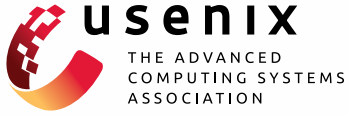

Tiered Replication: A Cost-effective Alternative to Full Cluster Geo-replication

Asaf Cidon, *Stanford University;* Robert Escriva, *Cornell University;* Sachin Katti and Mendel Rosenblum, *Stanford University;* Emin Gün Sirer, *Cornell University*

https://www.usenix.org/conference/atc15/technical-session/presentation/cidon

This paper is included in the Proceedings of the 2015 USENIX Annual Technical Conference (USENIC ATC '15).

July 8–10, 2015 • Santa Clara, CA, USA

ISBN 978-1-931971-225

Open access to the Proceedings of the 2015 USENIX Annual Technical Conference (USENIX ATC '15) is sponsored by USENIX.

Tiered Replication: A Cost-effective Alternative to Full Cluster Geo-replication

Asaf Cidon[1], Robert Escriva[2], Sachin Katti[1], Mendel Rosenblum[1], and Emin Gün Sirer[2]

[1]Stanford University

[2]Cornell University

ABSTRACT

Cloud storage systems typically use three-way random replication to guard against data loss within the cluster, and utilize cluster geo-replication to protect against correlated failures. This paper presents a much lower cost alternative to full cluster geo-replication. We demonstrate that in practical settings, using two replicas is sufficient for protecting against independent node failures, while using three random replicas is inadequate for protecting against correlated node failures.

We present Tiered Replication, a replication scheme that splits the cluster into a primary and backup tier. The first two replicas are stored on the primary tier and are used to recover data in the case of independent node failures, while the third replica is stored on the backup tier and is used to protect against correlated failures. The key insight of our paper is that, since the third replicas are rarely read, we can place the backup tier on separate physical infrastructure or a remote location without affecting performance. This separation significantly increases the resilience of the storage system to correlated failures and presents a low cost alternative to geo-replication of an entire cluster. In addition, the Tiered Replication algorithm optimally minimizes the probability of data loss under correlated failures. Tiered Replication can be executed incrementally for each cluster change, which allows it to supports dynamic environments in which nodes join and leave the cluster, and it facilitates additional data placement constraints required by the storage designer, such as network and rack awareness. We have implemented Tiered Replication on HyperDex, an open-source cloud storage system, and demonstrate that it incurs a small performance overhead. Tiered Replication improves the cluster-wide MTTF by a factor of 20,000 compared to random replication and by a factor of 20 compared to previous non-random replication schemes, without increasing the amount of storage.

1. INTRODUCTION

Popular cloud storage systems like HDFS [33], GFS [15] and Azure [6] typically replicate their data on three random machines to guard against data loss within a single cluster, and geo-replicate the entire cluster to a separate location to guard against correlated failures.

In prior literature, node failure events are broadly categorized into two types: independent node failures and correlated node failures [4, 5, 7, 14, 25, 38]. Independent node failures are defined as events during which nodes fail individually and independently in time (e.g., individual disk failure, kernel crash). Correlated failures are defined as failures in which several nodes fail simultaneously due to a common root cause [7, 11] (e.g., network failure, power outage, software upgrade). In this paper, we are focused on events that affect data durability rather than data availability, and are therefore concerned with node failures that cause permanent data loss, such as hardware and disk failures, in contrast to transient data availability events, such as software upgrades.

The conventional wisdom is that three-way replication is cost-effective for guarding against node failures within a cluster. We also note that, in many storage systems, the third replica was introduced mainly for durability and not for read performance [7, 8, 13, 34].

Our paper challenges this conventional wisdom. We show that two replicas are sufficient to protect against independent node failures, while three replicas are inadequate to protect against correlated node failures.

We show that in storage systems in which the third replica is only read when the first two are unavailable (i.e., the third replica is not used for client reads), the third replica would be used almost only during correlated failure events. In such a system, the third replica's workload is write-dominated, since it would be written to on every system write, but very infrequently read from.

This property can be leveraged by storage systems to increase durability and reduce storage costs. Storage systems can split their clusters into two tiers: the *primary tier* would contain the first and second copy of each replica, while the *backup tier* would contain the backup third replicas. The backup tier would only be

used when data is not available in the primary tier. Since the backup tier's replicas will be read infrequently they do not require high performance for read operations. The relaxed read requirements for the third replica enable system designers to further increase storage durability, by storing the backup tier on a remote site (e.g., Amazon S3), which significantly reduces the correlation in failures between nodes in the primary tier and the backup tier. This is a much lower cost alternative to full cluster geo-replication, in which all three replicas are stored in a remote site. Since the backup tier does not require high read performance, it may also be compressed, deduplicated or stored on a low-cost storage medium that does not offer low read latency but supports high write bandwidth (e.g., tape) to reduce storage capacity costs.

Existing replication schemes cannot effectively separate the cluster into tiers while maintaining cluster durability. Random replication, the scheme widely used by popular cloud storage systems, scatters data uniformly across the cluster and has been shown to be very susceptible to frequent data loss due to correlated failures [2, 7, 9]. Non-random replication schemes, like Copyset Replication [9], have significantly lower probability of data loss under correlated failures. However, Copyset Replication is not designed to split the replicas into storage tiers, does not support nodes joining and leaving, and does not allow storage system designers to add additional placement constraints, such as supporting chain replication or requiring replicas to be placed on different network partitions and racks.

We present Tiered Replication, a simple dynamic replication scheme that leverages the asymmetric workload of the third replica. Tiered Replication allows system designers to divide the cluster into primary and backup tiers, and its incremental operation supports nodes joining and leaving. In addition, unlike Random Replication, Tiered Replication enables system designers to limit the frequency of data loss under correlated failures. Moreover, Tiered Replication can support any data layout constraint, including support for chain replication [37] and topology-aware data placement.

Tiered Replication is an optimization-based algorithm that places chunks into the best available replication groups. The insight behind its operation is to select replication groups that both minimize the probability of data loss under correlated failures by reducing the overlap between replication groups, and satisfy data layout constraints defined by the storage system designer. Tiered Replication increases the MTTF by a factor of 20,000 times compared to Random Replication, and by a factor of 20 compared to Copyset Replication.

We implemented Tiered Replication on HyperDex, a cloud storage system that can scale up to hundreds of thousands of nodes [13]. Our implementation of Tiered Replication is versatile enough to satisfy constraints on replica assignment and load balancing, including HyperDex's data layout requirements for chain replication [37]. We analyze the performance of Tiered Replication on a HyperDex installation on Amazon, in which the backup tier, containing the third replicas, is stored on a separate Amazon availability zone. We show that Tiered Replication incurs a small performance overhead for normal operations and preserves the performance of node recovery. Our open source implementation of Tiered Replication on HypderDex is publicly available.

