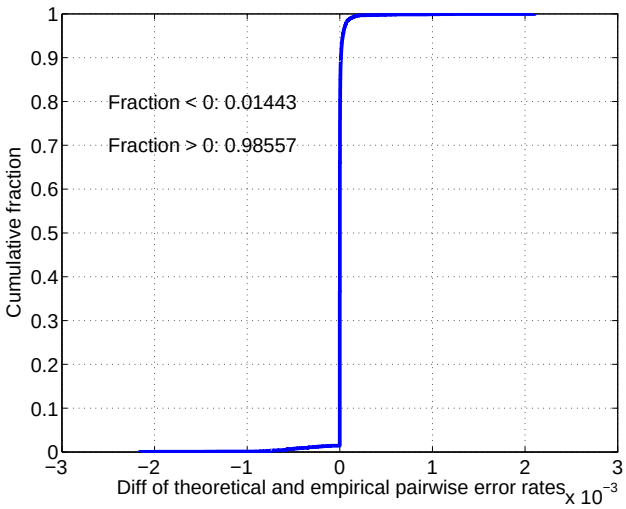


Figure 8. Pairwise Host Error Rates

5.3 Correlation of Error Rates

Using the trace of valid and erroneous workunit completion times, we computed the empirical probability that any two hosts had an error. That is, for each 10 minute period between April 20 to July 20, 2006, and for each pair of hosts, we counted the number of periods in which both hosts had an error, and the total number of periods in which both hosts computed a workunit. We then obtained the empirical probability that any two hosts would give an error simultaneously .

To compute the "theoretical" probability, we took the product of the individual host error rates, as described in Section 5.1. We then determined the difference between the theoretical and empirical probabilities for each host pairing. If the error rates for each pair of hosts are independent, then the theoretical probability should be equal to the empirical, and the difference should be 0.

Figure 8 shows the cumulative distribution for the differences between theoretical and empirical pairwise error rates. We find most (0.986) of the empirical pairwise error rates were greater than the theoretical. This suggests that the error rates between hosts are not correlated. Moreover, only 0.01443 of the pairings had differences less than 0. After carefully inspecting the number workunits computed by these host pairs, we believe these data points are in fact outliers due a few common errors made by both hosts over a relatively low number of workunits. That is, if we had more data, the number of instances where at least one of the two workunits is computed correctly would increase, and the empirical probability (and difference between the theoretical probability) would thereby decrease. We will continue to collect more data to give additional evidence for this conjecture (and make additions to the final paper as necessary, if it is accepted).

6 Comparing Intermediate Checkpoints for Long-Running Workunits

In this section, we present novel benefit analysis for a recently proposed mechanism for error detection [7]. This mechanism is based on checkpointing and replication, and is well-suited for long-running workunits. A number of projects such as `climateprediction.net`, `climatechange`, and `seasonalattribution` have workunits whose execution span months [10], and we believe early error detection for these projects would

be useful. The technique involves comparing intermediate checkpoint digests (provided for example by the MD5 [22] and SHA family [13] of algorithms) of redundant instances of the same task. (Note that often computations occupy a large space in memory often near the 100MB range [10] and/or sending a small, intermediate result for comparison may not be possible nor efficient.) If differences are found, the conclusion is that at least one task's execution is wrong. In contrast to the simple redundancy mechanism, where diverging computations can only be detected after a majority of tasks have completed, intermediate checkpoint comparison allows for earlier and more precise detection of errors, since execution divergence can be spotted at the next checkpoint following any error. This allows one to take proactive and corrective measures without having to wait for the completion of the tasks, and it allows for faster task completion, since faulty tasks can immediately be rescheduled.

