Efficient Replica Maintenance for Distributed Storage Systems

Byung-Gon Chun,† Frank Dabek,⋆ Andreas Haeberlen,‡ Emil Sit,⋆ Hakim Weatherspoon,†
M. Frans Kaashoek,⋆ John Kubiatowicz,† and Robert Morris⋆

⋆ *MIT Computer Science and Artificial Intelligence Laboratory,*
‡ *Rice University/MPI-SWS,* † *University of California, Berkeley*

Abstract

This paper considers replication strategies for storage systems that aggregate the disks of many nodes spread over the Internet. Maintaining replication in such systems can be prohibitively expensive, since every transient network or host failure could potentially lead to copying a server's worth of data over the Internet to maintain replication levels.

The following insights in designing an efficient replication algorithm emerge from the paper's analysis. First, durability can be provided separately from availability; the former is less expensive to ensure and a more useful goal for many wide-area applications. Second, the focus of a durability algorithm must be to create new copies of data objects faster than permanent disk failures destroy the objects; careful choice of policies for what nodes should hold what data can decrease repair time. Third, increasing the number of replicas of each data object does not help a system tolerate a higher disk failure probability, but does help tolerate bursts of failures. Finally, ensuring that the system makes use of replicas that recover after temporary failure is critical to efficiency.

Based on these insights, the paper proposes the Carbonite replication algorithm for keeping data durable at a low cost. A simulation of Carbonite storing 1 TB of data over a 365 day trace of PlanetLab activity shows that Carbonite is able to keep all data durable and uses 44% more network traffic than a hypothetical system that only responds to permanent failures. In comparison, Total Recall and DHash require almost a factor of two more network traffic than this hypothetical system.

1 Introduction

Wide-area distributed storage systems typically use replication to provide two related properties: durability and availability. *Durability* means that objects that an application has `put` into the system are not lost due to disk failure whereas *availability* means that `get` will be able to return the object promptly. Objects can be durably stored but not immediately available: if the only copy of an object is on the disk of a node that is currently powered off, but will someday re-join the system with disk contents intact, then that object is durable but not currently available. The paper's goal is to develop an algorithm to store immutable objects durably and at a low bandwidth cost in a system that aggregates the disks of many Internet nodes.

The threat to durability is losing the last copy of an object due to permanent failures of disks. Efficiently countering this threat to durability involves three main challenges. First, network bandwidth is a scarce resource in a wide-area distributed storage system. To store objects durably, there must be enough network capacity to create copies of objects faster than they are lost due to disk failure. Second, a system cannot always distinguish between transient failures and permanent disk failures: it may waste network bandwidth by creating new copies during transient failures. Third, after recovery from transient failures, some replicas may be on nodes that the replica lookup algorithm does not query and are thus effectively lost.

Since transient failures are common in wide-area systems, replication algorithms can waste bandwidth by making unneeded replicas. For example, the initial replication algorithm [6] that the DHash distributed hash table (DHT) [9] turned out to be inadequate to build storage applications such as UsenetDHT [34], Antiquity [11], and OverCite [35, 36].

A problem with DHash was that its design was driven by the goal of achieving 100% availability; this decision caused it to waste bandwidth by creating new replicas in response to temporary failures. Its design and similar ones (such as Total Recall [3]) are overkill for durability. Furthermore, users of many Internet applications can tolerate some unavailability. For example, Usenet readers will see all articles eventually, as long as they are stored durably. Our experience with these DHT applications has led us to the following insights:

- Durability is a more practical and useful goal than availability for applications that store objects (as op-

This research was supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660, `http://project-iris.net/`. Andreas Haeberlen was supported in part by the Max Planck Society. Emil Sit was supported in part by the Cambridge-MIT Institute. Hakim Weatherspoon was supported by an Intel Foundation PhD Fellowship.

posed to caching objects).

- The main goal of a durability algorithm should be to create new copies of an object faster than they are destroyed by disk failures; the choice of how replicas are distributed among nodes can make this task easier.
- Increasing the replication level does not help tolerate a higher average permanent failure rate, but it does help cope with bursts of failures.
- Reintegrating returning replicas is key to avoiding unnecessary copying.

Using these insights we have developed Carbonite, an efficient wide-area replication algorithm for keeping objects durable. After inserting a set of initial replicas, Carbonite begins by creating new replicas mostly in response to transient failures. However, over time it is increasingly able to ignore transient failures and approaches the goal of only producing replicas in response to permanent failures.

