Availability in Globally Distributed Storage Systems

Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong*,
Luiz Barroso, Carrie Grimes, and Sean Quinlan
{ford,flab,florentina,mstokely}@google.com, vatruong@ieor.columbia.edu
{luiz,cgrimes,sean}@google.com
Google, Inc.

Abstract

Highly available cloud storage is often implemented with complex, multi-tiered distributed systems built on top of clusters of commodity servers and disk drives. Sophisticated management, load balancing and recovery techniques are needed to achieve high performance and availability amidst an abundance of failure sources that include software, hardware, network connectivity, and power issues. While there is a relative wealth of failure studies of individual components of storage systems, such as disk drives, relatively little has been reported so far on the overall availability behavior of large cloud-based storage services.

We characterize the availability properties of cloud storage systems based on an extensive one year study of Google's main storage infrastructure and present statistical models that enable further insight into the impact of multiple design choices, such as data placement and replication strategies. With these models we compare data availability under a variety of system parameters given the real patterns of failures observed in our fleet.

1 Introduction

Cloud storage is often implemented by complex multi-tiered distributed systems on clusters of thousands of commodity servers. For example, in Google we run Bigtable [9], on GFS [16], on local Linux file systems that ultimately write to local hard drives. Failures in any of these layers can cause data unavailability.

Correctly designing and optimizing these multi-layered systems for user goals such as data availability relies on accurate models of system behavior and performance. In the case of distributed storage systems, this includes quantifying the impact of failures and prioritizing hardware and software subsystem improvements in the datacenter environment.

We present models we derived from studying a year of live operation at Google and describe how our analysis influenced the design of our next generation distributed storage system [22].

Our work is presented in two parts. First, we measured and analyzed the *component availability*, e.g. machines, racks, multi-racks, in tens of Google storage clusters. In this part we:

- Compare mean time to failure for system components at different granularities, including disks, machines and racks of machines. (Section 3)
- Classify the failure causes for storage nodes, their characteristics and contribution to overall unavailability. (Section 3)
- Apply a clustering heuristic for grouping failures which occurs almost simultaneously and show that a large fraction of failures happen in bursts. (Section 4)
- Quantify how likely a failure burst is associated with a given failure domain. We find that most large bursts of failures are associated with rack- or multi-rack level events. (Section 4)

Based on these results, we determined that the critical element in models of availability is their ability to account for the frequency and magnitude of *correlated* failures.

Next, we consider *data availability* by analyzing unavailability at the distributed file system level, where one file system instance is referred to as a *cell*. We apply two models of multi-scale correlated failures for a variety of replication schemes and system parameters. In this part we:

- Demonstrate the importance of modeling correlated failures when predicting availability, and show their

*Now at Dept. of Industrial Engineering and Operations Research Columbia University

impact under a variety of replication schemes and placement policies. (Sections 5 and 6)

- Formulate a Markov model for data availability, that can scale to arbitrary cell sizes, and captures the interaction of failures with replication policies and recovery times. (Section 7)
- Introduce multi-cell replication schemes and compare the availability and bandwidth trade-offs against single-cell schemes. (Sections 7 and 8)
- Show the impact of hardware failure on our cells is significantly smaller than the impact of effectively tuning recovery and replication parameters. (Section 8)

Our results show the importance of considering cluster-wide failure events in the choice of replication and recovery policies.

2 Background

We study end to end data availability in a cloud computing storage environment. These environments often use loosely coupled distributed storage systems such as GFS [1, 16] due to the parallel I/O and cost advantages they provide over traditional SAN and NAS solutions. A few relevant characteristics of such systems are:

- Storage server programs running on physical machines in a datacenter, managing local disk storage on behalf of the distributed storage cluster. We refer to the storage server programs as *storage nodes* or *nodes*.
- A pool of storage service masters managing data placement, load balancing and recovery, and monitoring of storage nodes.
- A replication or erasure code mechanism for user data to provide resilience to individual component failures.

A large collection of nodes along with their higher level coordination processes [17] are called a *cell* or *storage cell*. These systems usually operate in a shared pool of machines running a wide variety of applications. A typical cell may comprise many thousands of nodes housed together in a single building or set of colocated buildings.

2.1 Availability

A storage node becomes *unavailable* when it fails to respond positively to periodic health checking pings sent by our monitoring system. The node remains unavailable until it regains responsiveness or the storage system reconstructs the data from other surviving nodes.

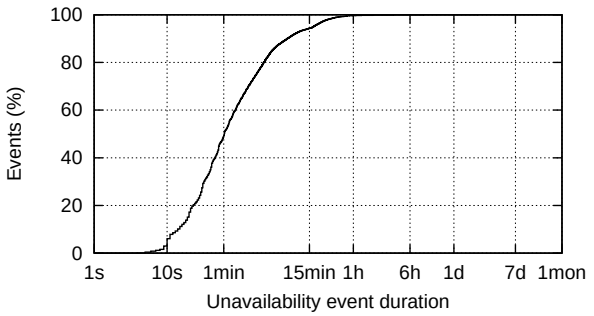

Figure 1: Cumulative distribution function of the duration of node unavailability periods.

Nodes can become unavailable for a large number of reasons. For example, a storage node or networking switch can be overloaded; a node binary or operating system may crash or restart; a machine may experience a hardware error; automated repair processes may temporarily remove disks or machines; or the whole cluster could be brought down for maintenance. The vast majority of such unavailability events are transient and do not result in permanent data loss. Figure 1 plots the CDF of node unavailability duration, showing that less than 10% of events last longer than 15 minutes. This data is gathered from tens of Google storage cells, each with 1000 to 7000 nodes, over a one year period. The cells are located in different datacenters and geographical regions, and have been used continuously by different projects within Google. We use this dataset throughout the paper, unless otherwise specified.

Experience shows that while short unavailability events are most frequent, they tend to have a minor impact on cluster-level availability and data loss. This is because our distributed storage systems typically add enough redundancy to allow data to be served from other sources when a particular node is unavailable. Longer unavailability events, on the other hand, make it more likely that faults will overlap in such a way that data could become unavailable at the cluster level for long periods of time. Therefore, while we track unavailability metrics at multiple time scales in our system, in this paper we focus only on events that are 15 minutes or longer. This interval is long enough to exclude the majority of benign transient events while not too long to exclude significant cluster-wide phenomena. As in [11], we observe that initiating recovery after transient failures is inefficient and reduces resources available for other operations. For these reasons, GFS typically waits 15 minutes before commencing recovery of data on unavailable nodes.

We primarily use two metrics throughout this paper. The average availability of all N nodes in a cell is defined as:

$$A_N = \frac{\sum_{N_i \in N} \text{uptime}(N_i)}{\sum_{N_i \in N} (\text{uptime}(N_i) + \text{downtime}(N_i))} \quad (1)$$

We use uptime(N_i) and downtime(N_i) to refer to the lengths of time a node N_i is available or unavailable, respectively. The sum of availability periods over all nodes is called *node uptime*. We define uptime similarly for other component types. We define unavailability as the complement of availability.

Mean time to failure, or *MTTF*, is commonly quoted in the literature related to the measurements of availability. We use MTTF for components that suffer transient or permanent failures, to avoid frequent switches in terminology.

$$MTTF = \frac{\text{uptime}}{\text{number failures}} \quad (2)$$

Availability measurements for nodes and individual components in our system are presented in Section 3.