2. MOTIVATION

In this section, we demonstrate why three-way replication is not cost-effective. First, we demonstrate that it is superfluous to use a replication factor of three to provide data durability against independent failures, and that two replicas provide sufficient redundancy for this type of failure. Second, building on previous work [7, 9], we show that random three-way replication falls short in protecting against correlated failures. These findings provide motivation for a replication scheme that more efficiently handles independent node failures and provides stronger durability in the face of correlated failures.

2.1 Analysis of Independent Node Failures

Consider a storage system with N nodes and a replication factor of R. Independent node failures are modeled as a Poisson Process with an arrival rate of λ. Typical parameters for storage systems are $N = 1,000$ to $N = 10,000$ and $R = 3$ [5, 7, 14, 33].

$\lambda = \frac{N}{MTTF}$, where $MTTF$ is the mean time to permanent failure of a standard node. We borrow the failure assumption used by Yahoo and LinkedIn, for which about 1% of the nodes in a typical cluster fail independently each month [7, 33]. Consequently, we use a node MTTF of 10 years. We also assume in the model that the number of nodes remains constant and that there is always an idle server available to replace a failed node.

When a node fails, the cluster re-replicates its data from several servers, which store replicas of the node's data and write the data into another set of nodes. The node's recovery time depends on the number of servers that can be read from in parallel to recover the data. Using previously defined terminology [9], we term $scatter\ width$ or S as the average number of servers that participate in a single node's recovery. For example, a node that has $S = 10$ has its data replicated uniformly on 10 other nodes, and when the node fails, the storage system can re-replicate the data by reading from and writing to 10 nodes in parallel.

A single node's recovery time is modeled as an exponential random variable, with a recovery rate of μ. We assume that recovery rate is a linear function of the

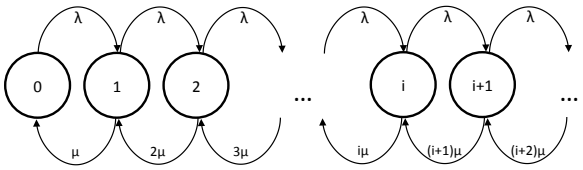

Figure 1: Markov chain of data loss due to independent node failures. Each state represents the number of nodes that are down simultaneously.

Number of Nodes	Pr(2 Failures)	Pr(3 Failures)	Pr(4 Failures)
1,000	6.51×10^{-7}	2.48×10^{-10}	7.07×10^{-14}
5,000	1.62×10^{-5}	3.08×10^{-8}	4.40×10^{-11}
10,000	6.44×10^{-5}	2.45×10^{-7}	7.00×10^{-10}
50,000	1.54×10^{-3}	2.93×10^{-5}	4.18×10^{-7}
100,000	5.81×10^{-3}	2.21×10^{-4}	6.31×10^{-6}

Table 1: Probability of simultaneous node failures due to independent node failures under different cluster sizes. The model uses $S = 10$, $R = 3$, $\tau = 60\ minutes$ and an average node MTTF of 10 years.

$scatter\ \ width$, or a linear function of the number of nodes that recover in parallel. $\mu = \frac{S}{\tau}$, where $\frac{\tau}{S}$ is the time to recover an entire server over the network with S nodes participating in the recovery. Typical values for $\frac{\tau}{S}$ are between 1-30 minutes [7, 14, 33]. For example, if a node stores 1 TB of data and has a scatter width of 10 (i.e., its data would be scattered across 10 nodes), and each node was read from at a rate of 500 MB/s, it would take about three minutes to recover the node's data. As a reference, Yahoo has reported that when the cluster recovers a node in parallel, it takes about two minutes to recover a node's data [33].

Throughout the paper we use $\tau = 60\ minutes$ with a scatter width of 10, which results in a recovery time of 6 minutes (three times higher than the recovery time reported by Yahoo [33]). Note that there is a practical lower bound to recovery time. Most systems first make sure the node has permanently failed before they start recovering the data. Therefore, we do not consider recovery times that are below one minute. We also assume that each node has a single 2 TB disk that can be recovered at a rate of 500 MB/s, and that each node's data is split into 10,000 chunks. These numbers match standard industry parameters [6, 9, 17].

The rate of data loss due to independent node failures is a function of two probabilities. The first is the probability that i nodes in the cluster have failed simultaneously at a given point in time: $Pr(i\ failed)$. The second is the probability of loss given i nodes failed simultaneously: $Pr(loss|i\ \ failed)$. In the next two subsections, we show how to compute these probabilities, and in the final subsection we show how to derive the overall rate of failure due to independent node failures.

2.1.1 Probability of i Nodes Failing

We first express $Pr(i\ \ failed)$ using a Continuous-time Markov chain, depicted in Figure 1. Each state in the Markov chain represents the number of failed nodes in a cluster at a given point in time.

The rate of transition between state i and $i + 1$ is the rate of independent node failures across the cluster, namely λ. The rate of the reverse transition between state i and $i - 1$ is the recovery rate of single node's data. Since there are i failed nodes, the recovery rate of a single node is $(i) \cdot \mu$ (in other words, as the number of nodes the cluster is trying to recover increases, the time it takes to recover the first node decreases, because more nodes participate in recovery). We assume that the number of failed nodes does not affect the rate of recovery. This assumption holds true as long as the number of failures is relatively small compared to the total number of nodes, which is true in the case of independent node failures in a large cluster (we demonstrate this below).

The probability of each state in a Markov chain with N states can always be derived from a set of N linear equations. However, since N is on the order of magnitude of 1,000 or more, and the number of simultaneous failures due to independent node failures in practical settings is very small compared to the number of nodes, we derived an approximate closed-form solution that assumes an infinite sized cluster. This solution is very simple to compute, and we provide the analysis for it in Appendix 8.

The probability of i nodes failing simultaneously is:

$$Pr(i\ failed) = \frac{\rho^i}{i!} e^{-\rho}$$

Where $\rho = \frac{\lambda}{\mu}$. The probabilities for different cluster sizes are depicted in Table 1. The results show that for clusters smaller than 10,000 nodes, the probability of two or more simultaneous independent failures is very low.

2.1.2 Data Loss Given i Node Failures

Now that we have estimated $Pr(i\ failed)$, we need to estimate $Pr(loss|i\ \ failed)$. Previous work has shown how to compute this probability for different types of replication techniques using simple combinatorics [9]. Replication algorithms map each chunk to a set of R nodes. A *copyset* is a set that stores all of the copies of a chunk. For example, if a chunk is replicated on nodes $\{7, 12, 15\}$, then these nodes form a copyset.

