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], and Franck Cappello[1]

[1] INRIA Futurs, France
[2] University of Coimbra, Portugal
[3] Polytechnic Institute of Leiria, Portugal

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 and empirical characterization of errors stemming from input or output (I/O) failures. 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.

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, computational biology, 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 the validation of results computed by volatile and possibly malicious hosts. In large and complex distributed systems, errors in results are inevitable, and errors can stem from different sources. Some sources can be computational. For example, an error could result from a CPU miscalculation due to overclocking and overheating [2]. Other sources can be related to failures during application input or output (I/O). For example, if a machine crashes when the application is writing to an output file or checkpoint, only a partial number in-memory data blocks could have been flushed to disk (not necessarily in order) [3], which would lead to an erroneous result[4]. Thus, effective error detection mechanisms are essential, and several methods have been proposed previously [7,8].

However, little is known about the nature of errors in real systems. Yet, the trade-offs and efficacy among different error detection mechanisms are dependent on how errors occur in real systems. Thus, we focus on characterizing errors, specifically I/O errors, in a real system by addressing the following critical questions:

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

To help answer those questions, we deployed an Internet desktop grid application across several thousand desktop hosts. We then validated the results returned by hosts, and we analyzed the invalid results to characterize quantitatively the I/O error rates in a real desktop grid project.

[4] While a number of mechanisms exist to ensure atomic writes or to detect file corruption, in practice, few if any are provided by desktop grid systems [4,5,6] or utilized by desktop grid applications themselves.

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 4, we define our error measurement method. In Section 5, we study the frequency, stationarity, and correlation of host errors. Finally, in Section 6, we conclude with a summary of our specific contributions and future work.

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 returned 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 [7] 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 [2] and physics [1]) require (low) bounds on error rates as the correctness of the computed results is essential for making accurate scientific conclusions.

3 Related Work

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 [9] 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.

Several mechanisms for reducing errors in desktop grids have been proposed. We discuss three of the most common state-of-the-art methods [7,8,2] namely spot-checking, majority voting, and credibility-based techniques, and emphasize the issues related to each method.

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 [7], 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 (for a summary of parameter definitions see Table 1).

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 an incorrect result being accepted after a majority vote is given by:

$$\varepsilon_{majv}(\varphi, m) = \sum_{j=m}^{2m-1} \binom{2m-1}{j} \varphi^j (1-\varphi)^{2m-1-j} \tag{1}$$

The redundancy of majority voting is $\frac{m}{1-\phi}$.

The main issues for majority voting are the following. First, the error bound assumes that error rates are not correlated among hosts. Second, majority voting is most effective when error rates are relatively low ($\leq 1\%$); otherwise the required redundancy could be too high.

A more efficient 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 returned from the erroneous host are discarded (perhaps unknowingly to the host).

Parameter	Definition
f	Fraction of hosts that commit at least one error
s	Error rate per erroneous host
φ	Probability that a worker (from the set of erroneous and non-erroneous hosts) 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	Number of workunits to be computed by each worker should include N/P workunits + work due to workers being blacklisted + work due to redundancy 1/1-q
ϑ	Threshold used by a credibility-based system
W	Benefit in time of intermediate checkpointing relative to state-of-the-art methods
$T_{k,j}$	Time from start of workunit to the time of checkpointing segment j on worker k
R	Number of workers on which a checkpointed task is replicated
c	Number of segments or equivalently checkpoints per task
$S_{k,g}$	Time from start of segment g to the time of checkpointing segment g on worker k
p, v	p is the probability of getting an error within a segment on any host. $v = 1 - p$
X	Random variable distributed geometrically with parameters p and v representing the number of task segments before an error occurs

Table 1. Parameter Definitions.

Erroneous workunit computation was modelled as a Bernoulli process [7] 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.

Allowing the model to exclude coordinated attacks, let q be the frequency of spot-checking, let n be the amount of work contributed by the erroneous worker, let f be the fraction of hosts that commit at least 1 error, and let s be the error rate per erroneous host. $(1 - qs)^n$ 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:

$$\varepsilon_{scbl}(q, n, f, s) = \frac{sf(1 - qs)^n}{(1 - f) + f(1 - qs)^n} \tag{2}$$

The amount of redundancy of spot-checking is given by $\frac{1}{1-q}$.