2.2 Data replication

Distributed storage systems increase resilience to failures by using replication [2] or erasure encoding across nodes [28]. In both cases, data is divided into a set of *stripes*, each of which comprises a set of fixed size data and code blocks called *chunks*. Data in a stripe can be reconstructed from some subsets of the chunks. For replication, $R = n$ refers to n identical chunks in a stripe, so the data may be recovered from any one chunk. For Reed-Solomon erasure encoding, $RS(n, m)$ denotes n distinct data blocks and m error correcting blocks in each stripe. In this case a stripe may be reconstructed from any n chunks.

We call a chunk available if the node it is stored on is available. We call a stripe available if enough of its chunks are available to reconstruct the missing chunks, if any.

Data availability is a complex function of the individual node availability, the encoding scheme used, the distribution of correlated node failures, chunk placement, and recovery times that we will explore in the second part of this paper. We do not explore related mechanisms for dealing with failures, such as additional application level redundancy and recovery, and manual component repair.

3 Characterizing Node Availability

Anything that renders a storage node unresponsive is a potential cause of unavailability, including hardware component failures, software bugs, crashes, system reboots, power loss events, and loss of network connectivity. We include in our analysis the impact of software upgrades, reconfiguration, and other maintenance. These planned outages are necessary in a fast evolving datacenter environment, but have often been overlooked in other availability studies. In this section we present data for storage node unavailability and provide some insight into the main causes for unavailability.

3.1 Numbers from the fleet

Failure patterns vary dramatically across different hardware platforms, datacenter operating environments, and workloads. We start by presenting numbers for disks.

Disks have been the focus of several other studies, since they are the system component that permanently stores the data, and thus a disk failure potentially results in permanent data loss. The numbers we observe for disk and storage subsystem failures, presented in Table 2, are comparable with what other researchers have measured. One study [29] reports ARR (annual replacement rate) for disks between 2% and 4%. Another study [19] focused on storage subsystems, thus including errors from shelves, enclosures, physical interconnects, protocol failures, and performance failures. They found AFR (annual failure rate) generally between 2% and 4%, but for some storage systems values ranging between 3.9% and 8.3%.

For the purposes of this paper, we are interested in disk errors as perceived by the application layer. This includes latent sector errors and corrupt sectors on disks, as well as errors caused by firmware, device drivers, controllers, cables, enclosures, silent network and memory corruption, and software bugs. We deal with these errors with background scrubbing processes on each node, as in [5, 31], and by verifying data integrity during client reads [4]. Background scrubbing in GFS finds between 1 in 10^6 to 10^7 of older data blocks do not match the checksums recorded when the data was originally written. However, these cell-wide rates are typically concentrated on a small number of disks.

We are also concerned with node failures in addition to individual disk failures. Figure 2 shows the distribution of three mutually exclusive causes of node unavailability in one of our storage cells. We focus on *node restarts* (software restarts of the storage program running on each machine), *planned machine reboots* (e.g. kernel version upgrades), and *unplanned machine reboots* (e.g. kernel crashes). For the purposes of this figure we do not exclude events that last less than 15 minutes, but we still end the unavailability period when the system reconstructs all the data previously stored on that node. Node restart events exhibit the greatest variability in duration, ranging from less than one minute to well over an

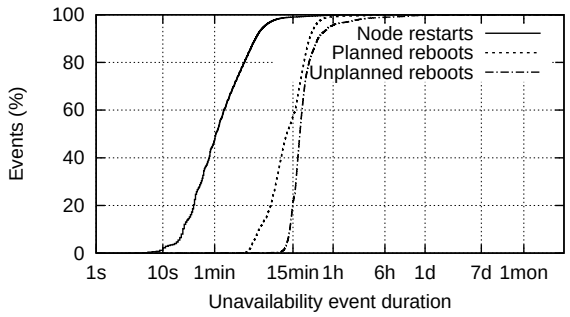

Figure 2: Cumulative distribution function of node unavailability durations by cause.

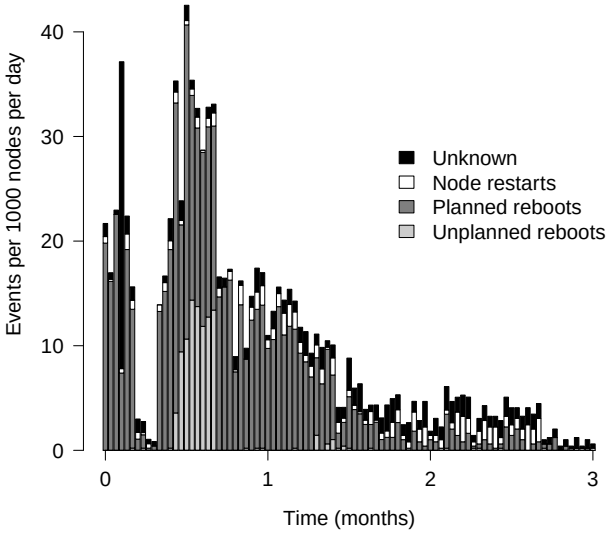

Figure 3: Rate of events per 1000 nodes per day, for one example cell.

hour, though they usually have the shortest duration. Unplanned reboots have the longest average duration since extra checks or corrective action is often required to restore machines to a safe state.

Figure 3 plots the unavailability events per 1000 nodes per day for one example cell, over a period of three months. The number of events per day, as well as the number of events that can be attributed to a given cause vary significantly over time as operational processes, tools, and workloads evolve. Events we cannot classify accurately are labeled *unknown*.

The effect of machine failures on availability is dependent on the rate of failures, as well as on how long the machines stay unavailable. Figure 4 shows the node unavailability, along with the causes that generated the unavailability, for the same cell used in Figure 3. The availability is computed with a one week rolling window, using definition (1). We observe that the majority of unavailability is generated by planned reboots.

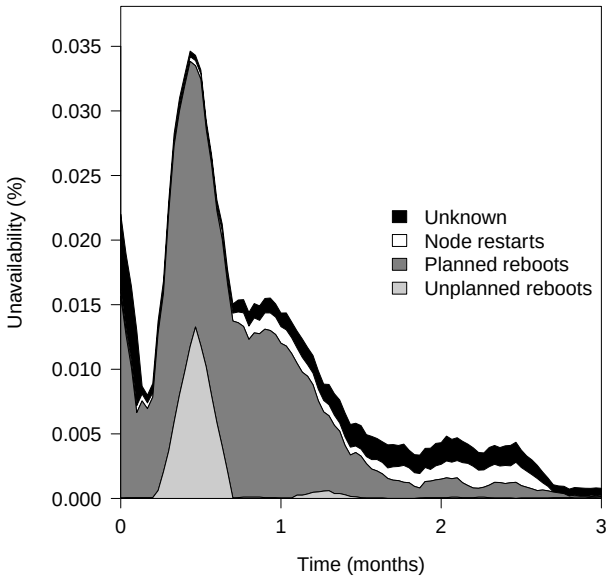

Figure 4: Storage node unavailability computed with a one week rolling window, for one example cell.