Carbonite's design assumes that the disks in the distributed storage system fail independently of each other: failures of geographically distributed hard drives from different manufacturers are likely to be uncorrelated.

In a year-long PlanetLab failure trace, however, we observe some correlated failures because of coordinated reinstalls of the PlanetLab software. Despite this, an evaluation using the PlanetLab failure trace shows that Carbonite is able to keep 1 TB of data durable, and consumes only 44% more network traffic than a hypothetical system that only responds to permanent failures. In comparison, Total Recall and DHash require almost a factor of two more network traffic than this hypothetical system.

The rest of this paper explains our durability models and algorithms, interleaving evaluation results into the explanation. Section 2 describes the simulated evaluation environment. Section 3 presents a model of the relationship between network capacity, amount of replicated data, number of replicas, and durability. Section 4 explains how to decrease repair time, and thus increase durability, by proper placement of replicas on servers. Section 5 presents an algorithm that reduces the bandwidth wasted making copies due to transient failures. Section 6 outlines some of the challenges that face practical implementations of these ideas, Section 7 discusses related work, and Section 8 concludes.

2 System environment

The behavior of a replication algorithm depends on the environment in which it is used: high disk failure rates or low network access link speeds make it difficult for any system to maintain durability. We will use the characteristics of the PlanetLab testbed as a representative environment when evaluating wide-area replication techniques.

Dates	1 March 2005 – 28 Feb 2006
Number of hosts	632
Number of transient failures	21255
Number of disk failures	219
Transient host downtime (s)	1208, 104647, 14242
Any failure interarrival (s)	305, 1467, 3306
Disk failures interarrival (s) (Median/Mean/90th percentile)	54411, 143476, 490047

Table 1: CoMon+PLC trace characteristics.

For explanatory purposes, we will also use a synthetic trace that makes some of the underlying trends more visible. This section describes both environments, as well as the simulator we used to evaluate our algorithm.

2.1 PlanetLab characteristics

PlanetLab is a large ($>$ 600 node) research testbed [28] with nodes located around the world. We chose this testbed as our representative environment mainly because it is a large, distributed collection of machines that has been monitored for long periods; we use this monitoring data to construct a realistic trace of failures in a mostly managed environment.

The main characteristics of PlanetLab that interest us are the rates of disk and transient failures. We use historical data collected by the CoMon project [25] to identify transient failures. CoMon has archival records collected on average every 5 minutes that include the uptime as reported by the system uptime counter on each node. We use resets of this counter to detect reboots, and we estimate the time when the node became unreachable based on the last time CoMon was able to successfully contact the node. This allows us to pinpoint failures without depending on the reachability of the node from the CoMon monitoring site.

We define a disk failure to be any permanent loss of disk contents, due to disk hardware failure or because its contents are erased accidentally or intentionally. In order to identify disk failures, the CoMon measurements were supplemented with event logs from PlanetLab Central [28]. This database automatically records each time a PlanetLab node is reinstalled (e.g., for an upgrade, or after a disk is replaced following a failure). The machine is then considered offline until the machine is assigned a regular boot state in the database. Table 1 summarizes the statistics of this trace. Figure 7(a) visualizes how transient and disk failures accumulate over time in this network.

2.2 Synthetic trace

We also generated synthetic traces of failures by drawing failure inter-arrival times from exponential distributions. Synthetic traces have two benefits. First, they let us simulate longer time periods, and second, they allow us to

increase the failure density, which makes the basic underlying trends more visible. We conjecture that exponential inter-failure times are a good model for disks that are independently acquired and operated at geographically separated sites; exponential intervals are possibly not so well justified for transient failures due to network problems.

Each synthetic trace contains 632 nodes, just like the PlanetLab trace. The mean session time and downtime match the values shown in Table 1; however, in order to increase the failure density, we extended the length to two years and reduced the average node lifetime to one year. Each experiment was run with ten different traces; the figures show the averages from these experiments.

2.3 Simulation

We use the failure traces to drive an event-based simulator. In the simulator, each node has unlimited disk capacity, but limited link bandwidth. However, it assumes that all network paths are independent so that there are no shared bottlenecks. Further it assumes that if a node is available, it is reachable from all other nodes. This is occasionally not the case on PlanetLab [14]; however, techniques do exist to mask the effects of partially unreachable nodes [1].