We determine the benefit of using this technique by means of theoretical analysis and simulation results. In [7], we presented the theoretical analysis and simulation results of the same error detection mechanism, but there were two main limitations which we address here. First, the previous analysis was conducted using hypothetical error rates instead of error rates obtained empirically from a real project. In fact, our previous work assumed error rates that were orders of magnitude higher than the rates we determined in this study. Nevertheless, we show here that substantial benefits can still be achieved using this novel technique with real but relatively lower error rates. Second, the theoretical analysis previously conducted made the assumption that checkpoints occur simultaneously across hosts at constant intervals. For reasons that we discuss in the next paragraph, this is an unrealistic assumption in volatile, heterogeneous desktop grids. We loosen the assumption to consider variable checkpointing intervals, and give new theoretical upper and lower bounds on the benefits of this technique using a significantly different mathematical approach.

We assume that each task is checkpointed locally and periodically (as is done in several existing desktop grid systems [9, 17]). With respect to CPU time, the application could conduct local checkpointing periodically (for example, every 10 minutes). However, with respect to wall-clock time, the time between checkpoints is random because of non-deterministic events that could delay checkpointing such as a host being powered off, or the worker being suspended or killed because of user activity [15].

Thus, we model the time between checkpoints as a random variable. In particular, each checkpoint delineates the end of a task segment to create a total of c segments. Let R be the number of workers on which a checkpointed task is replicated (see summary in Table 1). Let $S_{k,g}$ be a random variable that represents the time to checkpoint the current segment g, beginning from the last checkpoint (or start of the task, in the case of the first checkpoint), on worker k where $1 \leq g \leq c$, and $1 \leq k \leq R$.

Let $T_{k,j}$ be a random variable that represents the amount of time elapsed since the start of the task up to the checkpoint time of segment j, on worker k. Specifically, $T_{k,j} = \sum_{g=1}^{j} S_{k,g}$ (see Figure 9 for an example).

We assume that $S_{k,g}$ is distributed exponentially with parameter λ across all workers. Previously, the authors of [16] also made the same assumption regarding the distribution of task (or equivalently segment) completion. While a number of previous studies have characterized the distribution of availability intervals on *enterprise* desktop resources [11, 20, 15], it is unclear how these periods of availability relate to the time of checkpointing a segment on *Internet* environments. Thus, for future work, we will verify our assumption using resource traces, for example, those currently being collected on Internet desktop environments [2].

Given that $S_{k,g}$ is distributed exponentially, $T_{k,j}$

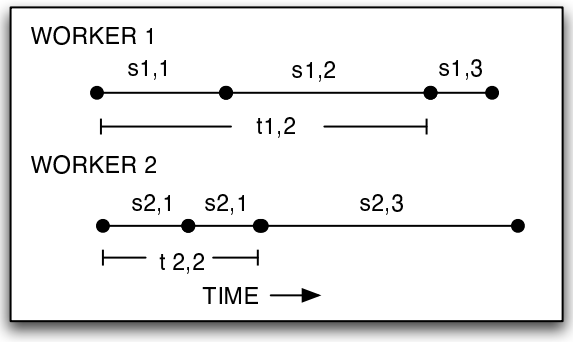


Figure 9. Example of Intermediate Checkpointing

has a gamma distribution with parameters $\alpha = j$ and $\beta = 1/\lambda$.

The time to validate the i^{th} segment is given by $T_{(R),i}$, which is the R^{th} order statistic of the set $T_{1,i}, ..., T_{R,i}$. That is, $T_{(R),i}$ represents the maximum time to complete segment i among all R workers.

The expected gain $E[W]$ for using intermediate checkpoints compared to state-of-the-art methods where the comparison is done at the end of the workunit is then given by:

$$E[W] = E[T_{(R),c} - T_{(R),i}] \tag{3}$$

where $1 \leq i \leq c$.

Let X be the number of trials, i.e., the segment in which an error occurs over all hosts, and let X have a truncated geometric distribution with parameters p and v, where p is the probability of getting an error within a segment over all hosts, and $v = 1 - p$. While we showed in Section 5.2 that error rates are not stationary, we believe our theoretical and simulation analysis gives a reasonable estimate of the long-term, expected benefit of our proposed method.