Cause	Unavailability (%) average / min / max
Node restarts	0.0139 / 0.0004 / 0.1295
Planned machine reboots	0.0154 / 0.0050 / 0.0563
Unplanned machine reboots	0.0025 / 0.0000 / 0.0122
Unknown	0.0142 / 0.0013 / 0.0454

Table 1: Unavailability attributed to different failure causes, over the full set of cells.

Table 1 shows the unavailability from node restarts, planned and unplanned machine reboots, each of which is a significant cause. The numbers are exclusive, thus the planned machine reboots do not include node restarts.

Table 2 shows the MTTF for a series of important components: disk, nodes, and racks of nodes. The numbers we report for component failures are inclusive of software errors and hardware failures. Though disks failures are permanent and most node failures are transitory, the significantly greater frequency of node failures makes them a much more important factor for system availability (Section 8.4).

4 Correlated Failures

The co-occurring failure of a large number of nodes can reduce the effectiveness of replication and encoding schemes. Therefore it is critical to take into account the statistical behavior of correlated failures to understand data availability. In this section we are more concerned with measuring the frequency and severity of such failures rather than root causes.

Component	Disk	Node	Rack
MTTF	10-50 years	4.3 months	10.2 years

Table 2: Component failures across several Google cells.

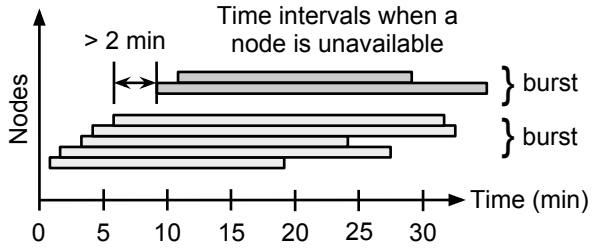

Figure 5: Seven node failures clustered into two failure bursts when the window size is 2 minutes. Note how only the unavailability start times matter.

We define a *failure burst* and examine features of these bursts in the field. We also develop a method for identifying which bursts are likely due to a failure domain. By failure domain, we mean a set of machines which we expect to simultaneously suffer from a common source of failure, such as machines which share a network switch or power cable. We demonstrate this method by validating physical racks as an important failure domain.

4.1 Defining failure bursts

We define a *failure burst* with respect to a window size w as a maximal sequence of node failures, each one occurring within a time window w of the next. Figure 5 illustrates the definition. We choose $w = 120$ s, for several reasons. First, it is longer than the frequency with which nodes are periodically polled in our system for their status. A window length smaller than the polling interval would not make sense as some pairs of events which actually occur within the window length of each other would not be correctly associated. Second, it is less than a tenth of the average time it takes our system to recover a chunk, thus, failures within this window can be considered as nearly concurrent. Figure 6 shows the fraction of individual failures that get clustered into bursts of at least 10 nodes as the window size changes. Note that the graph is relatively flat after 120 s, which is our third reason for choosing this value.

Since failures are clustered into bursts based on their times of occurrence alone, there is a risk that two bursts with independent causes will be clustered into a single burst by chance. The slow increase in Figure 6 past 120 s illustrates this phenomenon. The error incurred is small as long as we keep the window size small. Given a window size of 120 s and the set of bursts obtained from it, the probability that a random failure gets included in a

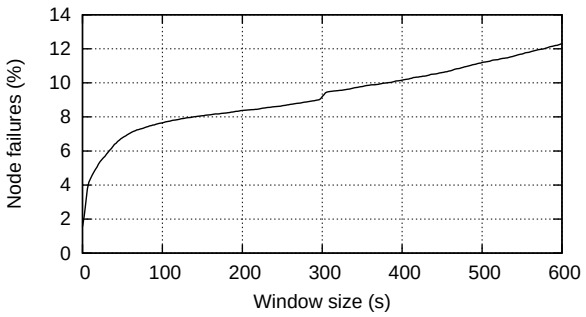

Figure 6: Effect of the window size on the fraction of individual failures that get clustered into bursts of at least 10 nodes.

burst (as opposed to becoming its own singleton burst) is 8.0%. When this inclusion happens, most of the time the random failure is combined with a singleton burst to form a burst of two nodes. The probability that a random failure gets included in a burst of at least 10 nodes is only 0.068%. For large bursts, which contribute most unavailability as we will see in Section 5.2, the fraction of nodes affected is the significant quantity and changes insignificantly if a burst of size one or two nodes is accidentally clustered with it.

Using this definition, we observe that 37% of failures are part of a burst of at least 2 nodes. Given the result above that only 8.0% of non-correlated failures may be incorrectly clustered, we are confident that close to 37% of failures are truly correlated.

4.2 Views of failure bursts

Figure 7 shows the accumulation of individual failures in bursts. For clarity we show all bursts of size at least 10 seen over a 60 day period in an example cell. In the plot, each burst is displayed with a separate shape. The n-th node failure that joins a burst at time t_n is said to have ordinal $n - 1$ and is plotted at point $(t_n, n - 1)$. Two broad classes of failure bursts can be seen in the plot:

1. Those failure bursts that are characterized by a large number of failures in quick succession show up as steep lines with a large number of nodes in the burst. Such failures can be seen, for example, following a power outage in a datacenter.

2. Those failure bursts that are characterized by a smaller number of nodes failing at a slower rate at evenly spaced intervals. Such correlated failures can be seen, for example, as part of rolling reboot or upgrade activity at the datacenter management layer.

Figure 8 displays the bursts sorted by the number of nodes and racks that they affect. The size of each bubble

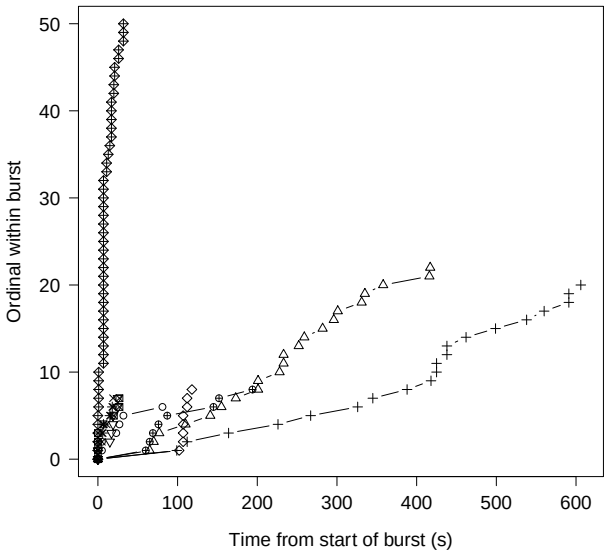

Figure 7: Development of failure bursts in one example cell.