Random replication selects copysets randomly from the entire cluster. Facebook has implemented its own

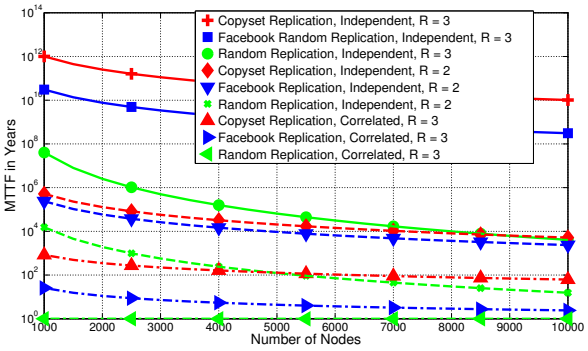

Figure 2: MTTF due to independent and correlated node failures of a cluster with a scatter width of 10.

random replication technique in which the R nodes are selected from a pre-designated window of nodes. For example, if the first replica is placed on node 10, the remaining two replicas will randomly be placed on two nodes out of a window of 10 subsequent nodes (i.e., they will be randomly selected from nodes $\{11, ..., 20\}$) [2, 9].

Unlike random schemes, Copyset Replication minimizes the number of copysets [9]. The following example demonstrates the difference between Copyset Replication and Facebook's scheme. Assume our storage system has: $R = 3$, $N = 9$ and $S = 4$. In Facebook's scheme, each chunk will be replicated on another node chosen randomly from a group of S nodes following the first node. E.g., if the primary replica is placed on node 1, the secondary replica will be randomly placed either on node 2, 3, 4 or 5. Therefore, if our system has a large number of chunks, it will create 54 distinct copysets.

In the case of a simultaneous failure of three nodes, the probability of data loss is the number of copysets divided by the maximum number of sets:

$$\frac{\#\ copysets}{\binom{N}{R}} = \frac{54}{\binom{9}{3}} = 0.64$$

Now, examine an alternative scheme using the same parameters. Assume we only allow our system to replicate its data on the following copysets:

$$\{1,2,3\},\{4,5,6\},\{7,8,9\}\{1,4,7\},\{2,5,8\},\{3,6,9\}$$

That is, if the primary replica is placed on node 3, the two secondary replicas can only be randomly placed on nodes 1 and 2 or 6 and 9. Note that with this scheme, each node's data will be split uniformly on four other nodes. The new scheme creates only 6 copysets. Now, if three nodes fail, the probability of data loss is:

$$\frac{\#\ copysets}{84} = \frac{6}{84} = 0.07.$$

Consequently, as we decrease the number of copysets, $Pr(loss|i\ failed)$ decreases. Therefore, this probability is significantly lower with Copyset Replication compared to Facebook's Random Replication.

Note however, that as we decrease the number of copysets, the frequency of data loss under correlated failures will decrease, but each correlated failure event will incur a higher number of lost chunks. This is a desirable tradeoff for many storage system designs, in which each data loss event incurs a fixed cost [9]. Another design choice that affects the number of copysets is the scatter width. As we increase the scatter width, the number of copysets used by the system also increases.

2.1.3 MTTF Due to Independent Node Failures

We can now compute the rate of loss due to independent node failures, which is:

$$Rate\ of\ Loss = \frac{1}{MTTF} = \lambda \sum_{i=1}^{N} Pr(i-1\ failed) \cdot (1 - Pr(loss|i-1\ failed)) \cdot Pr(loss|i\ failed)$$

The equation accounts for all events in which the Markov chain switches from state $i-1$, in which no loss occurs, to state i, in which data loss occurs. λ is the transition rate between state $i-1$ and i, $Pr(i-1\ failed)$ is the probability of state $i-1$, $(1-Pr(loss|i-1\ failed))$ is the probability that there was no data loss when $i-1$ nodes failed, and $Pr(loss|i\ failed)$ is the probability of data loss when i nodes failed. Since no data loss can occur when $i < R$, the sum can be computed from $i = R$.

In addition, Table 1 shows that under practical system parameters, the probability of i simultaneous node failures due to independent node failures drops dramatically as i increases. Therefore:

$$Rate\ of\ Loss = \frac{1}{MTTF} \approx \lambda \cdot Pr(R-1\ failed) \cdot Pr(loss|R\ failed)$$

Using this equation, Figure 2 depicts the MTTF of data loss under independent failures for $R = 2$ and $R = 3$ with three replication schemes, Random Replication, Facebook's Random Replication and Copyset Replication, as a function of the cluster's size. It is evident that Facebook's Random Replication and Copyset Replication have a much higher MTTF than Random Replication. The reason is that they use a much smaller number of copysets than Random Replication, and therefore their $Pr(loss|i\ failed)$ is smaller.

2.2 Analysis of Correlated Node Failures

Correlated failures occur when an infrastructure failure causes multiple nodes to be unavailable for a long period of time. Such failures include power outages that may affect an entire cluster, and network switch and

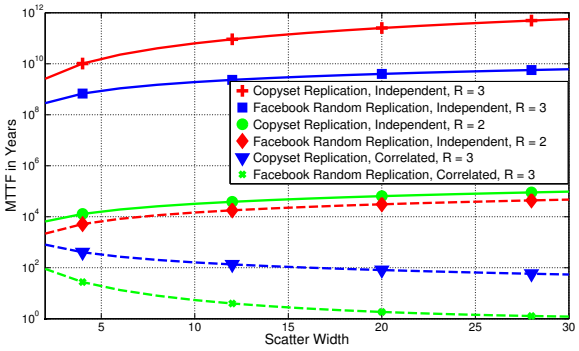

Figure 3: MTTF due to independent and correlated node failures of a cluster with 4000 nodes.

rack failures [7, 11]. Storage systems can avoid data loss related to some common correlated failure scenarios, by placing replicas on different racks or network segments [5, 7, 14]. However, these techniques only go so far to mitigate data loss, and storage systems still face unexpected simultaneous failures of nodes that share replicas. Such data loss events have been documented by multiple data center operators, such as Yahoo [33], LinkedIn [7] and Facebook [2, 5].

In order to analyze the affect of correlated failures on MTTF, we use the observation made by LinkedIn and Yahoo, that about once a year, 1% of the nodes do not recover after a cluster-wide power outage. This has been documented as the most common severe correlated failure [7, 33]. We can compute the probability of data loss for this event using combinatorics [9].

Figure 2 also presents the MTTF of data loss under correlated failures. It is evident from the graph that the MTTF due to correlated failures for $R = 3$ is three orders of magnitude lower than that for independent failures with $R = 2$ and six orders of magnitude lower than that for independent failures with $R = 3$, for any replication scheme.

We conclude that $R = 2$ is sufficient to protect against independent node failures, and that system designers should only focus on further increasing the MTTF under correlated failures, which is by far the main contributing factor to data loss. This has been corroborated in empirical studies conducted by Google [14] and LinkedIn [7].

