CITI Technical Report 07-3

Performance and Availability Tradeoffs in Replicated File Systems

Jiaying Zhang
jiayingz@umich.edu

Peter Honeyman
honey@citi.umich.edu

ABSTRACT

Replication is a key technique for improving fault tolerance. Replication can also improve application performance under some circumstances, but can have the opposite effect under others. In this paper we focus on a class of Grid applications—long-running, compute-intensive, and write-mostly—and develop a calculus that takes into consideration the I/O characteristics of applications and failure behavior of distributed storage nodes to prescribe a file system replication strategy that maximizes the utilization of computational resources.

October 8, 2007

Center for Information Technology Integration
University of Michigan
535 W. William St., Suite 3100
Ann Arbor, MI 48103-4978

Performance and Availability Tradeoffs in Replicated File Systems

Jiaying Zhang
jiayingz@umich.edu

Peter Honeyman
honey@citi.umich.edu

1. Introduction

The rapid growth of network bandwidth and computing power has made Grid computing a practical solution for problems that require massive computing. Unlike traditional clustered parallel systems, Grid computing is characterized by geographically distributed institutions sharing computing, storage, and instruments in dynamic virtual organizations [1, 2]. Access to Grid resources in large-scale heterogeneous environments such as these often come with twin penalties of large network latencies and frequent component failures, posing a significant challenge to running applications on the Grid.

Replication is a key technique for improving performance and fault tolerance in distributed systems. Failure can be hidden by making identical services available from replication servers. In the same way, replication can overcome latency penalties by offering nearby copies to services distributed over a wide area and address performance scaling requirements by tailoring the number of copies according to demand.

To facilitate sharing of resources on Grid, we developed a mutable replicated file system that provides users and applications efficient and reliable data access with conventional file system semantics [3]. With data replication, a fundamental challenge is to maintain consistent replicas without introducing high performance overhead. Preserving consistency is essential to guaranteeing correct behavior during concurrent writes. Consistency is also needed to guarantee durability of data modifications in the face of failure. By exploiting locality of reference in application updates, our earlier study shows that when concurrent writes occur at a moderate rate, we are able to maintain consistency with negligible overhead. However, durability guarantees can impose a considerable penalty on performance and require more careful examination. To explore the tradeoff between performance and failure resilience, this paper proposes an evaluation model that estimates the expected running time of an application given specified replication policy and application characteristics.

We focus on a specific class of Grid applications: those whose output can be reproduced by restarting or rolling back to a saved checkpoint, a strategy characteristic of long-running applications executing on clusters. In a replicated file system, updates are distributed to multiple file servers. In the ideal, if one or more file servers fail, the system can fully recover as long as one replication server holding the fresh data is accessible. Applications connected to a failed file server can continue their executions straightaway by diverting their requests to the available serves. However, if no surviving server holds a fresh copy of data, the system cannot hide the failure from applications. In that case, the applications need to roll back to a saved checkpoint or restart their executions after switching to a working server.

Accordingly, the durability guarantee that a storage system provides determines the expected cost to recover a failure that might occur during the execution of the program. Introducing replication into the file system improves durability and reduces the risk of losing the results of long-running applications if failure happens. On the other hand, the strength of the durability guarantee is determined by (1) the number of synchronous data copies maintained on different replication servers, and (2) the incidence of correlated failure among these servers. Guaranteeing high data durability requires the system to maintain up-to-date data copies on a number of replicas that seldom fail at the same time. When applications consist of a large amount of updates, this requirement can lead to expensive performance cost. In some cases, it is more efficient to trade durability for performance and let applications regenerate their execution results when the system cannot mask a failure.

In the remainder of this paper, we identify the factors that affect the performance of a Grid application over a replicated file system and present an evaluation model for estimating the expected running time of an application under various replication strategies. The main contribution of our study is a calculus that determines an optimal replication strategy for a Grid application based on the I/O characteristics of the application, the latency of the replication servers, the expected frequency of storage site failure, and the degree of correlated failure among replication servers.

The rest of the paper is organized as follows. In Section 2, we give a brief description of a mutable replicated file system that we developed for Grid applications. Section 3 develops a failure model for distributed resources using PlanetLab trace data. Section 4 introduces a Markov model to evaluate the performance of a Grid application over a replicated file system in the presence of failures. In Section 5, we combine the failure and performance models to predict the performance of applications with different running time and write characteristics. Section 6 reviews related work and Section 7 concludes.

2. Performance and Reliability Tradeoffs

In earlier work [3], we developed a mutable replicated file system to facilitate Grid computing over wide area networks that provides users high performance data access with standard file system semantics. In this section, we briefly describe that replicated file system.