The simulator takes as input a trace of transient and disk failure events, node repairs and object insertions. It simulates the behavior of nodes under different protocols and produces a trace of the availability of objects and the amount of data sent and stored by each node for each hour of simulated time. Each simulation calls `put` with 50,000 data objects, each of size 20 MB. Unless otherwise noted, each node is configured with an access link capacity of 150 KBytes/s, roughly corresponding to the throughput achievable under the bandwidth cap imposed by PlanetLab. The goal of the simulations is to show the percentage of objects lost and the amount of bandwidth needed to sustain objects over time.

3 Understanding durability

We consider the problem of providing durability for a storage system composed of a large number of nodes spread over the Internet, each contributing disk space. The system stores a large number of independent pieces of data. Each piece of data is immutable. The system must have a way to name and locate data; the former is beyond the scope of this work, while the latter may affect the possible policies for placing replicas. While parts of the system will suffer temporary failures, such as network partitions or power failures, the focus of this section is on failures that result in permanent loss of data. Section 5 shows how to efficiently manage transient failures; this section describes some fundamental constraints and challenges in providing durability.

3.1 Challenges to durability

It is useful to view permanent disk and node failures as having an average rate and a degree of burstiness. To provide high durability, a system must be able to cope with both.

In order to handle some average rate of failure, a high-durability system must have the ability to create new replicas of objects faster than replicas are destroyed. Whether the system can do so depends on the per-node network access link speed, the number of nodes (and hence access links) that help perform each repair, and the amount of data stored on each failed node. When a node n fails, the other nodes holding replicas of the objects stored on n must generate replacements: objects will remain durable if there is sufficient bandwidth available on average for the lost replicas to be recreated. For example, in a symmetric system each node must have sufficient bandwidth to copy the equivalent of all data it stores to other nodes during its lifetime.

If nodes are unable to keep pace with the average failure rate, no replication policy can prevent objects from being lost. These systems are *infeasible*. If the system is infeasible, it will eventually "adapt" to the failure rate by discarding objects until it becomes feasible to store the remaining amount of data. A system designer may not have control over access link speeds and the amount of data to be stored; fortunately, choice of object placement can improve the speed that a system can create new replicas as discussed in Section 4.

If the creation rate is only slightly above the average failure rate, then a burst of failures may destroy all of an object's replicas before a new replica can be made; a subsequent lull in failures below the average rate will not help replace replicas if no replicas remain. For our purposes, these failures are *simultaneous*: they occur closer together in time than the time required to create new replicas of the data that was stored on the failed disk. Simultaneous failures pose a constraint tighter than just meeting the average failure rate: every object must have more replicas than the largest expected burst of failures. We study systems that aim to maintain a target number of replicas in order to survive bursts of failure; we call this target r_L.

Higher values of r_L do *not* allow the system to survive a higher average failure rate. For examples, if failures were to arrive at fixed intervals, then either $r_L = 2$ would always be sufficient, or no amount of replication would ensure durability. If $r_L = 2$ is sufficient, there will always be time to create a new replica of the objects on the most recently failed disk before their remaining replicas fail. If creating new replicas takes longer than the average time between failures, no fixed replication level will make the system feasible; setting a replication level higher than two would only increase the number of bytes each node must copy in response to failures, which is already infeasible at $r_L = 2$.

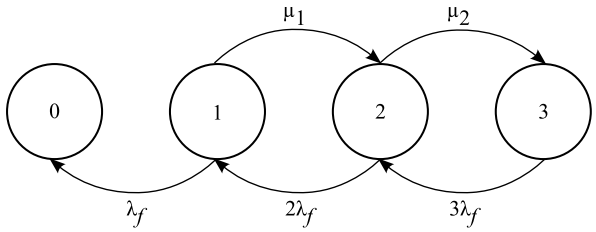

Figure 1: A continuous time Markov model for the process of replica failure and repair for a system that maintains three replicas ($r_L = 3$). Numbered states correspond to the number of replicas of each object that are durable. Transitions to the left occur at the rate at which replicas are lost; right-moving transitions happen at the replica creation rate.