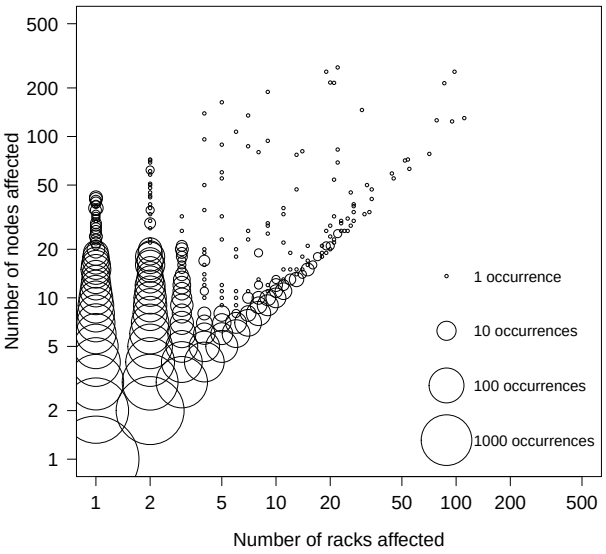

Figure 8: Frequency of failure bursts sorted by racks and nodes affected.

indicates the frequency of each burst group. The grouping of points along the 45° line represent bursts where as many racks are affected as nodes. The points furthest away from this line represent the most rack-correlated failure bursts. For larger bursts of at least 10 nodes, we find only 3% have all their nodes on unique racks. We introduce a metric to quantify this degree of domain correlation in the next section.

4.3 Identifying domain-related failures

Domain-related issues, such those associated with physical racks, network switches and power domains, are frequent causes of correlated failure. These problems can sometimes be difficult to detect directly. We introduce a metric to measure the likelihood that a failure burst is domain-related, rather than random, based on the pattern of failure observed. The metric can be used as an effective tool for identifying causes of failures that are connected to domain locality. It can also be used to evaluate the importance of domain diversity in cell design and data placement. We focus on detecting rack-related node failures in this section, but our methodology can be applied generally to any domain and any type of failure.

Let a failure burst be encoded as an n-tuple $(k_1, k_2, \ldots, k_n)$, where $k_1 \leq k_2 \leq \ldots \leq k_n$. Each k_i gives the number of nodes affected in the i-th rack affected, where racks are ordered so that these values are increasing. This *rack-based encoding* captures all relevant information about the rack locality of the burst. Let the *size* of the burst be the number of nodes that are affected, i.e., $\sum_{i=1}^{n} k_i$. We define the *rack-affinity score* of a burst to be

$$\sum_{i=1}^{n} \frac{k_i(k_i-1)}{2}$$

Note that this is the number of ways of choosing two nodes from the burst within the same rack. The score allows us to compare the rack concentration of bursts of the same size. For example the burst $(1, 4)$ has score 6. The burst $(1, 1, 1, 2)$ has score 1 which is lower. Therefore, the first burst is more concentrated by rack. Possible alternatives for the score include the sum of squares $\sum_{i=1}^{n} k_i^2$ or the negative entropy $\sum_{i=1}^{n} k_i \log(k_i)$. The sum of squares formula is equivalent to our chosen score because for a fixed burst size, the two formulas are related by an affine transform. We believe the entropy-inspired formula to be inferior because its log factor tends to downplay the effect of a very large k_i. Its real-valued score is also a problem for the dynamic program we use later in computation.

We define the *rack affinity* of a burst in a particular cell to be the probability that a burst of the same size affecting randomly chosen nodes in that cell will have a smaller burst score, plus half the probability that the two scores are equal, to eliminate bias. Rack affinity is therefore a number between 0 and 1 and can be interpreted as a vertical position on the cumulative distribution of the scores of random bursts of the same size. It can be shown that for a random burst, the expected value of its rack affinity is exactly 0.5. So we define a rack-correlated burst to be one with a metric close to 1, a rack-uncorrelated burst to be one with a metric close to 0.5, and a rack-anti-correlated burst to be one with a metric close to 0 (we have not observed such a burst). It is possible to ap-

proximate the metric using simulation of random bursts. We choose to compute the metric exactly using dynamic programming because the extra precision it provides allows us to distinguish metric values very close to 1.

We find that, in general, larger failure bursts have higher rack affinity. All our failure bursts of more than 20 nodes have rack affinity greater than 0.7, and those of more than 40 nodes have affinity at least 0.9. It is worth noting that some bursts with high rack affinity do not affect an entire rack and are not caused by common network or power issues. This could be the case for a bad batch of components or new storage node binary or kernel, whose installation is only slightly correlated with these domains.

5 Coping with Failure

We now begin the second part of the paper where we transition from node failures to analyzing replicated data availability. Two methods for coping with the large number of failures described in the first part of this paper include data replication and recovery, and chunk placement.

5.1 Data replication and recovery

Replication or erasure encoding schemes provide resilience to individual node failures. When a node failure causes the unavailability of a chunk within a stripe, we initiate a recovery operation for that chunk from the other available chunks remaining in the stripe.

Distributed filesystems will necessarily employ queues for recovery operations following node failure. These queues prioritize reconstruction of stripes which have lost the most chunks. The rate at which missing chunks may be recovered is limited by the bandwidth of individual disks, nodes, and racks. Furthermore, there is an explicit design tradeoff in the use of bandwidth for recovery operations versus serving client read/write requests.

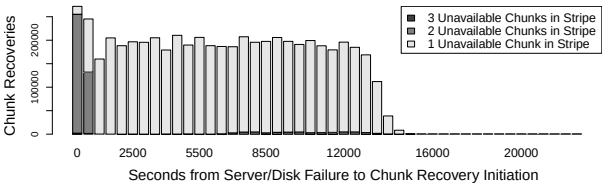

Figure 9: Example chunk recovery after failure bursts.

This limit is particularly apparent during correlated failures when a large number of chunks go missing at the same time. Figure 9 shows the recovery delay after a failure burst of 20 storage nodes affecting millions of stripes. Operators may adjust the rate-limiting seen in the figure.

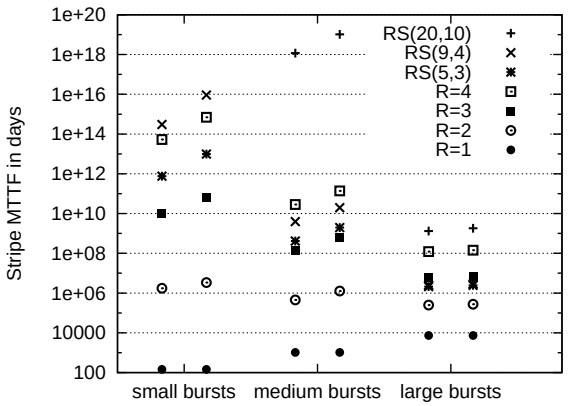

Figure 10: Stripe MTTF due to different burst sizes. Burst sizes are defined as a fraction of all nodes: small (0-0.001), medium (0.001-0.01), large (0.01-0.1). For each size, the left column represents uniform random placement, and the right column represents rack-aware placement.

The models presented in the following sections allow us to measure the sensitivity of data availability to this rate-limit and other parameters, described in Section 8.

5.2 Chunk placement and stripe unavailability

To mitigate the effect of large failure bursts in a single failure domain we consider known failure domains when placing chunks within a stripe on storage nodes. For example, racks constitute a significant failure domain to avoid. A rack-aware policy is one that ensures that no two chunks in a stripe are placed on nodes in the same rack.

Given a failure burst, we can compute the expected fraction of stripes made unavailable by the burst. More generally, we compute the probability that exactly k chunks are affected in a stripe of size n, which is essential to the Markov model of Section 7. Assuming that stripes are uniformly distributed across nodes of the cell, this probability is a ratio where the numerator is the number of ways to place a stripe of size n in the cell such that exactly k of its chunks are affected by the burst, and the denominator is the total number of ways to place a stripe of size n in the cell. These numbers can be computed combinatorially. The same ratio can be used when chunks are constrained by a placement policy, in which case the numerator and denominator are computed using dynamic programming.