Our mutable replicated file system is built as an extension to the NFS version 4 protocol [37], the Internet standard for distributed filing. As the protocol specifies, the first time a client accesses a replicated file system, it receives a list of replication server locations and chooses a nearby one. To support mutable replication, we use a variant of the well understood and intuitive *primary-copy scheme* to coordinate concurrent writes. Before a client can write a file or modify a directory, one of the replication servers must be designated as the primary server for the file or the directory to be modified. If there is none, the replication server that the client connects to is elected as the primary server. To guarantee synchronized data access, all of the other replication servers then forward client read and write requests for that file or directory to the primary server. When the client updates are complete and all replication servers are synchronized, the primary server releases its role. (For details, see our earlier paper [3]).

When there are no writers, the performance of our system is identical to a read-only replication system: all requests are serviced by a nearby server with no additional overhead. However, when updates occur, there are costs for maintaining consistent access. E.g., write sharing is synchronized by passing all client requests to the primary server, so clients being served elsewhere experience additional latencies as their requests and replies are relayed.

Write sharing is usually rare, but replication introduces two other sources of overhead. First, before a client can write a file or modify a directory, the system must use a consensus algorithm [38] to elect a primary server. Second, a primary server is responsible for distributing updates to other replication servers during file or directory modification.

We address the cost of electing a primary server by amortizing it over multiple updates: we allow a primary server to take control over more than just a single file or directory. In particular, we allow an election to grant control for a directory and all of its constituent entries or even for the entire subtree rooted at a directory. Our experimental results confirm that this strategy reduces the overhead for replication control to a negligible amount, even for update-intensive applications.

Reducing the cost of updating replication servers suggests a number of design options, each providing a different tradeoff between performance and failure resilience. For example, instead of awaiting update acknowledgements from all replication servers before processing a client update, a primary server can allow the client to proceed when it has heard from a majority of the replication servers. With this requirement, as long as more than half of the replication servers are available, a fresh copy of the file or directory can always be recovered. However, for scientific applications characterized by many synchronous updates, performance still suffers when most replication servers are distant [7].

On the other hand, if we allow a primary server to respond immediately to a client update and distribute the update to the other replication servers asynchronously, the latency penalty is eliminated. However, updates are at risk of loss if the primary server fails.

Between these two options, we can require that a primary server distribute updates to a specified number of backup servers before acknowledging a client update request. This still puts durability at risk, but reduces the risk: data is lost only if all of the updated servers fail simultaneously. Furthermore, while this approach reduces the cost of updating replication servers, it does not eliminate that cost.

We assume that the cost of updating a remote replication server is accounted for by its distance: updating nearby servers introduces low latency while updating distant servers leads to long latency. However, we hypothesize that the closer two servers are from each other, the more likely it is that they might fail at the same time. This introduces another tradeoff in designing a replication strategy.

Summarizing, maintaining synchronous replication servers can insulate a computation from failure, but increases the running time. For failure rates below some threshold, it is better not to distribute updates synchronously. When synchronous replication is advantageous, increasing the number of up-to-date replication servers improves the durability of application updates. Meanwhile, failure is correlated with the distance among these servers, so we should maintain synchronous data copies on distant servers as

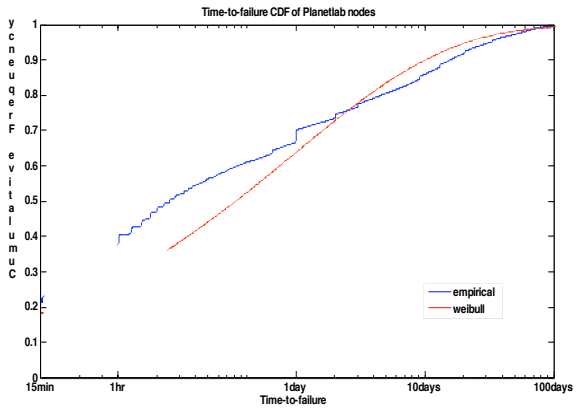

Figure 1. Time-to-failure CDF of PlanetLab nodes.

well as nearby ones. However, the cost of replication increases with the distance to the servers.

To determine the best replication configuration, we need to consider the failure conditions of the running environment, as well as application characteristics. Generally, we want to maintain more synchronous data copies when component failures are frequent and when applications are computation intensive. If failures are rare or applications rely heavily on synchronous writes or metadata updates, a delayed update distribution policy might provide a better performance tradeoff. In the following sections, we explore these tradeoffs.

3. Modeling Failure

To evaluate a replication strategy, we need to know the frequency, probability distribution, and correlation of failure. Our focus is on wide-area distribution, so we use PlanetLab [2] to exemplify a wide-area distributed computing environment. PlanetLab is an open, globally distributed platform, consisting (at this writing) of 716 machines, hosted at 349 sites, spanning 25 countries. All PlanetLab machines are connected to the Internet, which creates a unique environment for conducting experiments at Internet scale. We find PlanetLab a well-suited platform to study failure characteristics of large-scale distributed computing: PlanetLab nodes experience many of the correlated failures expected in widely distributed computation platforms. Moreover, failure traces of PlanetLab are collected over a long term and publicly available.