3.2 Creation versus failure rate

It might seem that any creation rate higher than the average failure rate will lead to an unbounded number of replicas, thus satisfying the burst constraint. However, this intuition is false. To see why, let us model the number of replicas of an object as a birth-death process using a continuous time Markov chain, which assumes independent exponential inter-failure and inter-repair times. This assumption is reasonable for independent disk failures.

An object is in state i when i disks hold a replica of the object. There are thus $r_L + 1$ possible states, as we start with r_L replicas and only create new replicas in response to failures. From a given state i, there is a transition to state $i+1$ with rate μ_i corresponding to repair, except for state 0 which corresponds to loss of durability and state r_L which does not need repair. The actual rate μ_i depends on how bandwidth is allocated to repair and may change depending on the replication level of an object. There is a transition to the next lower state $i-1$ with rate $i\lambda_f$ because each of the i nodes holding an existing replica might fail. Figure 1 shows this model for the case where $r_L = 3$.

This model can be analyzed numerically to shed light on the impact of r_L on the probability of data loss; we will show this in Section 3.3. However, to gain some intuition about the relationship between creation and failure rates and the impact this has on the number of replicas that can be supported, we consider a simplification of Figure 1 that uses a fixed μ but repairs constantly, even allowing for transitions out of state 0. While these changes make the model less realistic, they turn the model into an M/M/∞ queue [19] where the "arrival rate" is the repair rate and the "service rate" is the per-replica failure rate. The "number of busy servers" is the number of replicas: the more replicas an object has, the more probable it is that one of them will fail.

This simplification allows us to estimate the equilibrium number of replicas: it is μ/λ_f. Given μ and λ_f, a system cannot expect to support more than this number of replicas. For example, if the system must handle coincidental bursts of five failures, it must be able to support at least six replicas and hence the replica creation rate must be at least 6 times higher than the average replica failure rate. We will refer to μ/λ_f as θ. Choices for r_L are effectively limited by θ. It is not the case that durability increases continuously with r_L; rather, when using $r_L > \theta$, the system provides the best durability it can, given its resource constraints. Higher values of θ decrease the time it takes to repair an object, and thus the 'window of vulnerability' during which additional failures can cause the object to be destroyed.

To get an idea of a real-world value of θ, we estimate μ and λ_f from the historical failure record for disks on PlanetLab. From Table 1, the average disk failure inter-arrival time for the entire test bed is 39.85 hours. On average, there were 490 nodes in the system, so we can estimate the mean time between failures for a single disk as $490 \cdot 39.85$ hours or 2.23 years. This translates to $\lambda_f \approx 0.439$ disk failures per year.

The replica creation rate μ depends on the achievable network throughput per node, as well as the amount of data that each node has to store (including replication). PlanetLab currently limits the available network bandwidth to 150 KB/s per node, and if we assume that the system stores 500 GB of unique data per node with $r_L = 3$ replicas each, then each of the 490 nodes stores 1.5 TB. This means that one node's data can be recreated in 121 days, or approximately three times per year. This yields $\mu \approx 3$ disk copies per year.

In a system with these characteristics, we can estimate $\theta = \mu/\lambda_f \approx 6.85$, though the actual value is likely to be lower. Note that this ratio represents the equilibrium number of *disks* worth of data that can be supported; if a disk is lost, all replicas on that disk are lost. When viewed in terms of disk failures and copies, θ depends on the value of r_L: as r_L increases, the total amount of data stored per disk (assuming available capacity) increases proportionally and reduces μ. If $\lambda_f = \mu$, the system can in fact maintain r_L replicas of each object.

To show the impact of θ, we ran an experiment with the synthetic trace (i.e., with 632 nodes, a failure rate of $\lambda_f = 1$ per year and a storage load of 1 TB), varying the available bandwidth per node. In this case, 100 B/s corresponds to $\theta = 1.81/r_L$. Figure 2 shows that, as θ drops below one, the system can no longer maintain full replication and starts operating in a 'best effort' mode, where higher values of r_L do not give any benefit. The exception is if some of the initial r_L replicas survive through the entire trace, which explains the small differences on the left side of the graph.