Figure 10 shows the stripe MTTF for three classes of burst size. For each class of bursts we calculate the average fraction of stripes affected per burst and the rate of bursts, to get the combined MTTF due to that class. We see that for all encodings except $R = 1$, large failure bursts are the biggest contributor to unavailability

despite the fact that they are much rarer. We also see that for small and medium bursts sizes, and large encodings, using a rack-aware placement policy increases the stripe MTTF by a factor of 3 typically. This is a significant gain considering that in uniform random placement, most stripes end up with their chunks on different racks due to chance.

6 Cell Simulation

This section introduces a trace-based simulation method for calculating availability in a cell. The method replays observed or synthetic sequences of node failures and calculates the resulting impact on stripe availability. It offers detailed view of availability in short time frames.

For each node, the recorded events of interest are *down*, *up* and *recovery complete* events. When all nodes are up, they are each assumed to be responsible for an equal number of chunks. When a node goes down it is still responsible for the same number of chunks until 15 minutes later when the chunk recovery process starts. For simplicity and conservativeness, we assume that all these chunks remain unavailable until the *recovery complete* event. A more accurate model could model recovery too, such as by reducing the number of unavailable chunks linearly until the *recovery complete* event, or by explicitly modelling recovery queues.

We are interested in the expected number of stripes that are unavailable for at least 15 minutes, as a function of time. Instead of simulating a large number of stripes, it is more efficient to simulate all possible stripes, and use combinatorial calculations to obtain the expected number of unavailable stripes given a set of down nodes, as was done in Section 5.2.

As a validation, we can run the simulation using the stripe encodings that were in use at the time to see if the predicted number of unavailable stripes matches the actual number of unavailable stripes as measured by our storage system. Figure 11 shows the result of such a simulation. The prediction is a linear combination of the predictions for individual encodings present, in this case mostly $RS(5, 3)$ and $R = 3$.

Analysis of hypothetical scenarios may also be made with the cell simulator, such as the effect of encoding choice and of chunk recovery rate. Although we may not change the frequency and severity of bursts in an observed sequence, bootstrap methods [13] may be used to generate synthetic failure traces with different burst characteristics. This is useful for exploring sensitivity to these events and the impact of improvements in datacenter reliability.

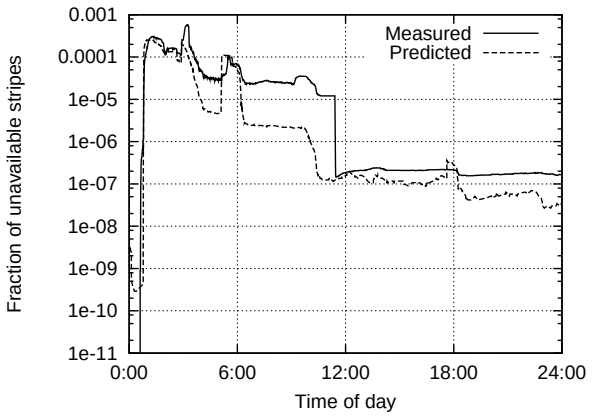

Figure 11: Unavailability prediction over time for a particular cell for a day with large failure bursts.

7 Markov Model of Stripe Availability

In this section, we formulate a Markov model of data availability. The model captures the interaction of different failure types and production parameters with more flexibility than is possible with the trace-based simulation described in the previous section. Although the model makes assumptions beyond those in the trace-based simulation method, it has certain advantages. First, it allows us to model and understand the impact of changes in hardware and software on end-user data availability. There are typically too many permutations of system changes and encodings to test each in a live cell. The Markov model allows us to reason directly about the contribution to data availability of each level of the storage stack and several system parameters, so that we can evaluate tradeoffs. Second, the systems we study may have unavailability rates that are so low they are difficult to measure directly. The Markov model handles rare events and arbitrarily low stripe unavailability rates efficiently.

The model focuses on the availability of a representative stripe. Let s be the total number of chunks in the stripe, and r be the minimum number of chunks needed to recover that stripe. As described in Section 2.2, $r = 1$ for replicated data and $r = n$ for $RS(n, m)$ encoded data. The state of a stripe is represented by the number of available chunks. Thus, the states are $s, s-1, \ldots, r, r-1$ with the state $r - 1$ representing all of the *unavailable* states where the stripe has less than the required r chunks available. Figure 12 shows a Markov chain corresponding to an $R = 2$ stripe.

The Markov chain transitions are specified by the rates at which a stripe moves from one state to another, due to chunk failures and recoveries. Chunk failures reduce the number of available chunks, and several chunks may fail 'simultaneously' in a failure burst event. Balancing this, recoveries increase the number of available chunks if any

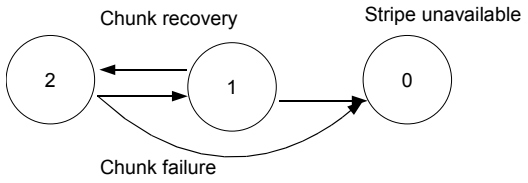

Figure 12: The Markov chain for a stripe encoded using $R = 2$.

are unavailable.

A key assumption of the Markov model is that events occur independently and with constant rates over time. This independence assumption, although strong, is not the same as the assumption that individual chunks fail independently of each other. Rather, it implies that failure events are independent of each other, but each event may involve multiple chunks. This allows a richer and more flexible view of the system. It also implies that recovery rates for a stripe depend only on its own current state.

In practice, failure events are not always independent. Most notably, it has been pointed out in [29] that the time between disk failures is not exponentially distributed and exhibits autocorrelation and long-range dependence. The Weibull distribution provides a much better fit for disk MTTF.

However, the exponential distribution is a reasonable approximation for the following reasons. First, the Weibull distribution is a generalization of the exponential distribution that allows the rate parameter to increase over time to reflect the aging of disks. In a large population of disks, the mixture of disks of different ages tends to be stable, and so the average failure rate in a cell tends to be constant. When the failure rate is stable, the Weibull distribution provides the same quality of fit as the exponential. Second, disk failures make up only a small subset of failures that we examined, and model results indicate that overall availability is not particularly sensitive to them. Finally, other authors ([24]) have concluded that correlation and non-homogeneity of the recovery rate and the mean time to a failure event have a much smaller impact on system-wide availability than the size of the event.

7.1 Construction of the Markov chain

We compute the transition rate due to failures using observed failure events. Let λ denote the rate of failure events affecting chunks, including node and disk failures. For any observed failure event we compute the probability that it affects k chunks out of the i available chunks in a stripe. As in Section 6, for failure bursts this computation takes into account the stripe placement strategy. The rate and severity of bursts, node, disk, and other failures may be adjusted here to suit the system parameters under exploration.

Averaging these probabilities over all failures events gives the probability, $p_{i,j}$, that a random failure event will affect $i-j$ out of i available chunks in a stripe. This gives a rate of transition from state i to state $j < i$, of $\lambda_{i,j} = \lambda p_{i,j}$ for $s \geq i > j \geq r$ and $\lambda_{i,r-1} = \lambda \sum_{j=0}^{r-1} p_{i,j}$ for the rate of reaching the unavailable state. Note that transitions from a state to itself are ignored.