By the law of total expectation,

$$\begin{aligned} & E[T_{(R),c} - T_{(R),i}] \\ &= \sum_{i=1}^{c} \big(E[T_{(R),c} - T_{(R),i}|X = i] \times Pr(X = i)\big) \\ &= \sum_{i=1}^{c}((E[T_{(R),c}] - E[T_{(R),i}|X = i]) \\ &\qquad \times Pr(X = i)) \end{aligned} \tag{4}$$

From [8], a lower bound on the expectation of the maximum of a set of random variables is the maximum of the expected value of each random variable in the set. Moreover, Hartley and David [14] report that an upper bound for the expectation of the maximum is $\mu + \sigma \times (n - 1)/\sqrt{2n - 1}$, given a set of n independent random variables with identical means and variances (μ, σ^2).

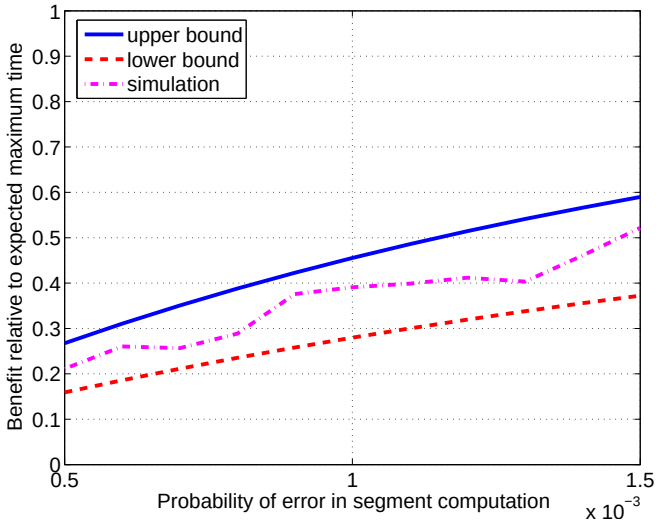
Substituting in Equation 3, using those bounds and Equation 4, we have the following:

$$\begin{aligned} & E[W] = E[T_{(R),c} - T_{(R),i}] \\ &= \sum_{i=1}^{c}((E[T_{(R),c}] - E[T_{(R),i}|X = i]) \times Pr(X = i)) \\ &\geq \sum_{i=1}^{c}((\mu \times i - (\mu + \sigma \times (R - 1)/\sqrt{2R - 1}]) \\ &\qquad \times Pr(X = i)) \end{aligned} \tag{5}$$

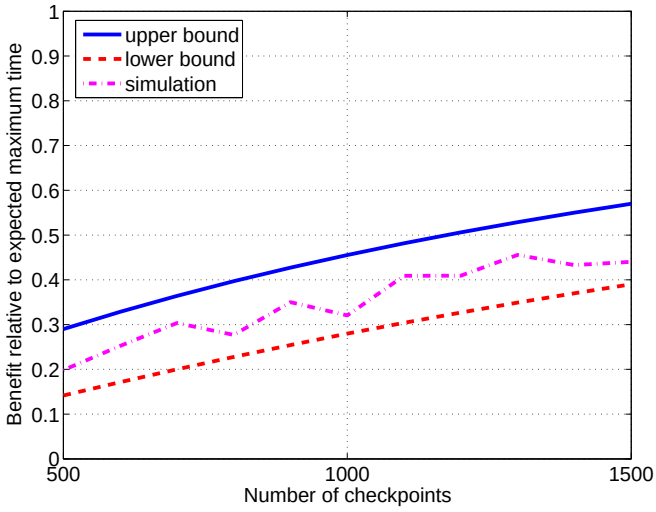
where $\mu = 1/\lambda$, $\sigma = \sqrt{i \times (1/\lambda)^2}$, $Pr(X = i) = pv^{i-1}$.