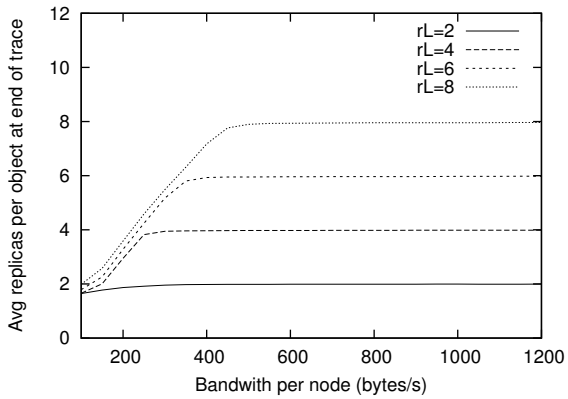

Figure 2: Average number of replicas per object at the end of a two-year synthetic trace for varying values of θ, which varies with bandwidth per node (on the x-axis) and total data stored (r_L). Where $\theta < 1$, the system cannot maintain the full replication level; increasing r_L further does not have any effect.

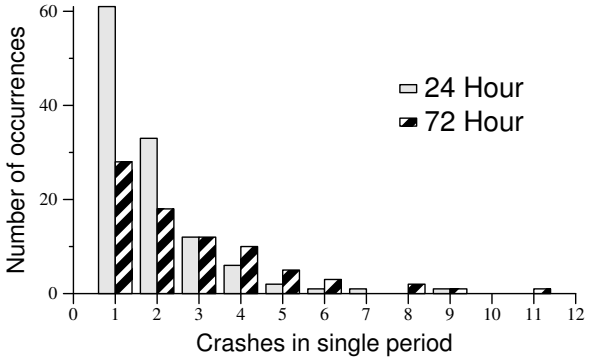

Figure 3: Frequency of "simultaneous" failures in the PlanetLab trace. These counts are derived from breaking the trace into non-overlapping 24 and 72 hour periods and noting the number of permanent failures that occur in each period. If there are x replicas of an object, there were y chances in the trace for the object to be lost; this would happen if the remaining replicas were not able to respond quickly enough to create new replicas of the object.

3.3 Choosing r_L

A system designer must choose an appropriate value of r_L to meet a target level of durability. That is, for a given deployment environment, r_L must be high enough so that a burst of r_L failures is sufficiently rare.

One approach is to set r_L to one more than the maximum burst of simultaneous failures in a trace of a real system. For example, Figure 3 shows the burstiness of permanent failures in the PlanetLab trace by counting the number of times that a given number of failures occurs in disjoint 24 hour and 72 hour periods. If the size of a failure burst exceeds the number of replicas, some objects may be lost. From this, one might conclude that 12 replicas are needed to maintain the desired durability. This value would likely provide durability but at a high cost. If a lower value of r_L would suffice, the bandwidth spent maintaining the extra replicas would be wasted.

There are several factors to consider in choosing r_L to provide a certain level of durability. First, even if failures are independent, there is a non-zero (though small) probability for every burst size up to the total number of nodes. Second, a burst may arrive while there are fewer than r_L replicas. One could conclude from these properties that the highest possible value of r_L is desirable. On the other hand, the simultaneous failure of even a large fraction of nodes may not destroy any objects, depending on how the system places replicas (see Section 4). Also, the workload may change over time, affecting μ and thus θ.

The continuous time Markov model described in Figure 1 reflects the distributions of both burst size and object replication level. The effect of these distributions is significant. An analysis of the governing differential equations can be used to derive the probability that an object will be at a given replication level after a given amount of time. In particular, we can determine the probability that the chain is in state 0, corresponding to a loss of durability.

We show the results of such an analysis in Figure 4; for details, see [7]. To explore different workloads, we consider different amounts of data per node. The graph shows the probability that an object will survive after four years as a function of r_L and data stored per node (which affects the repair rate and hence θ).

As r_L increases, the system can tolerate more simultaneous failures and objects are more likely to survive. The probability of object loss at $r_L = 1$ corresponds to using no replication. This value is the same for all curves since it depends only on the lifetime of a disk; no new replicas can be created once the only replica of the object is lost. To store 50 GB durably, the system must use an r_L of at least 3. As the total amount of data increases, the r_L required to attain a given survival probability also increases. Experiments confirm that data is lost on the PlanetLab trace only when maintaining fewer than three replicas.

4 Improving repair time

This section explores how the system can increase durability by replacing replicas from a failed disk in parallel. In effect, this reduces the time needed to repair the disk and increases θ.

Each node, n, designates a set of other nodes that can potentially hold copies of the objects that n is responsible for. We will call the size of that set the node's *scope*, and

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.