For chunk recoveries, we assume a fixed rate of ρ for recovering a single chunk, i.e. moving from a state i to $i + 1$, where $r \leq i < s$. In particular, this means we assume that the recovery rate does not depend on the total number of unavailable chunks in the cell. This is justified by setting ρ to a lower bound for the rate of recovery, based on observed recovery rates across our storage cells or proposed system performance parameters. While parallel recovery of multiple chunks from a stripe is possible, $\rho_{i,i+1} = (s - i)\rho$, we model serial recovery to gain more conservative estimates of stripe availability.

As with [12], the distributed systems we study use prioritized recovery for stripes with more than one chunk unavailable. Our Markov model allows state-dependent recovery that captures this prioritization, but for ease of exposition we do not use this added degree of freedom.

Finally, transition rates between pairs of states not mentioned are zero.

With the Markov chain thus completely specified, computing the MTTF of a stripe, as the mean time to reach the 'unavailable state' $r - 1$ starting from state s, follows by standard methods [27].

7.2 Extension to multi-cell replication

The models introduced so far can be extended to compute the availability of multi-cell replication schemes. An example of such a scheme is $R = 3 \times 2$, where six replicas of the data are distributed as $R = 3$ replication in each of two linked cells. If data becomes unavailable at one cell then it is automatically recovered from another linked cell. These cells may be placed in separate datacenters, even on separate continents. Reed-Solomon codes may also be used, giving schemes such as $RS(6, 3) \times 3$ for three cells each with a $RS(6, 3)$ encoding of the data. We do not consider here the case when individual chunks may be combined from multiple cells to recover data, or other more complicated multi-cell encodings.

We compute the availability of stripes that span cells by building on the Markov model just presented. Intuitively, we treat each cell as a 'chunk' in the multi-cell 'stripe', and compute its availability using the Markov model. We assume that failures at different data centers are independent, that is, that they lack a single point of failure such as a shared power plant or network link. Ad-

ditionally, when computing the cell availability, we account for any cell-level or datacenter-level failures that would affect availability.

We build the corresponding transition matrix that models the resulting multi-cell availability as follows. We start from the transition matrices M_i for each cell, as explained in the previous section. We then build the transition matrix for the combined scheme as the tensor product of these, $\bigotimes_i M_i$, plus terms for whole cell failures, and for cross-cell recoveries if the data becomes unavailable in some cells but is still available in at least one cell. However, it is a fair approximation to simply treat each cell as a highly-reliable chunk in a multi-cell stripe, as described above.

Besides symmetrical cases, such as R = 3×2 replication, we can also model inhomogeneous replication schemes, such as one cell with $R = 3$ and one with $R = 2$. The state space of the Markov model is the product of the state space for each cell involved, but may be approximated again by simply counting how many of each type of cell is available.

A point of interest here is the recovery bandwidth between cells, quantified in Section 8.5. Bandwidth between distant cells has significant cost which should be considered when choosing a multi-cell replication scheme.

8 Markov Model Findings

In this section, we apply the Markov models described above to understand how changes in the parameters of the system will affect end-system availability.

8.1 Markov model validation

We validate the Markov model by comparing MTTF predicted by the model with actual MTTF values observed in production cells. We are interested in whether the Markov model provides an adequate tool for reasoning about stripe availability. Our main goal in using the model is providing a relative comparison of competing storage solutions, rather than a highly accurate prediction of any particular solution.

We underline two observations that surface from validation. First, the model is able to capture well the effect of failure bursts, which we consider as having the most impact on the availability numbers. For the cells we observed, the model predicted MTTF with the same order of magnitude as the measured MTTF. In one particular cell, besides more regular unavailability events, there was a large failure burst where tens of nodes became unavailable. This resulted in an MTTF of 1.76E+6 days, while the model predicted 5E+6 days. Though the relative error exceeds 100%, we are satisfied with the model accuracy, since it still gives us a powerful enough tool to make decisions, as can be seen in the following sections.

Second, the model can distinguish between failure bursts that span racks, and thus pose a threat to availability, and those that do not. If one rack goes down, then without other events in the cell, the availability of stripes with R=3 replication will not be affected, since the storage system ensures that chunks in each stripe are placed on different racks. For one example cell, we noticed tens of medium sized failure bursts that affected one or two racks. We expected the availability of the cell to stay high, and indeed we measured MTTF = 29.52E+8 days. The model predicted 5.77E+8 days. Again, the relative error is significant, but for our purposes the model provides sufficiently accurate predictions.

Validating the model for all possible replication and Reed-Solomon encodings is infeasible, since our production cells are not set up to cover the complete space of options. However, because of our large number of production cells we are able to validate the model over a range of encodings and operating conditions.

8.2 Importance of recovery rate

To develop some intuition about the sensitivity of stripe availability to recovery rate, consider the situation where there are no failure bursts. Chunks fail independently with rate λ and recover with rate ρ. As in the previous section, consider a stripe with s chunks total which can survive losing at most $s-r$ chunks, such as $RS(r, s-r)$. Thus the transition rate from state $i \geq r$ to state $i-1$ is $i\lambda$, and from state i to $i+1$ is ρ for $r \geq i < s$.

We compute the MTTF, given by the time taken to reach state $r-1$ starting in state s. Using standard methods related to *Gambler's Ruin*, [8, 14, 15, 26], this comes to:

$$\frac{1}{\lambda}\left(\sum_{k=0}^{s-r}\sum_{i=0}^{k}\frac{\rho^i}{\lambda^i}\frac{1}{(s-k+i)_{(i+1)}}\right)$$

where $(a)_{(b)}$ denotes $(a)(a-1)(a-2)\cdots(a-b+1)$.

Assuming recoveries take much less time than node MTTF (i.e. $\rho >> \lambda$), gives a stripe MTTF of:

$$\frac{\rho^{s-r}}{\lambda^{s-r+1}}\frac{1}{(s)_{(s-r+1)}} + O\left(\frac{\rho^{s-r-1}}{\lambda^{s-r}}\right)$$

By similar computations, the recovery bandwidth consumed is approximately λs per r data chunks.

Thus, with no correlated failures reducing recovery times by a factor of μ will increase stripe MTTF by a factor of μ^2 for $R = 3$ and by μ^4 for $RS(9, 4)$.

Reducing recovery times is effective when correlated failures are few. For $RS(6, 3)$ with no correlated failures, a 10% reduction in recovery time results in a 19% reduction in unavailability. However, when correlated failures

Policy (% overhead)	MTTF(days) with correlated failures	MTTF(days) w/o correlated failures
$R = 2$ (100)	$1.47E+5$	$4.99E+05$
$R = 3$ (200)	$6.82E+6$	$1.35E+09$
$R = 4$ (300)	$1.40E+8$	$2.75E+12$
$R = 5$ (400)	$2.41E+9$	$8.98E+15$
$RS(4,2)$ (50)	$1.80E+6$	$1.35E+09$
$RS(6,3)$ (50)	$1.03E+7$	$4.95E+12$
$RS(9,4)$ (44)	$2.39E+6$	$9.01E+15$
$RS(8,4)$ (50)	$5.11E+7$	$1.80E+16$

Table 3: Stripe MTTF in days, corresponding to various data redundancy policies and space overhead.