In Figure 10, we give upper and lower bounds on the benefit $E[W]$ relative to the upper and lower bounds of the expected maximum time $E[T_{(R),c}]$ for checkpointing at the end of the task. In particular, in Figure 10(a), the number of checkpoints c is fixed to 1000, and p varies between [0.0005, 0.0015]. In Figure 10(b), the probability of error within each segment p is fixed at 0.001, and c varies between [500, 1000].

We observe potentially significant gains even for small error rates. For example, in Fig-



(a) Varying probability of error



(b) Varying checkpoint frequency

Figure 10. Benefits of intermediate checkpointing

ure 10(a), we find that if the probability of error p is 0.001 and the number of checkpoints per task c is 1000, then the potential benefit of intermediate checkpointing is between $\sim 30 - 45\%$. While 1000 checkpoints may seem abnormally large, if we assume a task checkpoints every 10 minutes a thousand times, this equates to a 7-day workunit. (This is a reasonable checkpoint frequency and workunit length as the frequency in real projects EINSTEIN@home, PREDICTOR@Home, and SIMAP is on the order of minutes [3, 4, 21] and execution is on the order of days or months [10].) In Figure 10(b), we find that if the number of checkpoints is 1050 (and probability of error is 0.001), then the potential benefit of intermediate checkpointing is between $\sim 30 - 45\%$.

We then confirmed and extended the theoretical results through simulation. We assign a number of tasks to a set of workers. Whenever a worker computes a checkpoint, it randomly determines whether that computation is wrong or correct. Once a checkpoint is wrong, all the remaining checkpoints from that worker are also considered as wrong.) In our experiments, the time that a worker needed to compute a checkpoint was given by an exponential distribution. We chose an arbitrary average checkpoint time (as it does not impact the *relative* benefit of our technique). We varied the number of checkpoints of each task and the probability of error in each checkpoint. In Figures 10(a) and 10(b), we show the results of our experiments for the same range of paramters as used for the theoretical analysis. The curve of the observed benefit is the average of 300 trials.

Our results show that the there is a considerable benefit in comparing intermediate checkpoints, especially for long-running workunits. Even for very small probabilities of error, which correspond to real values observed in real systems, the time savings can amount to 20%-45% of the time corresponding to state-of-the-art solutions.

One potential limitation of this method is scalability of receiving the high-frequency digest mes-

We used a constant value for the probability of error. We also tried random variables (truncated gaussian, exponential and others), with little if any impact on the outcome of the trials.

sages if digests are sent centrally to a "supervisor" for comparison. We are currently working on secure load-balancing techniques via distributed hash tables (DHT) [27] to remove this limitation, and we will report on this in future work.

7 Summary and Future Work

We characterized quantitatively the error rates in a real Internet desktop grid system with respect to the distribution of errors among hosts, the stationarity of error rates over time, and correlation among hosts. In summary, the characterization findings were as follows:

1. *A significant fraction of hosts (about 35%) will commit at least a single error over time.*

2. *The mean error rate over all hosts (0.0022) and over only erroneous hosts (0.0065) is quite low.*

3. *A large fraction of errors result from a small fraction of hosts.* For example, about 70% of error are caused by only 10% of the hosts.

4. *Error rates over time vary greatly and do not seem stationary.* Error rates can vary as much as 3.48 over time. The turnover rate for erroneous hosts can be as high as 50%.

5. *Error rates between two hosts is often not correlated.* While correlation errors can occur during an coordinated attack, we do not believe it commonly occurs in practice.

In light of these characterization findings, we showed the effectiveness of several error prevention and detection mechanisms (namely, blacklisting, majority voting, spot-checking, and credibility-based methods) and concluded the following (in parenthesis are the numbers corresponding to the characterization finding above, from which the conclusion was drawn):

1. *If one can afford redundancy or one needs an error rate to be less then* 1×10^3*, then majority voting should be used.* Majority voting will reduce errors exponentially. For $m = 2$, the expected error rate would be about 1×10^{-6} (2, 5)