We use failure distribution data from the all-pairs ping data [20] collected from January 2004 to June 2005. The data set consists of ICMP echo request/reply packets ("pings") sent every 15 minutes between all pairs of PlanetLab nodes, 692 nodes in total. Each node recorded and stored its results locally and periodically transferred the results to a central archive. We classify a node live in a 15-minute interval if at least one ping sent to it in that interval succeeded. If the archive received no data from a node for the given time period, then that node is classified failed. Thus, the failures detected in our study include nodes that crashed as well as network failures that partitioned nodes from the others. This agrees with the failure conditions in Grid computing: from an application's point of view, a network failure makes the data generated on a partitioned node inaccessible to other computing elements and requires that the partition be recovered to advance the computation.

An important measure in reliability study is time-to-failure (TTF), i.e., continuous time intervals when a node is live. Figure 2 shows the cumulative frequency of PlanetLab node TTF. The mean TTF is 122.8 hours. Previous studies have shown that TTF can be modeled by a Weibull distribution [6, 7, 9] and our analysis agrees: the best-fit Weibull distribution generated with MATLAB, shown in Figure 2, agrees pretty well with the empirical data. The scale and shape parameters of the best-fit Weibull distribution are 8.0556E+04 and 0.3549, respectively.

We next investigate correlated failures among PlanetLab nodes. In related work, Chun et al. use conditional probabilities P(X is down | Y is down) to characterize the correlated failures between nodes X and Y [19]. Since we assume that a failed node can be replaced with an active one when failure happens, we are more interested in the frequency that two nodes fail at the same time instead of the amount of time that two nodes are down simultaneously. We therefore quantify the failure correlations for nodes X and Y with the conditional probabilities P(X fails at time t | Y fails at time t). Similarly, we measure the failure correlation for nodes X1, X2, …, Xn by computing the conditional probabilities P(X2, …, Xn all fail at time t | X1 fails at time t). We note that in the formula, X1, X2, …, Xn are all supposed to be alive before time t. Thus, given a group of nodes, our calculation uses only the failure times that satisfy this condition.

We first look at the failure correlations for nodes in the same site. Our analysis proceeds as follows. We first pick a node from each PlanetLab site and then select a different node from the same site to calculate the failure correlations. In the failure data we analyzed, 264 sites have more than two nodes (but only 259 of them contain more than two nodes that simultaneously live), 65 sites have more than three nodes, 21 sites have more than four nodes, and only 11 sites have more than five PlanetLab nodes.

Table 1 presents the average failure correlations computed with different number of nodes and PlanetLab sites. In the table, the first column indicates the number of nodes we select from a PlanetLab site to compute the failure correlations. The first row enumerates the number of PlanetLab sites that contain more than 2, 3, 4, and 5 nodes, respectively. The data marked in bold on row **N** is calculated with the failure data from all the PlanetLab sites that contain at least **N** nodes. For comparison, we also compute the failure correlations with fewer sites, shown in the upper right part of the table above the diagonal.

In spite of the small numbers of sites available for computing the failure correlations among multiple nodes, several inferences can be drawn from Table 1. First, there is a high probability that two nodes in the same site fail simultaneously — more than half of the time, if one node fails, another node in the same site also fails. Furthermore, as we increase the number of nodes that we consider within a site, correlated failures do not fall dramatically. Table 1 suggests that it is common for all nodes at a site to fail simultaneously. These failures might include administrators powering down all PlanetLab nodes in a site, or network failures that partition an entire site from the rest of network.

Next, we explore the failure correlations among nodes chosen from different sites. We hypothesize that failure correlation decreases with increasing number of nodes and distance between nodes, so we focus on the impacts that these two aspects have on failure correlations.

To analyze the impact of RTT on failure correlations, we partition nodes into equivalence classes for various RTT intervals, with the length of each RTT interval set to 10 milliseconds. Specifically, for a given node X, a number *n*, and a range [*rtt*, *rtt+10*], we find all groups of *n*-1 nodes whose maximum RTT to X is between *rtt* and *rtt+10*. We then calculate the average failure correlations for all of these groups with different *n* values.

Figure 2 shows the results. For a given point <*x*, *y*> in the figure, the *x* value gives the median RTT of the corresponding RTT interval and the *y* value shows the average failure correlations for that RTT interval.

We observe that correlated failure for nodes chosen from different sites is half of that shown in Table 1. Moreover, although increasing the number of nodes reduces failure correlations, we still see correlated failures of 5-10%, even when we consider failure of four or five nodes. These correlated failures may be caused by broad DDoS attacks or system bugs.

Figure 2 bears out our hypothesis that failure correlation tends to decline as the RTTs between nodes increase. For example, when the RTT between two PlanetLab nodes is a few msec, the failure correlation is around 0.2, but when the RTT is 200 msec, the failure correlation drops to 0.13.