This provides further evidence that random replication is very susceptible to correlated and independent failures. Therefore, in the rest of the paper we compare Tiered Replication only against Facebook's Random Replication and Copyset Replication.

Figure 3 plots the MTTF for correlated and independent node failures using the same model as before, as a function of the scatter width. This graph demonstrates that Copyset Replication provides a much higher MTTF than Facebook's Random Replication scheme. The figure also shows that increasing the scatter width has an opposite effect on MTTF for independent and correlated node failures. The MTTF due to independent node failures increases as a function of the scatter width, since a higher scatter width provides faster node recovery times, since more nodes participate in simultaneous recovery. In contrast, the MTTF due to correlated node failures decreases as a function of the scatter width, since a higher scatter width produces more copysets.

Since the MTTF is determined primarily by correlated failures, we can also conclude that if system designers wish to reduce the cluster-wide MTTF, they should use a small scatter width.

2.3 The Peculiar Case of the Nth (or Third) Replica

This analysis prompted us to investigate whether we can further increase the MTTF under correlated failures. We assume that the third replica was introduced in most cases to provide increased durability and not for increased read throughput [7, 8, 13, 34].

Therefore, consider a storage system in which the third replica is never read unless the first two replicas have failed. We estimate how frequently the system requires the use of a third replica, by analyzing the probability of data loss under independent node failures for a replication factor of two. If a system loses data when it uses two replicas, it means that if a third replica existed and did not fail, the system would recover the data from it.

In the independent failure model depicted by Figures 2 and 3, Facebook Random Replication and Copyset Replication require the third replica very rarely, on the order of magnitude of every 10^5 years.

To leverage this property, we can split our storage system into two tiers. The *primary tier* contains the first and second replicas of each chunk (or the N-1 replicas of each chunk), while the *backup tier* contains the third (or Nth) replica of each chunk. If possible, failures in the primary tier will always be recovered using nodes from the primary tier. We only recover from the backup tier if both the first and second replicas fail simultaneously. In case the storage system requires more than two nodes for read availability, the primary tier will contain the number of replicas required for availability, while the backup tier will contain an additional replica. Additional backup tiers (containing a single replica) can be added to support read availability in multiple geographies.

Therefore, the backup tier will be mainly used for durability during severe correlated failures, which are infrequent (on the order of once a year), as reported by various operators [5, 7, 33]. Consequently, the backup tier can be viewed as write-dominated storage, since it is written to on every write (e.g., thousands of times per second), but only read from a few times a year.

Splitting the cluster into tiers provides multiple advantages. The storage system designer can significantly reduce the correlation between failures in the primary tier and the backup tier similar to full cluster geo-replication. This can be achieved by storing the backup tier in a geographically remote location, or by other means of physical separation such as using different network and power infrastructure. It has been shown by Google that storing data in a physical remote location significantly reduces the correlation between failures across the two sites [14].

Another possible advantage is that the backup tier can be stored more cost-effectively than the primary tier, since it does not require low read latency. For example, the backup tier can be stored on a cheaper storage medium (e.g., tape, or disk in the case of an SSD-based cluster), its data may be compressed [17, 19, 22, 28, 30], deduplicated [12, 27, 40] or may be configured in other ways to be optimized for a write dominated workload.

The idea of using full cluster geo-replication has been explored extensively. However, existing geo-replication techniques replicate all replicas from one cluster to a second cluster, which multiplies the cost of storage [14, 23].

In the next section, we design a replication technique, Tiered Replication, that supports tiered clusters and does not duplicate the entire cluster. Unlike random replication, Tiered Replication is not susceptible to correlated node failures, and unlike previous non-random techniques like Copyset Replication, it supports data topology constraints such as tiered replicas and minimizes the number of copysets, even when the number of nodes in the cluster changes over time [9].

3. DESIGN

The goal of Tiered Replication is to create copysets (groups of nodes that contain all copies of a single chunk). When a node replicates its data, it will randomly choose a copyset that it is a member of, and place the replicas of the chunk on all the nodes in its copyset. Tiered Replication attempts to minimize the number of copysets while providing sufficient scatter width (i.e., node recovery bandwidth), and ensuring that each copyset contains a single node from the backup tier. Tiered Replication also flexibly accommodates any additional constraints defined by the storage system designer (e.g., split copysets across racks or network tiers).

Algorithm 1 describes Tiered Replication, while Table 2 contains the definitions used in the algorithm. Tiered Replication continuously creates new copysets until all nodes are replicated with sufficient scatter width. Each copyset is formed by iteratively picking candidate nodes with a minimal scatter width that meet the constraints of the nodes that are already in the copyset. Algorithm 2 describes the part of the algorithm that checks whether the copyset has met the constraints. The first constraint satisfies the tier requirements, i.e., having exactly one node in each copyset that belongs to the backup tier. The second constraint enforces the minimization of the number of copysets, by requiring that the nodes in the new copyset do not appear with each other in previous copysets. This constraint minimizes the number of copysets, because each new copyset contributes the maximum increase of scatter width.

Name	Description
cluster	list of all the nodes in the cluster
node	the state of a single node
R	replication factor (e.g., 3)
cluster.S	desired minimum scatter width of all the nodes in the cluster
node.S	the current scatter width of a node
cluster.sort	returns a sorted list of the nodes in increasing order of scatter width
cluster.addCopyset(copyset)	adds copyset to the list of copysets
cluster.checkTier(copyset)	returns false if there is more than one node from the backup tier, or R nodes from the primary tier
cluster.didNotAppear(copyset)	returns true if each node never appeared with other nodes in previous copysets

Table 2: Tiered Replication algorithm's variables and helper functions.

Algorithm 1 Tiered Replication

```
1:  while ∃ node ∈ cluster s.t. NODE.S < CLUSTER.S do
2:      for all node ∈ cluster do
3:          if NODE.S < CLUSTER.S then
4:              copyset = {node}
5:              sorted = CLUSTER.SORT
6:              for all sortedNode ∈ sorted do
7:                  copyset = copyset ∪ {sortedNode}
8:                  if CLUSTER.CHECK(copyset) == false then
9:                      copyset = copyset - {sortedNode}
10:                 else if COPYSET.SIZE == R then
11:                     CLUSTER.ADDCOPYSET(copyset)
12:                     break
13:                 end if
14:             end for
15:         end if
16:     end for
17: end while
```

Algorithm 2 Check Constraints Function

```
1: function CLUSTER.CHECK(copyset)
2:     if CLUSTER.CHECKTIER(copyset) == true AND
           CLUSTER.DIDNOTAPPEAR(copyset) AND
           ... // additional data layout constraints then
3:         return true
4:     else
5:         return false
6:     end if
7: end function
```

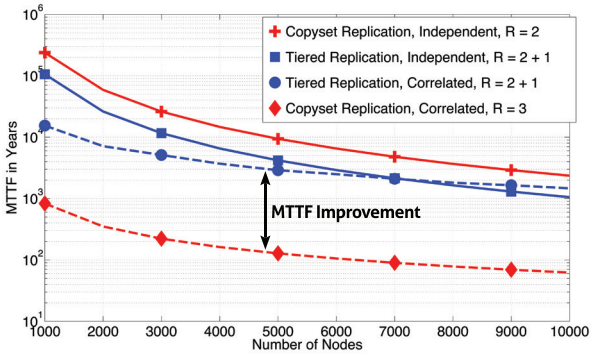

Figure 4: MTTF improvement of Tiered Replication with a scatter width of 10.