There are several critical issues related to spot-checking with blacklisting. First, it assumes that blacklisting will effectively remove erroneous hosts, in spite of the possibility of hosts registering with new identities or high host churn as shown by [10]. Without blacklisting, the upper bound on the error rate is much higher and does not decrease inversely with n. Second, spot-checking is effective only if error rates are consistent over time. Third, spot-checking is most effective when error rates are high ($> 1\%$); otherwise, the number of workunits to be computed per worker n must be extremely high.

To address the potential weaknesses of majority voting and spot-checking, **credibility-based systems** were proposed [7], which use the conditional probabilities of errors given the history of host result correctness. Due to space limitations, we only describe the method at a high-level. 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 each host's past performance to determine workunit assignment.

Note that the methods described in this section were designed to detect errors from any source, including a computational source (for example, CPU miscalculations) or an I/O source (for example, a crash during

a checkpoint). In the sections that follow, we determine the methods' efficacy in detecting errors caused by I/O failures.

4 Method

We studied the error rates of a real Internet desktop grid project called XtremLab [11]. XtremLab uses the BOINC infrastructure [4] 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[5].

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. After careful inspection of output files that failed synatatic or semantic checks, we found that most of these output files were truncated prematurely or had scrambled output (perhaps due to an out-of-order flush of in-memory blocks [3], for example). As such, we assume that any output file that fails a syntactic or semantic check would have corresponded to an I/O error of an application (for example, an erroneous checkpoint) that would lead to an erroneous result [6]. 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 I/O error rates for a real Internet desktop grid project.

5 Error Characterization

5.1 Frequency of Errors

The effectiveness of different methods by which errors are detected is heavily dependent on the frequency of errors among hosts. We measured the fraction of workunits with errors per host, and show the cumulative distribution function (CDF) of these fractions in Figure 1. For example, the point (1×10^{-3}, 0.78) shows that 0.78 of the hosts had error rates less than or equal to 1×10^{-3}. The mean error rate was 0.002, and the max error rate was 0.098.

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 hosts 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 $\varphi = 0.002$ as shown in Figure 1, we can simply

[5] Note that when we increased the threshold to 2, 3, or 4 weeks, the conclusions of our analysis did not change.

[6] 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 [4]. 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.

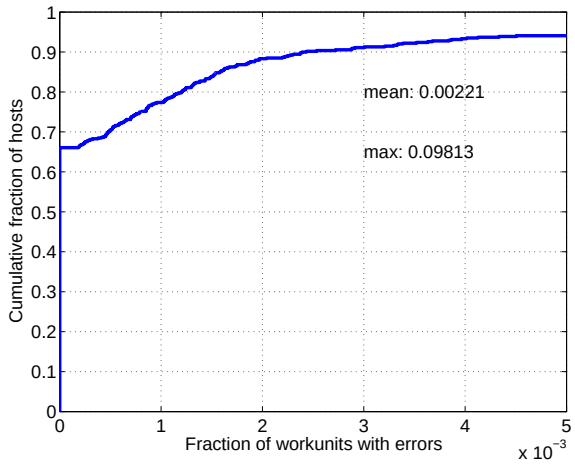


Fig. 1. Error Rates of of Hosts in Entire Platform

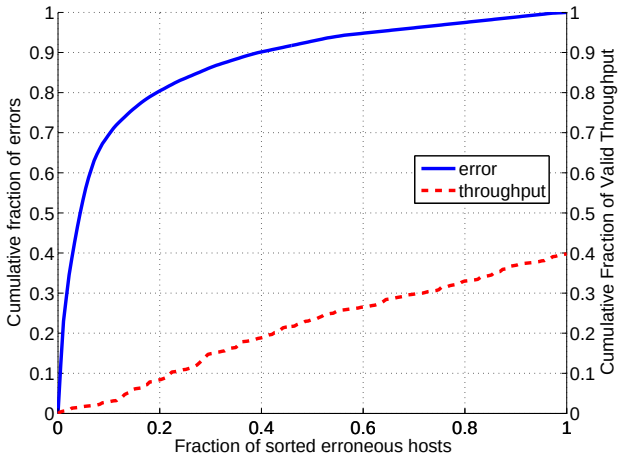


Fig. 2. Cumulative Error Rates and Effect on Throughput

conduct a majority vote where $m = 2$ to achieve an error rate of about 1×10^{-5} and with a redundancy of about 2.0.