Table 1. Failure Correlations for PlanetLab nodes from the same site

nodes \ sites	259	65	21	11
2	**0.526**	0.593	0.552	0.561
3		**0.546**	0.440	0.538
4			**0.378**	0.488
5				**0.488**

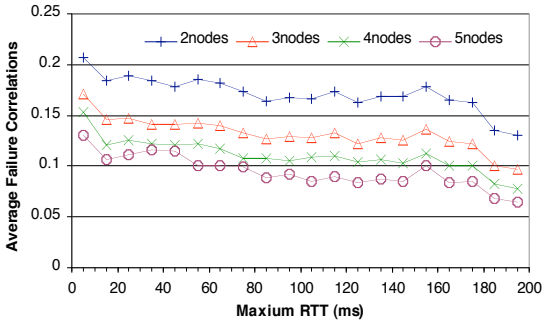

Figure 2. Failure correlations for PlanetLab nodes from different sites.

Overall, the analysis of PlanetLab failure shows that correlated failures are reduced as the number of nodes increases and as the distance between nodes increases. This suggests that we can improve the durability of data by maintaining copies on remote replicas and by increasing the number of replicas. However, both of these strategies come at a cost: the former increases update latency while the latter imposes storage and network overheads. In the next section, we propose a model that uses failure statistics and application characteristics to estimate the expected execution time of an application for various replication configurations. We then show how to use the model to minimize the expected execution time of a Grid computation by selecting an optimal replication configuration given available storage resources.

4. The Evaluation Model

In this section, we describe a model for estimating the expected running time of an application that uses a replicated file system subject to failure. We use the following nomenclature, with some terms borrowed from previous studies by other researchers on optimal checkpoint intervals [24, 25, 28].

Failure-free no-replication running time (F) is the running time of an application in the absence of failure without replication. This is equal to the execution time with a single local server that does not fail.

Replication overhead (C) is the performance penalty for maintaining synchronous data copies on replication servers (which we call backup servers in the following discussion) in a failure-free execution

following discussion) in a failure-free execution of the application. We can estimate C as follows. First, we assume (and our experiments confirm) that the replication overhead is strictly proportional to the maximal distance between the primary server and the backup servers. Let rtt represent the maximal round-trip time (in msec.) between the primary server and backup servers and let C_{msec} denote the replication overhead to update a backup server with a one msec. round-trip time from the primary server. C_{msec} depends only on application write characteristics and can be measured during a test run of the application. We can then calculate the replication overhead $C = rtt \times C_{msec}$.

Recovery time (R) is the time for the system to detect the failure of a replication server and replace it with another active server.

Expected execution time (E) is the expected application execution time in the presence of failures.

Utilization ratio (U), defined as U = F / E, describes the fraction of time that the system spends doing useful work.

We model the execution of an application with a four-state Markov chain, shown in Figure 3. Application execution begins in an initial *start* state and makes an immediate transition to the *run* state, where it remains until a replication server fails or the execution completes. Upon replication server failure, the execution is suspended by transitioning to the *recover* state. During recovery, a replacement server is sought and the system attempts to recover the data under modification on the failed server. If a synchronous data copy survives on any active replication server, the system can recover the data on the application's behalf. On the other hand, if the failed server holds the only valid copy of the data (i.e., the server distributes updates to other replication servers asynchronously) or if all replication servers that maintain synchronous copies fail simultaneously, then the system cannot recover the data generated up to the point that the execution halted. After the failure recovery, the client where the application executes is migrated to the replacement server. Then depending on whether the output data generated by the application is recovered, the application either resumes its computation (continue in the *run* state) or restarts from the beginning (from the initial *start* state). When execution finishes, the application exits to the *end* state.

In the Markov model just described, the expected running time of an application in the presence of failure can be expressed as the expectation of the time to transit from the initial *start* state to the *end* state. This can be estimated using the specified time-to-failure distribution and the failure correlations of the

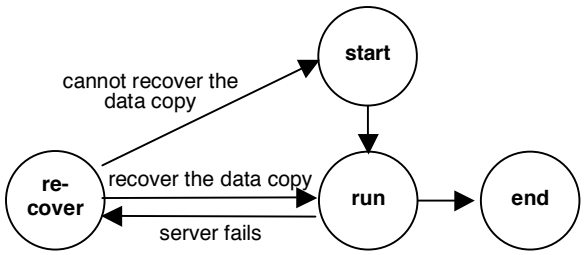

Figure 3. Four-state Markov chain describing the execution of an application over a replicated file system in the presence of failures.

replication servers that maintain synchronous data copies. In particular, the time-to-failure distribution determines the waiting time in the *run* state before moving to the *recover* state, while the failure correlation gives the probability of moving from the *recover* state to the *start* state.