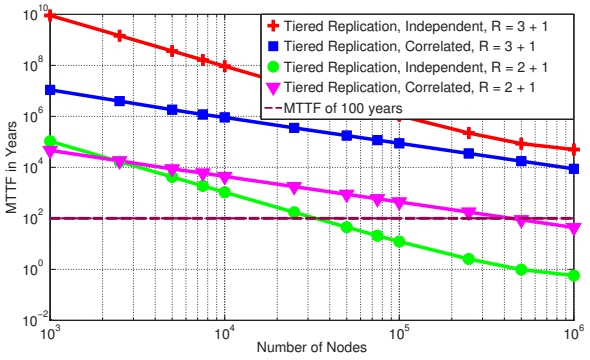

Figure 5: Tiered Replication would require 4 replicas to maintain an MTTF of 100 years once the cluster scales to 25,000 nodes with a scatter width of 10.

Note that there may be cases in which the algorithm does not succeed to find a copyset that satisfies all the constraints. In this case, the algorithm is run again, with a relaxed set of constraints (e.g., we can relax the constraint of minimizing the number of copysets, and allow more overlap between copysets). In practical scenarios, in which the number of nodes is an order of magnitude or more greater than S, the algorithm will easily satisfy all constraints.

3.1 Analysis of Tiered Replication

We evaluate the durability of Tiered Replication under independent and correlated node failures. To measure the MTTF under independent node failures, we use the same Continuous-time Markov model that we presented in Section 2. The results are presented in Figures 4 and 5. Note that $R = 2 + 1$ means we use Tiered Replication with two replicas in the primary tier and one replica in the backup tier.

Under Tiered Replication, when a replica fails in the primary tier, if possible, it is only recovered from other nodes in the primary tier. Therefore, fewer nodes will participate in recovery, because the backup tier nodes will not be recovered from. In order to compensate for this effect, system designers that use Tiered Replication may choose to increase the scatter width. For our analysis we compute the MTTF using the same scatter width for Tiered Replication and other replication schemes. Figure 4 shows that for $S = 10$, the MTTF under independent node failures is higher for Copyset Replication compared to Tiered Replication, because fewer nodes participate in the recovery of primary replicas and its single-node recovery time is therefore higher.

Also, note that in Figures 4 and 5, we assume that for $R = 2+1$, the third replica is never used to recover from independent node failures. In reality, the backup tier is used for any failure of two nodes from the primary tier, and therefore will be used in the rare case of an independent node failure that simultaneously affects two nodes in the primary tier that are in the same copyset. Hence, the MTTF under independent node failures for Tiered Replication is even higher than depicted by the graphs.

To evaluate the durability of Tiered Replication under correlated failures, we quantify the probability that all the nodes in one copyset or more fail. Since the primary and backup tiers are stored on separate infrastructure, we assume that their failures are independent.

Since each copyset includes two nodes from the primary tier, when these nodes fail simultaneously, data loss will occur only if the third copyset node from the backup tier failed at the same time. Since our assumption is that correlated failures occur once a year and affect 1% of the nodes each time (i.e., an MTTF of 100 years for a single node), while independent failures occur once in every 10 years for a node, it is 10 times more likely that if a backup node fails, it is due to an independent node failure. Therefore, the dominant cause of failures for Tiered Replication is when a correlated failure occurs in the primary tier, and at the same time an independent node failure occured in the backup tier.

To compute the MTTF due to this scenario, we need to compute the probability that a node failure will occur in the backup cluster while a correlated failure event is occurring in the primary cluster. To be on the conservative side, we assume that it takes 12 hours to fully recover the data after the correlated failure in the primary tier (LinkedIn data center operators report that unavailability events typically take 1-3 hours to recover from [7]). We compute the probability of data loss in this scenario, using the same combinatorial methods that we used to compute the MTTF under correlated failures before.

Figure 4 shows that the MTTF of Tiered Replication is more than two orders of magnitude greater than Copyset Replication. This is due to the fact that it is much less likely to lose data under correlated failures when one of the replicas is stored on an independent cluster. Recall

that Copyset Replication's MTTF was already three orders of magnitude greater than random replication.

In Figure 5 we explore the following: what is the turning point, when a storage system needs to use $R = 4$ instead of $R = 3$? We plot the MTTF of Tiered Replication and extend it $N = 1,000,000$, which is a much larger number of nodes than is used in today's clusters. Assuming that storage designers are targeting an MTTF of at least 100 years, our results show that at around 25,000 nodes, storage systems should switch to a default of $R = 4$. Note that Figure 4 shows that Copyset Replication needs to switch to $R = 4$ much sooner, at about 5,000 nodes. Other replication schemes, like Facebook's scheme, fail to achieve an MTTF of 100 years with $R = 3$, even for very small clusters.

3.2 Dynamic Cluster Changes

Since running Tiered Replication is fast to execute (on the order of milliseconds, see Section 4) and the algorithm is structured to create new copysets incrementally, the storage system can run it every time the cluster changes its configuration.

When a new node joins the cluster, we simply run Tiered Replication again. Since the new node does not belong to any copysets, it starts with a scatter width of 0. Tiered Replication's greedy operation ensures that the node is assigned to a sufficient number of copysets that will increase its scatter width to the value of S.

When a node dies (or leaves the cluster), it leaves behind copysets that are missing a single node. The simplest way to re-instate the copysets is to assume that the old copysets are down and run the algorithm again. The removal of these copysets will reduce the scatter width of the nodes that were contained in the removed copysets, and the algorithm will create a new set of copysets to replace the old ones. The data in the old copysets will need to be re-replicated R times again. We chose this approach when implementing Tiered Replication on HyperDex, due to its simplicity.

Alternatively, the algorithm can be optimized to look for a replacement node, which addresses the constraints of the remaining nodes in the copyset. In this scenario, if the algorithm succeeds in finding a replacement, the data will be re-replicated only once.

3.3 Additional Constraints

Tiered Replication can be extended to support different requirements of storage system designers by adding more constraints to the $cluster.check(copysets)$ function. The following provides two examples.

