Copysets: Reducing the Frequency of Data Loss in Cloud Storage

Asaf Cidon, Stephen Rumble, Ryan Stutsman,
Sachin Katti, John Ousterhout and Mendel Rosenblum
Stanford University
cidon@stanford.edu, {rumble,stutsman,skatti,ouster,mendel}@cs.stanford.edu

ABSTRACT

Random replication is widely used in data center storage systems to prevent data loss. However, random replication is almost guaranteed to lose data in the common scenario of simultaneous node failures due to cluster-wide power outages. Due to the high fixed cost of each incident of data loss, many data center operators prefer to minimize the frequency of such events at the expense of losing more data in each event.

We present Copyset Replication, a novel general-purpose replication technique that significantly reduces the frequency of data loss events. We implemented and evaluated Copyset Replication on two open source data center storage systems, HDFS and RAMCloud, and show it incurs a low overhead on all operations. Such systems require that each node's data be scattered across several nodes for parallel data recovery and access. Copyset Replication presents a near optimal trade-off between the number of nodes on which the data is scattered and the probability of data loss. For example, in a 5000-node RAMCloud cluster under a power outage, Copyset Replication reduces the probability of data loss from 99.99% to 0.15%. For Facebook's HDFS cluster, it reduces the probability from 22.8% to 0.78%.

1. INTRODUCTION

Random replication is used as a common technique by data center storage systems, such as Hadoop Distributed File System (HDFS) [25], RAMCloud [24], Google File System (GFS) [14] and Windows Azure [6] to ensure durability and availability. These systems partition their data into chunks that are replicated several times (we use R to denote the replication factor) on randomly selected nodes on different racks. When a node fails, its data is restored by reading its chunks from their replicated copies.

However, large-scale correlated failures such as cluster power outages, a common type of data center failure scenario [7, 10, 13, 25], are handled poorly by random replication. This scenario stresses the availability of the system because a non-negligible percentage of nodes (0.5%-1%) [7, 25] do not come back to life after power has been restored. When a large number of nodes do not power up there is a high probability that all replicas of at least one chunk in the system will not be available.

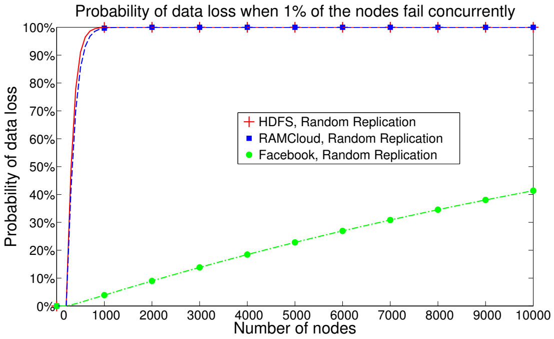

Figure 1: Computed probability of data loss with $R = 3$ when 1% of the nodes do not survive a power outage. The parameters are based on publicly available sources [5, 7, 24, 25] (see Table 1).

Figure 1 shows that once the size of the cluster scales beyond 300 nodes, this scenario is nearly guaranteed to cause a data loss event in some of these systems. Such data loss events have been documented in practice by Yahoo! [25], LinkedIn [7] and Facebook [5]. Each event reportedly incurs a high fixed cost that is not proportional to the amount of data lost. This cost is due to the time it takes to locate the unavailable chunks in backup or recompute the data set that contains these chunks. In the words of Kannan Muthukkaruppan, Tech Lead of Facebook's HBase engineering team: "Even losing a single block of data incurs a high fixed cost, due to the overhead of locating and recovering the unavailable data. Therefore, given a fixed amount of unavailable data each year, it is much better to have fewer incidents of data loss with more data each than more incidents with less data. We would like to optimize for minimizing the probability of incurring *any* data loss" [22]. Other data center operators have reported similar experiences [8].

Another point of view about this trade-off was expressed by Luiz André Barroso, Google Fellow: "Having a framework that allows a storage system provider to manage the profile of frequency vs. size of data losses is very useful, as different systems prefer different policies. For example, some providers might prefer frequent, small losses since they are less likely to tax storage nodes and fabric with spikes in data reconstruction traffic. Others services may not work well when even a small fraction of the data is unavailable. Those will prefer to have all or nothing, and would opt for fewer events even if they come at a larger loss penalty." [3]

Random replication sits on one end of the trade-off between the frequency of data loss events and the amount lost at each event. In this paper we introduce Copyset Replication, an alternative general-purpose replication scheme with the same performance of random replication, which sits at the other end of the spectrum.