While spot-checking can achieve a similar error rate of about 1×10^{-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 $q = 0.10$, $f = 0.35$ (from Figure 1), $s = 0.003$ (as shown in Table 2), Equation 2 requires that the number of workunits (n) processed by each worker be *greater* than 5300. 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 in [1]), 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).

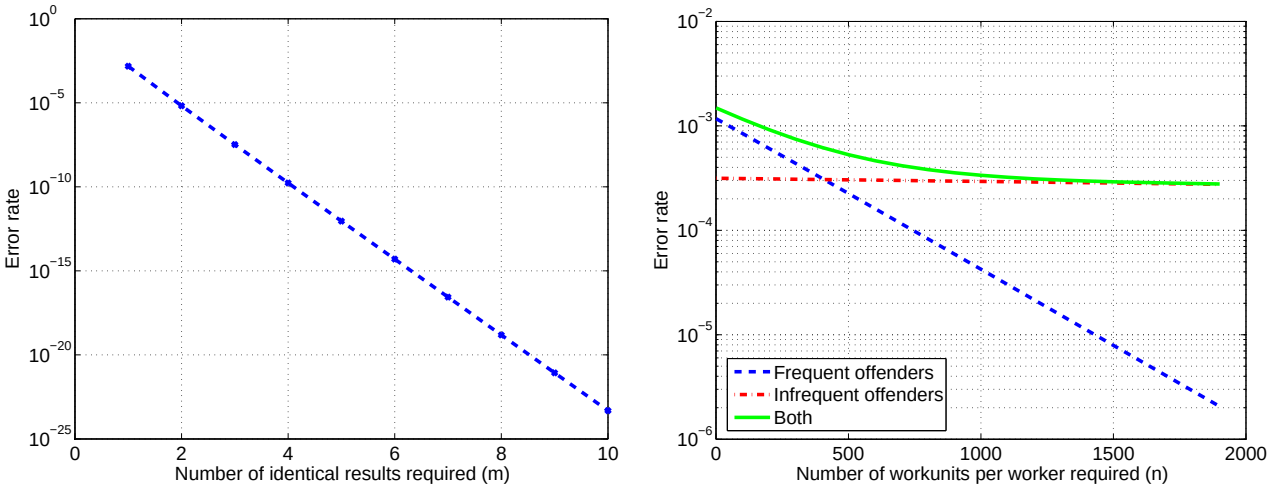
We also focused on characterizing the hosts that returned at least one corrupt result, and found that the mean error rate over this set of hosts is still relatively low at 0.0065.

Figure 2 shows the skew of the frequency 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 2, shows the cumulative fraction of errors. For example, the point (0.10, 0.70) shows that the 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 2 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 2, but a similar problem described earlier occurs. The error rate for majority voting ε_{majv} is given by Equation 1, where $\varphi = f_{all} \times s_{frequent} \times f_{frequent} + f_{all} \times s_{infrequent} \times f_{infrequent}$. Note that 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.



(a) With Majority Voting

(b) With Spot-Checking and Blacklisting

Fig. 3. Error Rate Bounds

We plot ε_{majv} as a function of m in Figure 3(a). 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 $\varepsilon_{scbk,infrequent}$. Note that $\varepsilon_{scbk,infrequent}$ 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 3(b) $\varepsilon_{scbk,frequent}$, $\varepsilon_{scbk,infrequent}$, 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 errors 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.

Thus, 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. If all frequent offenders are detected by spot-checking and blacklisted, then by Figure 2, this will reduce error rates by 0.70 (or equivalently, an error rate of 63×10^{-5}) and cause only a 0.05 reduction in throughput due to blacklisting. However, to reduce the error rate down to 1×10^{-5}, spot-checking must detect errors from both frequent *and* infrequent offenders. As shown by Figure 3(b), spot-checking will not efficiently detect errors from infrequent offenders because it requires a huge number workunits to be processed by each worker. From Figure 3(b), we conclude that spot-checking can reduce error rates down to about 2×10^{-4} quickly and efficiently. To achieve lower error rates, one should consider using 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.