Controlled Power Down: Some storage designers would like to allow parts of the cluster to be temporarily switched off to reduce power consumption (e.g., according to diurnal patterns). For example, Sierra [36], allows a cluster with three replicas to power down two thirds of its cluster and still allow access to data. This feature can easily be added as a constraint to Tiered Replication by forcing each copyset to contain a node that belongs to a different tier. This feature is also important for supporting controlled software or hardware upgrades, during which parts of the cluster may be powered down without affecting the cluster availability.

Chain Replication: Chain replication can provide improved performance and consistency. Each replica is assigned a position in the chain (e.g., head, middle, tail) [37]. A desirable property of chain replication is that each node will have an equal number of replicas in each position. It is straightforward to incorporate this requirement into Tiered Replication. In order to ensure that nodes have an even distribution of chain positions for their replicas, when the algorithm assigns nodes to copysets and chain positions, it tries to balance the number of times the node will appear in each chain position. For example: if a node has been assigned to the head position twice, middle position twice and tail position once, the algorithm will enforce that it will be assigned to a tail position in the next copyset the node will belong to.

To demonstrate the ability to incorporate additional constraints to Tiered Replication, we implemented it on HyperDex [13], a storage system that uses chain replication. Note that Copyset Replication and Random Replication are inefficient for supporting balanced chain sets. Copyset Replication is not designed for incorporating such constraints because it randomly permutes the entire set of nodes. Random Replication is not effective for this requirement because its random placement of nodes frequently creates imbalanced chain positions.

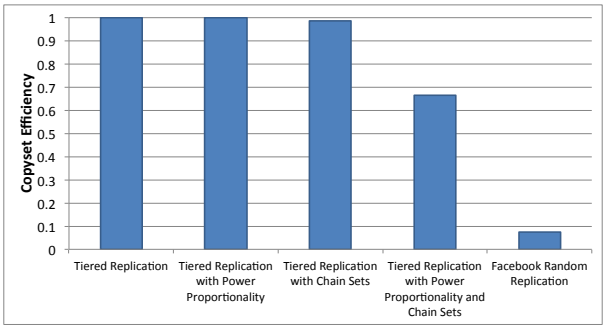

Figure 6: Adding constraints on Tiered Replication increases the number of copysets, on a cluster with $N = 4000$, $R = 3$ and $S = 10$.

3.4 Analysis of Additional Constraints

Figure 6 demonstrates the effect of adding constraints on the number of copysets. In the figure, *copyset effi-*

ciency is equal to the ratio between the number of copysets generated by an optimal replication scheme that minimizes the number of copysets, and the number of copysets generated by the replication scheme with the constraint. Note that in an optimal replication scheme, nodes will not appear more than once with each other in different copysets, or in other words, the number of copysets would be equal to $S \cdot (R - 1)$.

The graph shows that as we further constrain Tiered Replication, it is less likely to generate copysets that meet multiple constraints, and its copyset efficiency will decrease. The figure also shows that the chain set constraint has a greater impact on the number of copysets than the power down constraint. In any case, Tiered Replication with additional constraints significantly outperforms any Random Replication scheme.

4. IMPLEMENTATION

In order to evaluate Tiered Replication in a practical setting, we have extended the open-source HyperDex [13] key-value store to use Tiered Replication for replica set selection. HyperDex is a distributed, fault-tolerant, and strongly consistent key-value store. For our purposes, HyperDex's architecture is especially suited for evaluating different replication strategies. In HyperDex, a replicated state machine serves as the coordinator, and is responsible for maintaining the cluster membership and metadata. Part of this metadata includes the complete specification of all replica sets in the system.

Since Tiered Replication can be implemented efficiently, and is only run when nodes join or leave the cluster, we implement the greedy algorithm directly in the replicated HyperDex coordinator.

In HyperDex, a variant of chain replication [37] called value-dependent chaining specifies the replica set of an object as a chain of nodes through which writes propagate. We incorporate this into Tiered Replication in the form of additional constraints that track the positions of nodes within chains as well as the replica sets each node appears in. This helps ensure that each node will be balanced across different positions in the chain.

We implement Tiered Replication using synchronous replication. All writes will be acknowledged by all replicas in both tiers before the client is notified of success. Practically, this means writes will not be lost except in the case of correlated failure of all replicas in both tiers.

In total, our changes to HyperDex are minimal. We added or changed about 600 lines of C++ code, of which 250 lines constitute the greedy Tiered Replication algorithm. The rest of our implementation provides the administrator with tools to denote whether a node belongs to the primary or the backup tier of the cluster.

5. EVALUATION

In order to measure the performance impact of Tiered Replication in a practical setting, we do not attempt to measure the frequency of data loss under realistic scenarios, because it is impractical to run a cluster of thousands of nodes for decades.

5.1 Performance Benchmarks

We set up a 9 node HyperDex cluster on Amazon EC2 using M3 xlarge instances. Each node has a high frequency Intel Xeon E5-2670 v2 (Ivy Bridge) with 15 GiB of main memory and two 40 GB SSD volumes configured to store HyperDex data.

We compare Tiered Replication to HyperDex's default replication scheme, which does not support smart placement across two different availability zones. We ran 6 nodes in one availability zone (us-east-1a) and the three remaining nodes in a second availability zone (us-east-1b). In Amazon EC2, each availability zone runs on its own physically distinct, independent infrastructure. Common points of failures like generators and cooling equipment are not shared across availability zones. Additionally, they are physically separate, such that even extremely uncommon disasters such as fires, tornados or flooding would only affect a single availability zone [1].

In both deployments, the cluster is physically split across the availability zones, but only the tiered replication scheme ensures that there is exactly one node from the backup tier in each chain. In both deployments, we measured the throughput and latency of one million requests with the Yahoo! Cloud Serving Benchmark [10]. YCSB is an ideal choice of benchmark, because it has become the de-facto standard for benchmarking key-value stores, and provides a variety of workloads drawn from real workloads in place at Yahoo. We configured YCSB to run 32 client threads per host on each of the hosts in the cluster, with the database prepopulated with 10 million 1KiB objects. Figure 7 shows the throughput measured for both deployments. The difference in throughput is due to the fact that Tiered Replication does not load balance the nodes evenly in very small clusters. The figure includes error bars for the observed throughput over any one second interval through the course of the experiment.

5.2 Write Latency

We measure the write latency overhead of Tiered Replication. The workload consists of 50% read operations and 50% write operations. When we compared the write latency of Tiered Replication across two availability zones with Random Replication across two zones, we did not find any difference in latency.