In our study, we calculate the expected execution time of an application through simulation. We wrote a simulator that takes input the time-to-failure distribution data and the running time parameters of an application with a specified replication policy, i.e., F, C, and R. The simulation proceeds as follows. The simulator begins with the *start* state and moves directly to the *run* state. In the *run* state, the simulator either waits for F+C and then exists to the *end* state, or jumps to the *recover* state if a failure happens within F+C. After spending the amount of time R in the *recover* state, the simulator either moves back to the *run* state or restarts from the *start* state, with the probability of the latter equal to the given failure correlations. We assume that the same replication policy is used for an application throughout a simulation. This implies that the replication overhead C does not change after an application is migrated to a replacement server.

5. Simulation Results

In this section, we use discrete event simulation, based on the analyzed PlanetLab failure statistics from Section 3, to evaluate the efficiency of different replication policies with various application running time characteristics.

We use the replicated file system described in Section 2 as the reference model for our study. Since the system can automatically detect and recover from the failure of a replication server, we suggest that a small amount of time for failure recovery is reasonable. In our simulation experiments, we fix the failure recovery time R to 10 minutes. Further analysis (not detailed in this paper) shows that varying R in the range from 1 minute to 1 hour does not have much effect on the results for the (much larger) expected application running times we are most interested in.

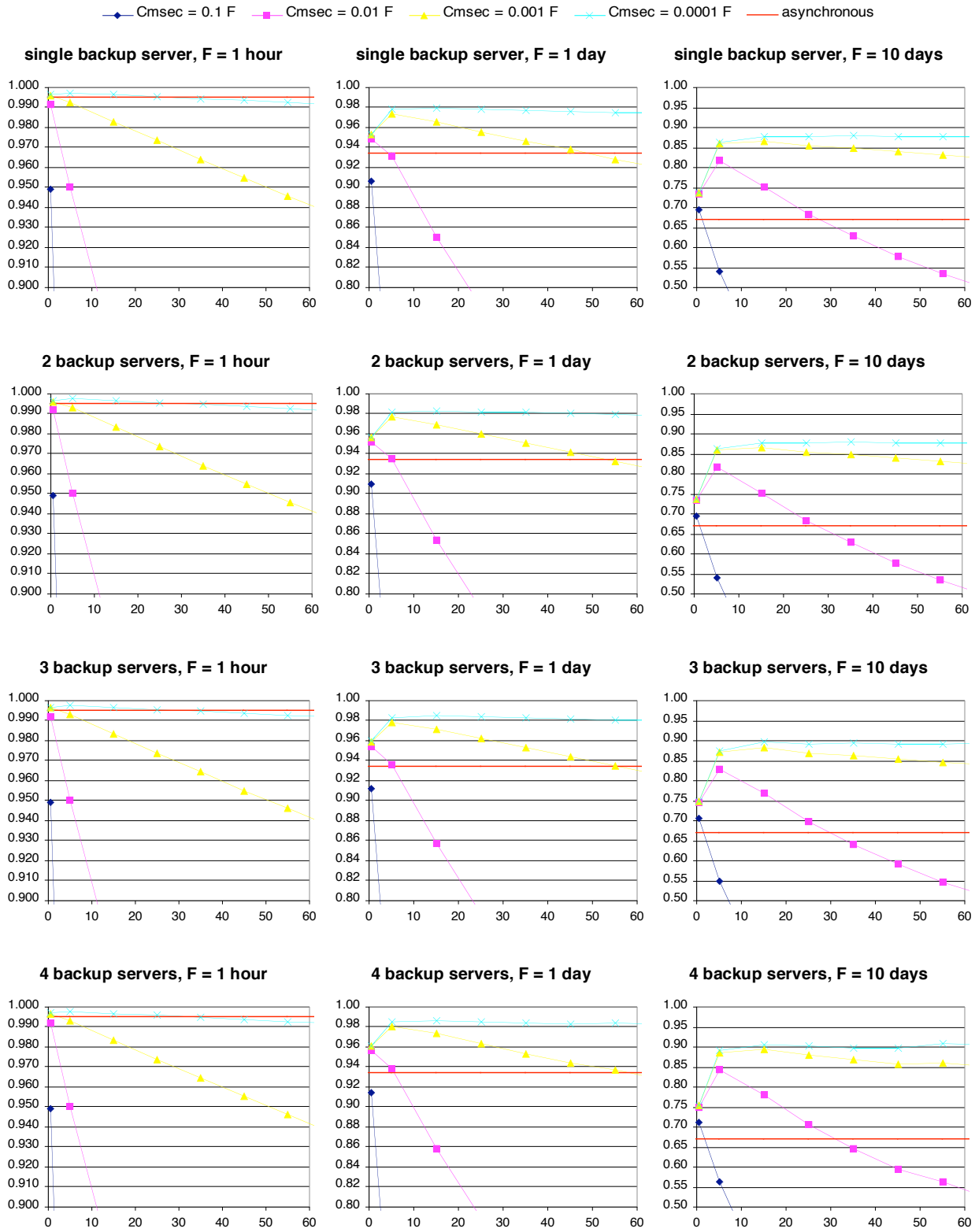

Figure 4. Utilization ratio (F/E) as the RTT between the primary server and backup servers increases. In each graph, X-axis indicates the maximum RTT (in ms) between the primary server and backup servers, and Y-axis indicates the utilization ratio.