2. *If one can afford an error rate greater than* 1×10^3 *and can make batches relatively long (ideally with at least 1000 work units and at least 1 week of CPU time per worker), then one should use spot-checking with blacklisting.* To minimize the affects of non-stationary error rates such false positives and false negatives, one should use spot-checking for as long as a period as possible on as many workunits as possible. Blacklisting should be used because it is an effective way of removing frequent offenders. (3, 4)

3. *Fluctuations in error rates over time may limit the effectiveness of credibility-based systems.* For example, a worker could build up good credibility (either intentionally or simply because error rates appear to be non-stationary), and then once it is assigned work, perform frequent errors. By contrast, majority voting is less susceptible as error rates and also the confidence bounds on error rates decrease exponentially with the number of votes. (4)

4. *If one has a long-running application (> 1 week), then one should consider using the digest of intermediate checkpoints to improve and accelerate error detection.* We presented novel analysis for a recently proposed mechanism for error detection, which applies majority voting for comparing the digest of intermediate checkpoints. We show both theoretically and in simulation significant gains (as high as 45%) compared to the state-of-the-art replication mechanisms.

For current and future work, we will develop techniques to focus on the cause of errors.

Also, we will develop secure and scalable load-balancing techniques for comparing intermediate checkpoint digests.

References

[1] EINSTEN@home. http://einstein.phys.uwm.edu.

[2] XtremLab. http://xtremlab.lri.fr.

[3] B. Allen. Personal communication, April 2006.

[4] C. An. Personal communication, April 2006.

[5] D. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, 2004.

[6] D. Anderson and G. Fedak. The Computational and Storage Potential of Volunteer Computing. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, 2006.

[7] F. Araujo, P. Domingues, and L. M. Kondo D. Silva. Validating Desktop Grid Results By Comparing Intermediate Checkpoints. *Submitted to 2nd Coregrid Integration Workshop*, 2006.

[8] T. Aven. Upper (lower) bounds on the mean of the maximum (minimum) of a number of random variables. *Journal of Applied Probability*, 22:723–728, 1985.

[9] The berkeley open infrastructure for network computing. http://boinc.berkeley.edu/.

[10] Catalog of boinc projects. http://boinc-wiki.ath.cx/index.php?title=Catalog_of_BOINC_Powered_Proje%cts.

[11] J. Brevik, D. Nurmi, and R. Wolski. Quantifying Machine Availability in Networked and Desktop Grid Systems. Technical Report CS2003-37, Dept. of Computer Science and Engineering, University of California at Santa Barbara, November 2003.

[12] C. Christensen, T. Aina, and D. Stainforth. The challenge of volunteer computing with lengthy climate model simulations. In *1st IEEE International Conference on e-Science and Grid Computing*, Melbourne, Australia, 2005.

[13] D. Eastlake and P. Jones. RFC 3174: US Secure Hash Algorithm 1 (SHA1). *Request for Comments, September*, 2001.

[14] H. Hartely and H. David. Universal bounds for mean range and extreme observations. *The Annals of Mathematical Statistics*, 25:85–89, 1954.

[15] D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. Chien. Characterizing and Evaluating Desktop Grids: An Empirical Study. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'04)*, April 2004.

[16] Y. Li and M. Mascagni. Improving performance via computational replication on a large-scale computational grid. In *Proc. of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, May 2003.

[17] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS)*, 1988.

[18] P. Malecot, D. Kondo, and G. Fedak. Xtremlab: A system for characterizing internet desktop grids (abstract). In *in Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, 2006.

[19] D. Molnar. The SETI@home Problem. *ACM Crossroads Student Magazine*, september 2000.

[20] M. Mutka and M. Livny. The available capacity of a privately owned workstation environment . *Performance Evaluation*, 4(12), July 1991.

[21] T. Rattel. Personal communication, April 2006.

[22] R. Rivest. RFC-1321 The MD5 Message-Digest Algorithm. *Network Working Group, IETF*, April 1992.

[23] L. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. In *Proceedings of IEEE International Symposium on Cluster Computing and the Grid*, May. 2001.