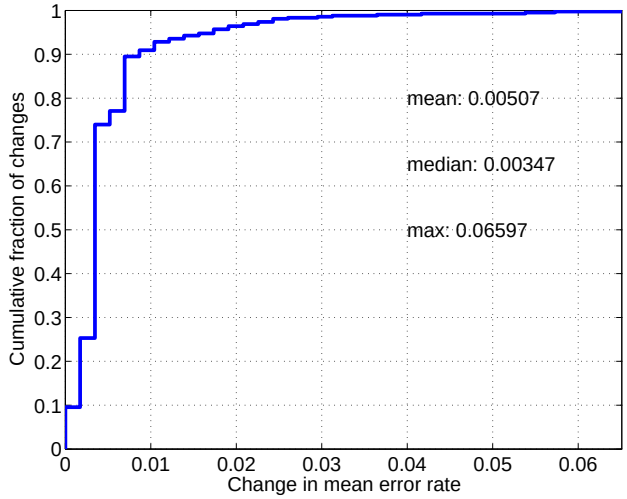


Fig. 4. Error Rate Stationarity

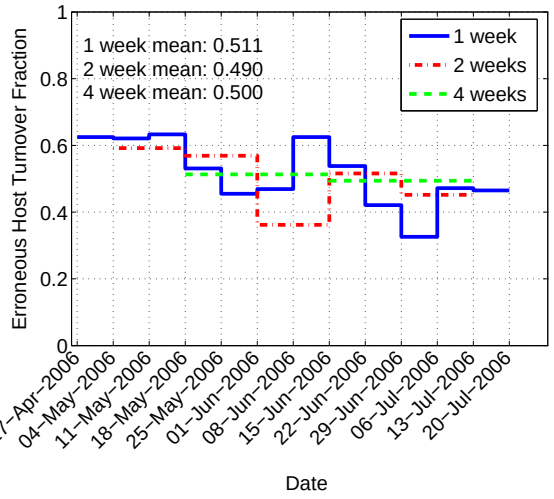


Fig. 5. Turnover Rate of Erroneous Hosts

We measured the stationarity of error rates by determining the change in mean error rates over 96 hour 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 in error rates from one period to the next[7]. 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 4 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 the 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[8]. 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.

	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.

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

[7] We also tried 12, 24, 48 hour periods, but found similar results.

[8] We also graphed the CDF for the top 10% and bottom 90% of erroneous hosts, but found similar patterns.

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 5). For example, the first segment at about 0.62 corresponding 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[9].

We find that for the 1 week and 2 week periods, the turnover rate fluctuates between ~ 0.35 and ~ 0.60. 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.

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 [10], 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 a host lifetime in Internet desktop grids was on average 91 days.

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 2.

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 5). 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 4), 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 about 1×10^{-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.

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 at the same time. 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 computed an erroneous workunit, and the total number of periods in which both hosts computed a workunit (erroneous or correct). Using those counts, we then determined the empirical probability that any two hosts would give an error simultaneously, i.e., within the same 10 minute period. In this way, we determined the the empirical joint probability of two hosts having an error simultaneously.

[9] Note that the trace period began on April 20, 2006. Thus, the plots depicted in Figure 5 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.

We then determined the "theoretical" probability of two hosts having an error simultaneously by taking the product of their individual host error rates. The individual host error rates are given by dividing the number of erroneous workunits per host by the total number of workunits computed per host (as described in Section 5.1). After determining the "theoretical" probabilities for each host pair, we then determined the difference between the theoretical and empirical probabilities for each host pair. If the error rates for each pair of hosts are not positively correlated, then the theoretical probability should be greater than or equal to the empirical, and the difference should be nonnegative.

Figure 6 shows the cumulative distribution for the differences between theoretical and empirical pairwise error rates. We find most (0.986) of the theoretical pairwise error rates were greater than the empirical. This suggests that the error rates between hosts are not positively correlated. Moreover, only 0.01443 of the pairings had differences less than 0. After carefully inspecting the number of 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.