In our simulation, each measured expected execution time is the average execution time from 100,000 consecutive runs of simulation. The PlanetLab data does not contain enough failures for so many simulations, so we use MATLAB to generate time-to-failure data from the Weibull distribution that best fits the PlanetLab failure data, analyzed in Section 3. For failure correlations with different replication configurations, we use the probability data calculated in Section 3.

Figure 4 shows the results of the simulation. In each graph, the *X*-axis indicates the maximum RTT (in milliseconds) between the primary server and backup servers, and *Y*-axis indicates the utilization ratio.

We assume that asynchronous update distribution adds no performance cost to an application's execution, i.e., C is always zero. Furthermore, with asynchronous update distribution, no synchronous data copy is available if the primary server fails, so we always restart an execution from the beginning. Thus, the utilization ratio with asynchronous update distribution depends on only the application running parameters and time-to-failure distribution. The utilization ratios with asynchronous update distribution for F = 1 hour, F = 1 day, and F = 10 days are 0.996048, 0.947075, and 0.689764, respectively, which is marked as red horizontal line in each graph.

The results suggest that applications with different characteristics benefit from different replication policies.

For applications that make heavy use of synchronous writes or metadata updates (C = 0.1F), whether long- or short-running, maintaining synchronous replicated data copies is costly even with nearby backup servers, so asynchronous update distribution is usually prescribed. For very long-running applications (10 days), the cost of losing intermediate computation results becomes enormous, so it is beneficial to maintain synchronous data copies on local backup servers. We observe that the utilization ratio for long-running applications is relatively low. This indicates the benefit of using checkpoint to shorten the modeled execution time.

For applications that write at a moderate rate (C = 0.01F), maintaining nearby backup servers provides the highest utilization. When the running time of an application is small, a local backup server offers the best tradeoff between performance and failure resilience. As the execution time of an application increases, the cost of losing intermediate computation results because of multiple failures also grows. Here, maintaining synchronous data copies in the same local area network is inadequate since this replication policy cuts correlated failures only in half. Instead, the simulation indicates that the performance penalty of backing up data to a different site is more than compensated by the expected reduction in the execution time lost to correlated failure.

If applications make few synchronous writes or metadata updates, replication overhead is relatively small even when we maintain synchronous data copies far away from the primary server. For these applications, maintaining remote backup servers provides the highest utilization.

Finally, we find that increasing the number of backup servers does not yield much improvement in utilization. For example, with F = 10 days, the maximum utilization ratio increases from 0.68 to 0.71 as we raise the number of backup servers from 1 to 4. Furthermore, we observe that increasing the distance between the primary server and backup servers provides limited advantage even for read-dominant applications. That is, although the failure analysis in Section 3 shows that increasing the number of synchronous data copies and the distance among the maintained data copies helps to reduce correlated failures, they offer small benefits for reducing the expected running time. These findings follow from the low overall failure rate; correlated failures are addressed effectively by maintaining a single backup server in a different site.

In summary, our simulation results indicate that applications with different characteristics benefit most from different replication policies. A Grid infrastructure that provides a mechanism for choosing a replication policy based on application characteristics and the failure conditions of the environment can improve the utilization of computational resources. Focusing on the tradeoff between performance and failure resilience, our evaluation omits other replication overhead such as network bandwidth and storage space. However, the work presented in this paper constitutes a first step towards dynamic replication management in the Grid computing.

6. Related Work

Our work is related to three research fields: availability studies on system, Internet services and experimental wide-area computing platforms, optimal checkpoint interval analysis, and wide-area replication studies.

Availability studies. Availability problems are widely studied by other researchers on different computing systems. In particular, we take many insights from the previous works on availability of cluster systems, Internet services, the PlanetLab test bed [1], and the continuously growing Grid computing platforms [2, 3].

There is a large amount of work on measuring and charactering failures in cluster systems. Xu et al. [4]

studied the error logs from Windows NT servers. Their analysis shows that while the average availability of individual servers is over 99%, there is a high probability that multiple servers fail within a short interval. Sahoo et al. [5] analyzed the failure data collected at an IBM research center. They find that failure rates exhibit time varying behavior and different forms of strong correlation. Heath et al. [6] studied the reboot logs from three campus clusters and observed that the time between reboots is best modeled by a Weibull distribution. This observation is also indicated by Nurmi et al. [7], who investigate the suitability of different statistical distributions to model machine availability and by Schroeder et al. in a more recent work [9] that analyzed the failure logs collected over the past 9 years at Los Alamos National Lab.