[24] L. Sarmenta. *Volunteer Computing*. PhD thesis, MIT, March 2001.

[25] L. Sarmenta and S. Hirano. Bayanihan: Building and Studying Web-Based Volunteer Computing Systems Using Java. *Future Generation Computer Systems*, 15(5-6):675–686, 1999.

[26] M. Shirts and V. Pande. Screen Savers of the World, Unite! *Science*, 290:1903–1904, 2000.

[27] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[28] M. Taufer, D. Anderson, P. Cicotti, and C. L. B. III. Homogeneous redundancy: a technique to ensure integrity of molecular simulation results using public computing. In *Proceedings of the International Heterogeneity in Computing Workshop*, 2005.

[29] S. Zhao and V. Lo. Result Verification and Trust-based Scheduling in Open Peer-to-Peer Cycle Sharing Systems. In *Proceedings of IEEE Fifth International Conference on Peer-to-Peer Systems*, May 2001.

Appendix

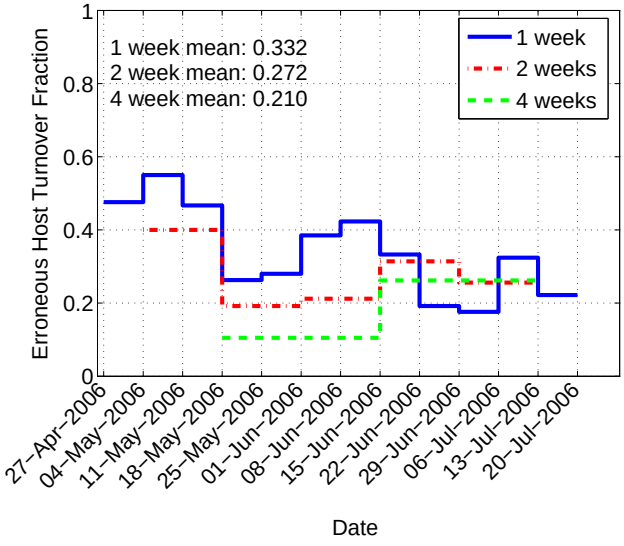


Figure 11. Turnover Rate of Top .10 of Erroneous Hosts

We also computed the turnover rate for the top 0.10 of erred hosts (see Figure 11) and the bottom 0.90 of erred hosts (see Figure 12). For the top 0.10 of erred hosts, we find that even among the hosts that err the most often, the turnover rate can

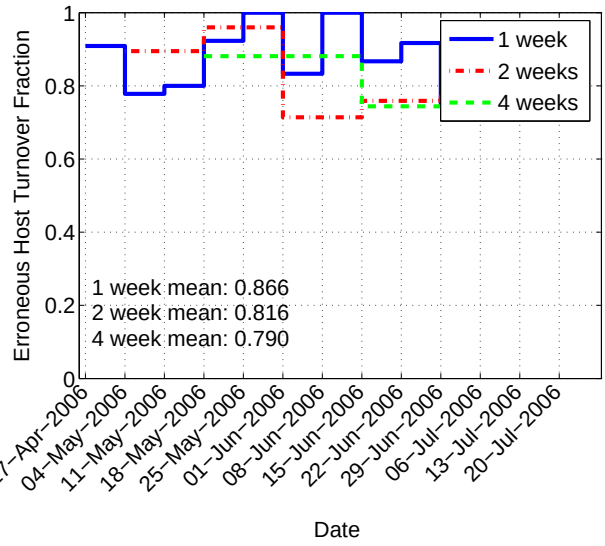


Figure 12. Turnover Rate of Bottom .90 of Erroneous Hosts

be substantial, i.e., usually greater than 0.20. This means that 0.20 of the hosts had not erred in the previous period of time. Nevertheless, on average this turnover rate is lower than the rates that consider all hosts in the platform. For the bottom 0.90 of erred hosts, we find that the mean turnover rate is usually greater than 0.79, which is on average significantly higher than the rates that consider all hosts in the platform.