Policy (recovery time)	MTTF (days)	Bandwidth (per PB)
$R = 2 \times 2$(1day)	$1.08E+10$	6.8MB/day
$R = 2 \times 2$(1hr)	$2.58E+11$	6.8MB/day
$RS(6,3) \times 2$(1day)	$5.32E+13$	97KB/day
$RS(6,3) \times 2$(1hr)	$1.22E+15$	97KB/day

Table 4: Stripe MTTF and inter-cell bandwidth, for various multi-cell schemes and inter-cell recovery times.

are taken into account, even a 90% reduction in recovery time results in only a 6% reduction in unavailability.

8.3 Impact of correlation on effectiveness of data-replication schemes

Table 3 presents stripe availability for several data-replication schemes, measured in MTTF. We contrast this with stripe MTTF when node failures occur at the same total rate but are assumed independent.

Note that failing to account for correlation of node failures typically results in overestimating availability by at least two orders of magnitude, and eight in the case of RS(8,4). Correlation also reduces the benefit of increasing data redundancy. The gain in availability achieved by increasing the replication number, for example, grows much more slowly when we have correlated failures. Reed Solomon encodings achieve similar resilience to failures compared to replication, though with less storage overhead.

8.4 Sensitivity of availability to component failure rates

One common method for improving availability is reducing component failure rates. By inserting altered failure rates of hardware into the model we can estimate the impact of potential improvements without actually building or deploying new hardware.

We find that improvements below the node (server) layer of the storage stack do not significantly improve data availability. Assuming $R = 3$ is used, a 10% reduction in the latent disk error rate has a negligible effect on stripe availability. Similarly, a 10% reduction in the disk failure rate increases stripe availability by less than 1.5%. On the other hand, cutting node failure rates by 10% can increase data availability by 18%. This holds generally for other encodings.

8.5 Single vs multi-cell replication schemes

Table 4 compares stripe MTTF under several multi-cell replication schemes and inter-cell recovery times, taking into consideration the effect of correlated failures within cells.

Replicating data across multiple cells (data centers) greatly improves availability because it protects against correlated failures. For example, $R = 2 \times 2$ with 1 day recovery time between cells has two orders of magnitude longer MTTF than $R = 4$, shown in Table 3.

This introduces a tradeoff between higher replication in a single cell and the cost of inter-cell bandwidth. The extra availability for $R = 2 \times 2$ with 1 day recoveries versus $R = 4$ comes at an average cost of 6.8 MB/(user PB) copied between cells each day. This is the inverse MTTF for $R = 2$.

It should be noted that most cross-cell recoveries will occur in the event of large failure bursts. This must be considered when calculating expected recovery times between cells and the cost of on-demand access to potentially large amounts of bandwidth.

Considering the relative cost of storage versus recovery bandwidth allows us to choose the most cost effective scheme given particular availability goals.

9 Related Work

Several previous studies [3, 19, 25, 29, 30] focus on the failure characteristics of independent hardware components, such as hard drives, storage subsystems, or memory. As we have seen, these must be included when considering availability but by themselves are insufficient.

We focus on failure bursts, since they have a large influence on the availability of the system. Previous literature on failure bursts has focused on methods for discovering the relationship between the size of a failure event and its probability of occurrence. In [10], the existence of near-simultaneous failures in two large distributed systems is reported. The beta-binomial density and the bi-exponential density are used to fit these distributions in [6] and [24], respectively. In [24], the authors further note that using an over-simplistic model for burst size, for example a single size, could result in "dramatic inaccuracies" in practical settings. On the other hand, even

though the mean time to failure and mean time to recovery of system nodes tend to be non-uniform and correlated, this particular correlation effect has only a limited impact on system-wide availability.

There is limited previous work on discovering patterns of correlation in failures. The conditional probability of failures for each pair of nodes in a system has been proposed in [6] as a measure of correlation in the system. This computation extends heuristically to sets of larger nodes. A paradigm for discovering maximally independent groups of nodes in a system to cope with correlated failures is discussed in [34]. That paradigm involves collecting failure statistics on each node in the system and computing a measure of correlation, such as the mutual information, between every pair of nodes. Both of these approaches are computationally intensive and the results found, unlike ours, are not used to build a predictive analytical model for availability.

Models that have been developed to study the reliability of long-term storage fall into two categories, non-Markov and Markov models. Those in the first category tend to be less versatile. For example, in [5] the probability of multiple faults occurring during the recovery period of a stripe is approximated. Correlation is introduced by means of a multiplicative factor that is applied to the mean time to failure of a second chunk when the first chunk is already unavailable. This approach works only for stripes that are replicated and is not easily extendable to Reed-Solomon encoding. Moreover, the factor controlling time correlation is neither measurable nor derivable from other data.

In [33], replication is compared with Reed-Solomon with respect to storage requirement, bandwidth for write and repair and disk seeks for reads. However, the comparison assumes that sweep and repair are performed at regular intervals, as opposed to on demand.

Markov models are able to capture the system much more generally and can be used to model both replication and Reed-Solomon encoding. Examples include [21], [32], [11] and [35]. However, these models all assume independent failures of chunks. As we have shown, this assumption potentially leads to overestimation of data availability by many orders of magnitude. The authors of [20] build a tool to optimize the disaster recovery according to availability requirements, with similar goals as our analysis of multi-cell replication. However, they do not focus on studying the effect of failure characteristics and data redundancy options.

Node availability in our environment is different from previous work, such as [7, 18, 23], because we study a large system that is tightly coupled in a single administrative domain. These studies focus on measuring and predicting availability of individual desktop machines from many, potentially untrusted, domains. Other authors [11] studied data replication in face of failures, though without considering availability of Reed-Solomon encodings or multi-cell replication.

10 Conclusions

We have presented data from Google's clusters that characterize the sources of failures contributing to unavailability. We find that correlation among node failures dwarfs all other contributions to unavailability in our production environment.

In particular, though disks failures can result in permanent data loss, the multitude of transitory node failures account for most unavailability. We present a simple time-window-based method to group failure events into failure bursts which, despite its simplicity, successfully identifies bursts with a common cause. We develop analytical models to reason about past and future availability in our cells, including the effects of different choices of replication, data placement and system parameters.

Inside Google, the analysis described in this paper has provided a picture of data availability at a finer granularity than previously measured. Using this framework, we provide feedback and recommendations to the development and operational engineering teams on different replication and encoding schemes, and the primary causes of data unavailability in our existing cells. Specific examples include:

- Determining the acceptable rate of successful transfers to battery power for individual machines upon a power outage.
- Focusing on reducing reboot times, because planned kernel upgrades are a major source of correlated failures.
- Moving towards a dynamic delay before initiating recoveries, based on failure classification and recent history of failures in the cell.

Such analysis complements the intuition of the designers and operators of these complex distributed systems.

Acknowledgments

Our findings would not have been possible without the help of many of our colleagues. We would like to thank the following people for their contributions to data collection: Marc Berhault, Eric Dorland, Sangeetha Eyunni, Adam Gee, Lawrence Greenfield, Ben Kochie, and James O'Kane. We would also like to thank a number of our colleagues for helping us improve the presentation of these results. In particular, feedback from John Wilkes, Tal Garfinkel, and Mike Marty was helpful. We

would also like to thank our shepherd Bianca Schroeder and the anonymous reviewers for their excellent feedback and comments, all of which helped to greatly improve this paper.