Pang et al. [10] investigated the availability characteristics of the Domain Name Service (DNS). They observe that most unavailability to DNS servers is not correlated within individual network domains. Padmanabhan et al. [12] measured the faults when repeatedly downloading content from a collection of websites. Regarding to the websites that have replicas, they find that most correlated replica failures are due to websites whose replicas are on the same subnet. The recent availability studies on peer-to-peer systems [13–17] reveal low host availabilities in such environments as most of their participants are unreliable end-users' desktops and can depart the system at will.

Several recent works investigate the availability characteristics of the globally distributed PlanetLab platform. Chun et al. [19] studied all-pairs ping data set [20] collected on PlanetLab over a three-month period. They find that failures on the PlanetLab exhibit high correlations. The similar finding is also observed and further addressed by Yalagandula [21] and Nath [22] in their studies on correlated failures of PlanetLab nodes.

As the Grid technology is still under the rapid development, few works are done on charactering component failures of the Grid infrastructure. Instead, the existing works mostly focus on job failures. The Grid2003 report [34] indicates that some projects observe the job failure rates as high as 30% and a large number of such failures are caused by overfilled disks. Li et al [35] analyzed the job failure data collected from the LHC computing Grid and argued for the importance to take into account the historical failure patterns when allocating jobs. Hwang et al. [36] proposed a framework that allows Grid applications to choose the desired fault tolerant mechanisms and evaluated the effects of the supported recovery techniques.

Research on optimal checkpoint Interval. Our work is similar in spirit to determining optimal checkpoint intervals in high-performance computing. Checkpoint is a typical technique for ameliorating the amount of re-execution in case of failures. Since checkpoint also introduces performance overhead, it is important to select an *optimal checkpoint frequency* that minimizes the expected execution of an application in the presence of failures.

The selection of optimal checkpoint intervals has been studied for a long time. The problem was first formalized by Chandy et al. on transactional systems [23]. After that, Vaidya [24] derived equations of average performance with checkpointing and rollback recovery by assuming Poisson failure distribution. Wong et al. [25] modeled the availability and performance of synchronous checkpointing in distributed computing. Plank et al. investigated the performance of parallel computing with checkpoints [27]. Their results show that the optimal number of active processors can vary widely, and the number of active processors can have a significant effect on application performance. Oliner et al. [28] evaluated the periodic checkpoint behavior of BlueGene with a failure trace collected from a large-scale cluster. The study shows that when the overhead of checkpoint is high, overly frequent checkpointing can be more detrimental to performance than failure.

Related works on replication. Many systems use replication to reduce the risk of data loss. Total Recall [29] measures and predicts the availability of its constituent hosts to determine the appropriate redundancy mechanisms and repair policies. Glacier [30] uses massive redundancy to mask large-scale correlated failures. Carbonite [31] strives to create data copies only faster than they are destroyed by permanent failures to reduce the bandwidth cost of replication maintenance. However, all these studies focus on masking the low host reliability in peer-to-peer systems. The tradeoff between availability and performance are not addressed.

Some recent studies investigate the fault-tolerant techniques against correlated failures. Phoenix [33] takes advantage of platform diversity in cooperative systems. Oceanstore [32] uses introspection to discover groups of nodes that are independent in their failure characteristics. It then chooses data replicas from such a group to enhance the system availability. These techniques can be utilized in most replication systems while the evaluation of their benefits is beyond the scope of this paper.

7. Conclusion

In this paper, we describe an evaluation model for determining the best-fit replication configuration given the specified failure statistics and application

characteristics. With the failure data from the PlanetLab platform, we evaluate the feasibility of various replication configurations in terms of the overhead they introduce and the expected cost to reproduce the execution results in case that the system cannot mask a failure from an application. Our results show that different applications desire different replication configurations and a replication system should balance the tradeoff between performance and failure resilience flexibly, based on the failure conditions of the running environment as well as application characteristics.

References

[1] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An Overlay testbed for broad-coverage services. PlanetLab Design Note PDN-03-009," ACM SIGCOMM Computer Communication Review, Vol. 33, Issue 3 (July 2003).

[2] The Globus Alliance project. http://www.globus.org/.

[3] The LHC Computing Grid (LCG) project. http://lcg.web.cern.ch/LCG/.

[4] J. Xu, Z. Kalbarczyk, and R. K. Iyer, "Networked Windows NT system field failure data analysis," in Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing (Dec. 1999).

[5] R. K. Sahoo, R. K., A. Sivasubramaniam, M. S. Squillante, and Y. Zhang, "Failure data analysis of a large-scale heterogeneous server environment," in Proceedings of the 2004 International Conference on Dependable Systems and Networks (2004).