Figure 8 comares the write latency of Tiered Replication across two zones with Random Replication across a single zone. As expected, Tiered Replication adds some

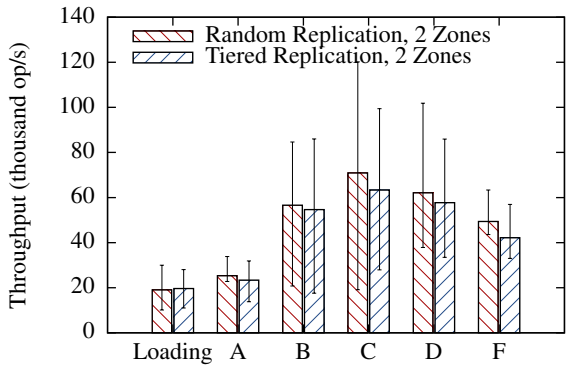

Figure 7: Tiered Replication throughput under YCSB benchmark. Each bar represents the throughput under a different YCSB workload.

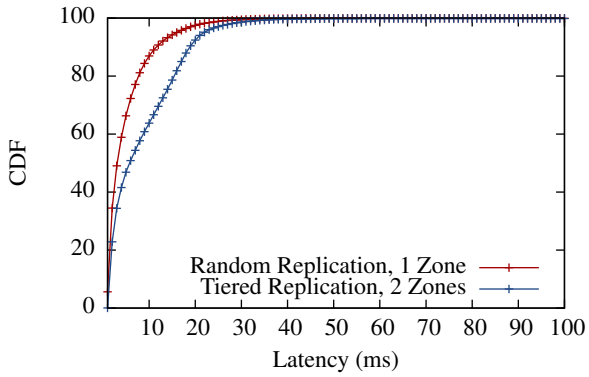

Figure 8: Comparison of write latency between Tiered Replication across two availability zones and Random Replication on a single availability zone under YCSB benchmark.

latency overhead, because it spreads the requests across two availability zones, and our implementation uses synchronous replication, which requires waiting for all replicas to be written before acknowledging the write.

5.3 Recovery Evaluation

We measured single node recovery times under two scenarios: a node failure in the primary tier and in the backup tier. We repeated the experiment from the first two benchmarks and ran YCSB workload B. Thirty seconds into the benchmark, we forcibly failed one node in the cluster, and initiated recovery thirty seconds later.

In the primary tier node failure experiment, it took approximately 80 seconds to recover the failed node. It then took another 48 seconds for the recovered node to synchronize missing data with nodes in the backup tier. This two-step reintegration process is required by chain replication: in the first step, the node performs state transfer with its predecessor, while in the second step, the node verifies its data with its successor. The recovered node only transfers to the backup tier nodes the data written during the thirty seconds of down time, and not the total data set.

When we repeated the same experiment and failed a node in the backup tier, it took 91 seconds to recover the node. Because this node is the last in the chain, the node can completely reintegrate with the cluster in one step. The recovery time for both of these experiments is similar to that of the default HyperDex replication algorithm.

5.4 Bandwidth Across Availability Zones

Tiered Replication relies on having two failure-independent locations, with a high bandwidth interconnect between them. Our testing shows that Amazon's availability zones are an ideal setup. They provide reliable links with high bandwidth and low latency. Using the same machines from the previous experiments, we conducted pairwise bandwidth tests using the iperf performance measuring tool. All servers in the cluster were able to communicate at 1.09Gbit/s (even when communicating across availability zones), which seems to be the maximum capability of the instances we purchased.

6. RELATED WORK

Several researchers have made observations that the MTTF under independent failures is much higher than from correlated failures [25, 38]. A LinkedIn field study reported no record of data loss due to independent node failures [7]. Google researchers have shown that the MTTF with three replicas under correlated failures is almost three orders of magnitude lower than the MTTF under independent node failures [14].

Google researchers developed an analysis based on a Markov model that computes the MTTF for a single stripe under independent and correlated failures. However, they did not provide an analysis for the MTTF of the entire cluster [14]. Nath et al. modeled the affect of correlated node failures and demonstrated that replication techniques that prevent data loss under independent node failures are not always effective for preventing correlated node failures [25]. In addition, several researchers have modeled the MTTF for individual device components, and in particular for disks [3, 18, 26, 31, 32].

Several replication schemes addressed the high probability of data loss under correlated failures. Facebook's HDFS implementation [2, 5] limits the scatter width of Random Replication, in order to reduce the probability of data loss under correlated failures. Copyset Replication [9] improved Facebook's scheme, by restricting the replication to a minimal number of copysets for a given scatter width. Tiered Replication is the first repli-

cation technique that not only minimizes the probability of data loss under correlated failures, but also leverages the much higher MTTF under independent failures to further increase the MTTF under correlated failures. In addition, unlike Copyset Replication, Tiered Replication can gracefully tolerate dynamic cluster changes, such as nodes joining and leaving and planned cluster power downs. It also supports chain replication and the ability to distribute replicas to different racks and failure domains, which is a desirable requirement of replication schemes [5, 24].

The common way to increase durability under correlated failures is to use geo-replication of entire cluster to a remote site [14, 21, 23, 35, 39]. Therefore, if the cluster was using three replicas, once it is geo-replicated, the storage provider will effectively use six replicas. Similarly, Glacier [16] and Oceanstore [20, 29] design an archival storage layer that provides extra protection against correlated failures by adding multiple new replicas to the storage system. While the idea of using archival replicas is not new, Tiered Replication is more cost-efficient, since does not require any additional storage for the backup: it migrates one replica from the original cluster to a backup tier. In addition, previous replication techniques utilize random placement schemes and do not minimize the number of copysets, which leaves them susceptible to correlated failures.

Storage coding is used for reducing the storage overhead of replication [17, 19, 22, 28, 30]. De-duplication is also commonly used to reduce the overhead of redundant copies of data [12, 27, 40]. Tiered Replication is fully compatible with any coding or de-duplication schemes for further reduction of storage costs of the backup tier. Moreover, Tiered Replication enables storage systems to further reduce costs by storing the third replicas of their data on a cheap storage medium such as tape, or hard disks in the case of an solid-state based storage cluster.

7. CONCLUSION

Cloud storage systems typically rely on three-way replication within a cluster to protect against independent node failures, and on full geo-replication of an entire cluster to protect against correlated failures. We provided an analytical framework for computing the probability of data loss under independent and correlated node failures, and demonstrated that the standard replication architecture used by cloud storage systems is not cost-effective. Three-way replication is excessive for protecting against independent node failures, and clearly falls short of protecting storage systems from correlated node failures. The key insight of our paper is that since the third replica is rarely needed for recovery from independent node failures, it can be placed on a geographically separated cluster, without causing a significant impact to the recovery time from independent node failures, which occur frequently in large clusters.

We presented Tiered Replication, a replication technique that automatically places the n-th replica on a separate cluster, while minimizing the probability of data loss under correlated failures, by minimizing the number of copysets. Tiered Replication improves the cluster-wide MTTF by a factor of 20,000 compared to random replication, without increasing the storage capacity. Tiered Replication supports additional data placement constraints required by the storage designer, such as rack awareness and chain replication assignments, and can dynamically adapt when nodes join and leave the cluster. An implementation of Tiered Replication on HyperDex, a key-value storage system, demonstrates that it incurs a small performance overhead.