References

[1] HDFS (Hadoop Distributed File System) architecture. `http://hadoop.apache.org/common/docs/current/hdfs_design.html`, 2009.

[2] ANDREAS, E. S., HAEBERLEN, A., DABEK, F., GON CHUN, B., WEATHERSPOON, H., MORRIS, R., KAASHOEK, M. F., AND KUBIATOWICZ, J. Proactive replication for data durability. In *Proceedings of the 5th Intl Workshop on Peer-to-Peer Systems (IPTPS)* (2006).

[3] BAIRAVASUNDARAM, L. N., GOODSON, G. R., PASUPATHY, S., AND SCHINDLER, J. An analysis of latent sector errors in disk drives. In *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (2007), pp. 289–300.

[4] BAIRAVASUNDARAM, L. N., GOODSON, G. R., SCHROEDER, B., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEA, R. H. An analysis of data corruption in the storage stack. In *FAST '08: Proceedings of the 6th USENIX Conference on File and Storage Technologies* (2008), pp. 1–16.

[5] BAKER, M., SHAH, M., ROSENTHAL, D. S. H., ROUSSOPOULOS, M., MANIATIS, P., GIULI, T., AND BUNGALE, P. A fresh look at the reliability of long-term digital storage. In *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems* (2006), pp. 221–234.

[6] BAKKALOGLU, M., WYLIE, J. J., WANG, C., AND GANGER, G. R. Modeling correlated failures in survivable storage systems. In *Fast Abstract at International Conference on Dependable Systems & Networks* (June 2002).

[7] BOLOSKY, W. J., DOUCEUR, J. R., ELY, D., AND THEIMER, M. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on measurement and modeling of computer systems* (2000), pp. 34–43.

[8] BROWN, D. M. The first passage time distribution for a parallel exponential system with repair. In *Reliability and fault tree analysis* (1974), Defense Technical Information Center.

[9] CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. E. Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (Nov. 2006), pp. 205–218.

[10] CHARACTERISTICS, M. F., YALAG, P., NATH, S., YU, H., GIBBONS, P. B., AND SESHAN, S. Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems. In *WORLDS '04: First Workshop on Real, Large Distributed Systems* (2004).

[11] CHUN, B.-G., DABEK, F., HAEBERLEN, A., SIT, E., WEATHERSPOON, H., KAASHOEK, M. F., KUBIATOWICZ, J., AND MORRIS, R. Efficient replica maintenance for distributed storage systems. In *NSDI '06: Proceedings of the 3rd conference on Networked Systems Design & Implementation* (2006), pp. 45–58.

[12] CORBETT, P., ENGLISH, B., GOEL, A., GRCANAC, T., KLEIMAN, S., LEONG, J., AND SANKAR, S. Row-diagonal parity for double disk failure correction. In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (2004), pp. 1–14.

[13] EFRON, B., AND TIBSHIRANI, R. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.

[14] EPSTEIN, R. *The Theory of Gambling and Statistical Logic*. Academic Press, 1977.

[15] FELLER, W. *An Introduction to Probability Theory and Its Application*. John Wiley and Sons, 1968.

[16] GHEMAWAT, S., GOBIOFF, H., AND LEUNG, S.-T. The Google file system. In *SOSP '03: Proceedings of the 19th ACM Symposium on Operating Systems Principles* (Oct. 2003), pp. 29–43.

[17] ISARD, M. Autopilot: automatic data center management. *ACM SIGOPS Opererating Systems Review 41*, 2 (2007), 60–67.

[18] JAVADI, B., KONDO, D., VINCENT, J.-M., AND ANDERSON, D. Mining for statistical models of availability in large-scale distributed systems: An empirical study of SETI@home (2009). pp. 1–10.

[19] JIANG, W., HU, C., ZHOU, Y., AND KANEVSKY, A. Are disks the dominant contributor for storage failures?: a comprehensive study of storage subsystem failure characteristics. In *FAST '08: Proceedings of the 6th USENIX Conference on File and Storage Technologies* (2008), pp. 1–15.

[20] KEETON, K., SANTOS, C., BEYER, D., CHASE, J., AND WILKES, J. Designing for disasters. In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (2004), pp. 59–62.

[21] LIAN, Q., CHEN, W., AND ZHANG, Z. On the impact of replica placement to the reliability of distributed brick storage systems. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems* (2005), pp. 187–196.

[22] MCKUSICK, M. K., AND QUINLAN, S. GFS: Evolution on fast-forward. *Communications of the ACM 53*, 3 (2010), 42–49.

[23] MICKENS, J. W., AND NOBLE, B. D. Exploiting availability prediction in distributed systems. In *NSDI '06: Proceedings of the 3rd conference on Networked Systems Design & Implementation* (2006), pp. 73–86.

[24] NATH, S., YU, H., GIBBONS, P. B., AND SESHAN, S. Subtleties in tolerating correlated failures in wide-area storage systems. In *NSDI '06: Proceedings of the 3rd conference on Networked Systems Design & Implementation* (2006), pp. 225–238.

[25] PINHEIRO, E., WEBER, W.-D., AND BARROSO, L. A. Failure trends in a large disk drive population. In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies* (2007), pp. 17–23.

[26] RAMABHADRAN, S., AND PASQUALE, J. Analysis of long-running replicated systems. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications* (2006), pp. 1–9.

[27] RESNICK, S. I. *Adventures in stochastic processes*. Birkhauser Verlag, 1992.

[28] RODRIGUES, R., AND LISKOV, B. High availability in DHTs: Erasure coding vs. replication. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems* (2005).

[29] SCHROEDER, B., AND GIBSON, G. A. Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you? In *FAST '07: Proceedings of the 5th USENIX conference on File and Storage Technologies* (2007), pp. 1–16.

[30] SCHROEDER, B., PINHEIRO, E., AND WEBER, W.-D. DRAM errors in the wild: a large-scale field study. In *SIGMETRICS '09: Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems* (2009), pp. 193–204.

[31] SCHWARZ, T. J. E., XIN, Q., MILLER, E. L., LONG, D. D. E., HOSPODOR, A., AND NG, S. Disk scrubbing in large archival storage systems. *International Symposium on Modeling, Analysis, and Simulation of Computer Systems* (2004), 409–418.

[32] T., S. Generalized Reed Solomon codes for erasure correction in SDDS. In *WDAS-4: Workshop on Distributed Data and Structures* (2002).

[33] WEATHERSPOON, H., AND KUBIATOWICZ, J. Erasure coding vs. replication: A quantitative comparison. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems* (2002), Springer-Verlag, pp. 328–338.

[34] WEATHERSPOON, H., MOSCOVITZ, T., AND KUBIATOWICZ, J. Introspective failure analysis: Avoiding correlated failures in peer-to-peer systems. In *SRDS '02: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems* (2002), pp. 362–367.

[35] XIN, Q., MILLER, E. L., SCHWARZ, T., LONG, D. D. E., BRANDT, S. A., AND LITWIN, W. Reliability mechanisms for very large storage systems. In *MSS '03: Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies* (2003), pp. 146–156.