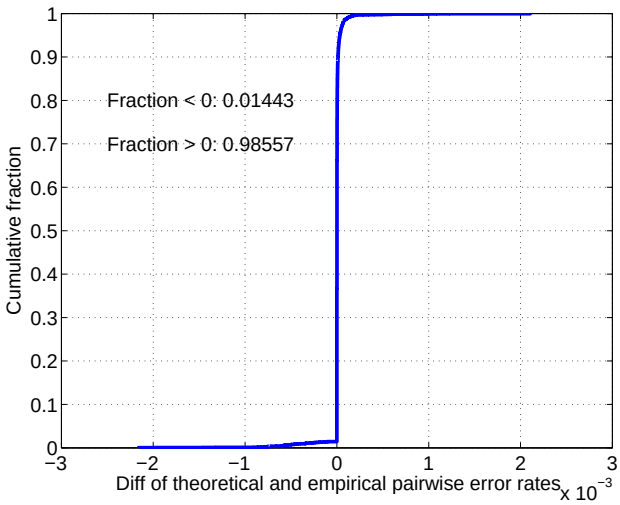


Fig. 6. Pairwise Host Error Rates

6 Summary

We characterized quantitatively the I/O 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 often seem uncorrelated.* While correlated errors could occur during a coordinated attack or after worm propagation, we do not believe it is the most common source of errors 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. We concluded the following (in parenthesis are the point numbers in the characterization listing above from which the conclusion was drawn):

1. *If one can afford redundancy or one needs an error rate to be less then* 2×10^{-4}*, then majority voting should be strongly considered.* Majority voting will reduce errors exponentially. For $m = 2$, the expected error rate would be about 1×10^{-5} (2, 5)
2. *If one can afford an error rate greater than* 2×10^{-4} *and can make batches relatively long (ideally with at least 1000 work units and at least 1 week of CPU time per worker), then spot-checking with blacklisting should be strongly considered.* To minimize the effects 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)

For future work, we plan to address several limitations of this study. First, our method measures the I/O 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 I/O or computation patterns could potentially differ in error rates. Thus, to study I/O and computational errors of another application, we will deploy a real scientific application that has easily verifiable results (for example, an application that solves the satisfiability problem). Nonetheless, we believe this is the first study of a real project to give quantitative estimates of I/O error rates.

Second, the error detection methods that we evaluated were designed to detect errors of all types, including errors stemming from both computational miscalculations and I/O failures. However, we believe from preliminary results, that computational errors (for example, those resulting from CPU miscalculations) are relatively rare in real systems compared to I/O errors. Thus, for future work, we will confirm this hypothesis, and we will implement efficient mechanisms specifically for preventing I/O errors in desktop grid applications, such as computing and storing the message digest of a checkpoint, or using temporary files to ensure atomic writes.

References

1. : Catalog of boinc projects. `http://boinc-wiki.ath.cx/index.php?title=Catalog_of_BOINC_Powered_Proje%cts`
2. Taufer, M., Anderson, D., Cicotti, P., III, C.L.B.: Homogeneous redundancy: a technique to ensure integrity of molecular simulation results using public computing. In: Proceedings of the International Heterogeneity in Computing Workshop. (2005)
3. Oltean, A.: How to do atomic writes in a file. `http://blogs.msdn.com/adioltean/archive/2005/12/28/507866.aspx` (December 2005)
4. Anderson, D.: Boinc: A system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA (2004)
5. Fedak, G., Germain, C., N'eri, V., Cappello, F.: XtremWeb: A Generic Global Computing System. In: Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'01). (May 2001)
6. Chien, A., Calder, B., Elbert, S., Bhatia, K.: Entropia: Architecture and Performance of an Enterprise Desktop Grid System. Journal of Parallel and Distributed Computing **63** (2003) 597–610
7. Sarmenta, L.: Sabotage-tolerance mechanisms for volunteer computing systems. In: Proceedings of IEEE International Symposium on Cluster Computing and the Grid. (May. 2001)
8. Zhao, S., Lo, V.: 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)
9. Kondo, D., Taufer, M., Brooks, C., Casanova, H., Chien, A.: Characterizing and Evaluating Desktop Grids: An Empirical Study. In: Proceedings of the IPDPS'04. (April 2004)
10. Anderson, D., Fedak, G.: The Computational and Storage Potential of Volunteer Computing. In: Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06). (2006)
11. Malecot, P., Kondo, D., Fedak, G.: Xtremlab: A system for characterizing internet desktop grids (abstract). In: in Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing. (2006)