[6] T. Heath, R. Martin, and T. D. Nguyen, "Improving cluster availability using workstation validation," in Proceedings of the ACM SIGMETRICS (2002).

[7] D. Nurmi, J. Brevik, and R. Wolski, "Modeling machine availability in enterprise and wide-area distributed computing environments," in Proceedings of Europar 2005 (Aug. 2005).

[8] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A computation management agent for multi-institutional grids," in Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (2001).

[9] B. Schroeder, and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in Proceedings of the 2006 International Conference on Dependable Systems and Networks (2006).

[10] J. Pang, J. Hendricks, A. Akella, R. De Prisco, B. Maggs, and S. Seshan, "Availability, usage, and deployment characteristics of the domain name system," in Proceedings of the 4th ACM SIGCOMM Conference on internet Measurement (2004).

[11] B. Krishnamurthy and J. Wang, "On network-aware clustering of web clients," in Proceedings of the SIGCOMM '00 Symposium on Communications Architectures and Protocols (2000).

[12] V. N. Padmanabhan, S. Ramabhadran, and J. Padhye, "Client-based characterization and analysis of End-to-End Internet faults, "Microsoft Research Technical Report, MSR-TR-2005-29 (March 2005).

[13] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs," in Proceedings of 2000 SIGMETRICS (June 2000).

[14] J. Chu, K. Labonte, and B. Levine, "Availability and locality measurements of peer-to-peer file systems," in Proceedings of ITCom: Scalability and Traffic Control in IP Networks (July 2002).

[15] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in Proceedings of Multimedia Computing and Networking (2002).

[16] R. Bhagwan, S. Savage, and G. Voelker, "Understanding availability," In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (2003).

[17] S. Guha, N. Daswani, and R. Jain, "An experimental study of the Skype Peer-to-Peer VoIP system," in Proceedings of the 5th International Workshop on Peer-to-Peer Systems (2006).

[18] N. Spring, L. Peterson, A. Bavier, and V. S. Pai, "Using planetlab for network research: Myths, realities, and best practices," ACM SIGOPS Operating Systems Review, 40(1) (2006).

[19] B. Chun, and A. Vahdat, "Workload and failure characterization on a large-scale federated testbed," Tech. Rep. IRB-TR-03-040, Intel Research Berkeley (Nov. 2003).

[20] Jeremy Stribling. PlanetLab all-pairs ping. http://infospect.planet-lab.org/pings

[21] P. Yalagandula, S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Beyond Availability: Towards a Deeper Understanding of Machine Failure Characteristics in Large Distributed Systems," in Proceedings of the First Workshop On Real Large Distributed Systems (2004).

[22] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Subtleties in tolerating correlated failures," In Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (2006).

[23] K.M. Chandy and C.V. Ramamoorthy, "Rollback and recovery strategies for computer programs," IEEE Transactions on Computers, pages 546--556 (June 1972).

[24] N. H. Vaidya, "Impact of checkpoint latency on overhead ratio of a checkpointing scheme," IEEE Transactions on Computers, C-46 (8), 942–947 (1997).

[25] K. Wong and M. Franklin, "Distributed computing systems and checkpointing," in Proceedings of the 2nd IEEE Symposium on High Performance Distributed Computing (1993).

[26] J. S. Plank, and W. R. Elwasif, "Experimental assessment of workstation failures and their impact on checkpointing systems," in Proceedings of the 28th International Symposium on Fault-Tolerant Computing (1998).

[27] J. S. Plank and M. G. Thomason, "The average availability of parallel checkpointing systems and its importance in selecting runtime parameters," in Proceedings of the 29th International Symposium on Fault-Tolerant Computing (1999).

[28] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, "Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems," in Proceedings of the 19th IEEE international Parallel and Distributed Processing Symposium, Workshop 18 - Volume 19 (2005).

[29] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker, "Total Recall: Systems support for automated availability management, in Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (2004).

[30] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: Highly durable, decentralized storage despite massive correlated failures," in Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (2005).

[31] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient replica maintenance for distributed storage systems," in Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (2006).

[32] H. Weatherspoon, T. Moscovitz, and J. Kubiatowicz, "Introspective failure analysis: Avoiding correlated failures in Peer-to-Peer systems," in Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (2002).

[33] F. P. Junqueira, R. Bhagwan, A. Hevia, K. Marzullo, and G. M. Voelker, "Surviving Internet catastrophes," in Proceedings of USENIX Annual Technical Conference (2005).

[34] I. Foster, and others, "The Grid2003 Production Grid: Principles and Practice," in Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (2004).

[35] H. Li, D. Groep, L. Wolters, and J. Templon, "Job Failure Analysis and Its Implications in a Large-scale Production Grid," in Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing (2006).

[36] Hwang, S. and C. Kesselman, "GridWorkflow: A Flexible Failure Handling Framework for the Grid," in Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (2003).