8. APPENDIX

This section contains the closed-form solution for the Markov chain described in Section 2 and Figure 1 with an infinite number of nodes. The state transitions the Continuous-time Markov chain state i are:

$$i \cdot \mu \cdot Pr(i) = \lambda \cdot Pr(i-1)$$

Therefore:

$$Pr(i) = \frac{\rho}{i} Pr(i-1)$$

Where $\rho = \frac{\lambda}{\mu}$. If we apply this formula recursively:

$$Pr(i) = \frac{\rho}{i} Pr(i-1) = \frac{\rho^2}{i \cdot (i-1)} Pr(i-2) = \frac{\rho^i}{i!} Pr(0)$$

In order to find $Pr(0)$, we use the fact that the sum of all the Markov state probabilities is equal to 1:

$$\sum_{i=0}^{\infty} Pr(i) = 1$$

If we apply the recursive formula:

$$\sum_{i=0}^{\infty} Pr(i) = \sum_{i=0}^{\infty} \frac{\rho^i}{i!} Pr(0) = 1$$

Using the equality $\sum_{i=0}^{\infty} \frac{\rho^i}{i!} = e^{\rho}$, we get: $Pr(0) = e^{-\rho}$. Therefore, we now have a simple closed-form formula for all of the Markov state probabilities:

$$Pr(i) = \frac{\rho^i}{i!} e^{-\rho}$$

References

[1] How isolated are availability zones from one another? http://aws.amazon.com/ec2/faqs/#How_isolated_are_Availability_Zones_from_one_another.

[2] Intelligent block placement policy to decrease probability of data loss. https://issues.apache.org/jira/browse/HDFS-1094.

[3] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An analysis of latent sector errors in disk drives. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07, pages 289–300, New York, NY, USA, 2007. ACM.

[4] M. Bakkaloglu, J. J. Wylie, C. Wang, and G. R. Ganger. On correlated failures in survivable storage systems. Technical report, DTIC Document, 2002.

[5] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer. Apache Hadoop goes realtime at Facebook. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 1071–1080, New York, NY, USA, 2011. ACM.

[6] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. Windows Azure Storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 143–157, New York, NY, USA, 2011. ACM.

[7] R. J. Chansler. Data Availability and Durability with the Hadoop Distributed File System. *;login: The USENIX Magazine*, 37(1), February 2012.

[8] 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. *NSDI*, 6:4–4, 2006.

[9] A. Cidon, S. M. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum. Copysets: Reducing the frequency of data loss in cloud storage. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, USENIX ATC'13, pages 37–48, Berkeley, CA, USA, 2013. USENIX Association.

[10] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[11] J. Dean. Evolution and future directions of large-scale storage and computation systems at Google. In *SoCC*, page 1, 2010.

[12] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki. HYDRAstor: A scalable secondary storage. In *FAST*, volume 9, pages 197–210, 2009.

[13] R. Escriva, B. Wong, and E. G. Sirer. HyperDex: A distributed, searchable key-value store. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 25–36, New York, NY, USA, 2012. ACM.

[14] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–7, Berkeley, CA, USA, 2010. USENIX Association.

[15] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *SOSP*, pages 29–43, 2003.

[16] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: highly durable, decentralized storage despite massive correlated failures. In *IN PROC. OF NSDI*, 2005.

[17] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in Windows Azure Storage. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[18] W. Jiang, C. Hu, Y. Zhou, and A. Kanevsky. Are disks the dominant contributor for storage failures?: A comprehensive study of storage subsystem failure characteristics. *Trans. Storage*, 4(3):7:1–7:25, Nov. 2008.

[19] R. Kotla, L. Alvisi, and M. Dahlin. SafeStore: a durable and practical storage system. In *USENIX Annual Technical Conference*, pages 129–142, 2007.

[20] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, et al. Oceanstore: an architecture for global-scale persistent storage. *ACM Sigplan Notices*, 35(11):190–201, 2000.

[21] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 265–278, Berkeley, CA, USA, 2012. USENIX Association.

[22] R. Li, P. P. Lee, and Y. Hu. Degraded-first scheduling for MapReduce in erasure-coded storage clusters.

[23] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Stronger semantics for low-latency geo-replicated storage. In *Symposium on Networked Systems Design and Implementation*, 2013.

[24] J. MacCormick, N. Murphy, V. Ramasubramanian, U. Wieder, J. Yang, and L. Zhou. Kinesis: A new approach to replica placement in distributed storage systems. *ACM Transactions On Storage (TOS)*, 4(4):11, 2009.

[25] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. In *NSDI*, volume 6, pages 225–238, 2006.

[26] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *5th USENIX Conference on File and Storage Technologies (FAST 2007)*, pages 17–29, 2007.

[27] S. Quinlan and S. Dorward. Venti: A new approach to archival data storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[28] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster. In *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*. USENIX, 2013.

[29] S. C. Rhea, P. R. Eaton, D. Geels, H. Weatherspoon, B. Y. Zhao, and J. Kubiatowicz. Pond: The oceanstore prototype. In *FAST*, volume 3, pages 1–14, 2003.

[30] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing elephants: novel erasure codes for big data. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 325–336. VLDB Endowment, 2013.

[31] B. Schroeder and G. A. Gibson. Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07, Berkeley, CA, USA, 2007. USENIX Association.

[32] B. Schroeder, E. Pinheiro, and W.-D. Weber. DRAM errors in the wild: A large-scale field study. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 193–204, New York, NY, USA, 2009. ACM.

[33] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. *Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference on*, 0:1–10, 2010.

[34] E. Sit, A. Haeberlen, F. Dabek, B.-G. Chun, H. Weatherspoon, R. Morris, M. F. Kaashoek, and J. Kubiatowicz. Proactive replication for data durability. In *IPTPS*, 2006.

[35] Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 385–400. ACM, 2011.

[36] E. Thereska, A. Donnelly, and D. Narayanan. Sierra: practical power-proportionality for data center storage. *Proceedings of Eurosys 11*, pages 169–182, 2011.

[37] R. van Renesse and F. B. Schneider. Chain replication for supporting high throughput and availability. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 7–7, Berkeley, CA, USA, 2004. USENIX Association.

[38] 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 *USENIX WORLDS*, 2004.

[39] Y. Zhang, R. Power, S. Zhou, Y. Sovran, M. K. Aguilera, and J. Li. Transaction chains: achieving serializability with low latency in geo-distributed storage systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 276–291. ACM, 2013.

[40] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the Data Domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 18:1–18:14, Berkeley, CA, USA, 2008. USENIX Association.