Copyset Replication splits the nodes into $copysets$, which are sets of R nodes. The replicas of a single chunk can only be stored on one copyset. This means that data loss events occur only when all the nodes of some copyset fail simultaneously.

The probability of data loss is minimized when each node is a member of exactly one copyset. For example, assume our system has 9 nodes with $R = 3$ that are split into three copysets: $\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9\}$. Our system would only lose data if nodes 1, 2 and 3, nodes 4, 5 and 6 or nodes 7, 8 and 9 fail simultaneously.

In contrast, with random replication and a sufficient number of chunks, any combination of 3 nodes would be a copyset, and any combination of 3 nodes that fail simultaneously would cause data loss.

The scheme above provides the lowest possible probability of data loss under correlated failures, at the expense of the largest amount of data loss per event. However, the copyset selection above constrains the replication of every chunk to a single copyset, and therefore impacts other operational parameters of the system. Notably, when a single node fails there are only $R - 1$ other nodes that contain its data. For certain systems (like HDFS), this limits the node's recovery time, because there are only $R - 1$ other nodes that can be used to restore the lost chunks. This can also create a high load on a small number of nodes.

To this end, we define the $scatter\ width$ (S) as the number of nodes that store copies for each node's data.

Using a low scatter width may slow recovery time from independent node failures, while using a high scatter width increases the frequency of data loss from correlated failures. In the 9-node system example above, the following copyset construction will yield $S = 4$: $\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9\}, \{1, 4, 7\}, \{2, 5, 8\}, \{3, 6, 9\}$. In this example, chunks of node 5 would be replicated either at nodes 4 and 6, or nodes 2 and 8. The increased scatter width created more copyset failure opportunities.

The goal of Copyset Replication is to minimize the probability of data loss, *given any scatter width* by using the smallest number of copysets. We demonstrate that Copyset Replication provides a near optimal solution to this problem. We also show that this problem has been partly explored in a different context in the field of combinatorial design theory, which was originally used to design agricultural experiments [26].

Copyset Replication transforms the profile of data loss events: assuming a power outage occurs once a year, it would take on average a 5000-node RAMCloud cluster 625 years to lose data. The system would lose an average of 64 GB (an entire server's worth of data) in this rare event. With random replication, data loss events occur frequently (during every power failure), and several chunks of data are lost in each event. For example, a 5000-node RAMCloud cluster would lose about 344 MB in each power outage.

To demonstrate the general applicability of Copyset Replication, we implemented it on two open source data center storage systems: HDFS and RAMCloud. We show that Copyset Replication incurs a low overhead on both systems. It reduces the probability of data loss in RAMCloud from 99.99% to 0.15%. In addition, Copyset Replication with 3 replicas achieves a lower data loss probability than the random replication scheme does with 5 replicas. For Facebook's HDFS deployment, Copyset Replication reduces the probability of data loss from 22.8% to 0.78%.

The paper is split into the following sections. Section 2 presents the problem. Section 3 provides the intuition for our solution. Section 4 discusses the design of Copyset Replication. Section 5 provides details on the implementation of Copyset Replication in HDFS and RAMCloud and its performance overhead. Additional applications of Copyset Replication are presented in in Section 6, while Section 7 analyzes related work.

2. THE PROBLEM

In this section we examine the replication schemes of three data center storage systems (RAMCloud, the default HDFS and Facebook's HDFS), and analyze their vulnerability to data loss under correlated failures.

2.1 Definitions

The replication schemes of these systems are defined by several parameters. R is defined as the number of replicas of each chunk. The default value of R is 3 in these systems. N is the number of nodes in the system. The three systems we investigate typically have hundreds to thousands of nodes. We assume nodes are indexed from 1 to N. S is defined as the scatter width.

System	Chunks per Node	Cluster Size	Scatter Width	Replication Scheme
Facebook	10000	1000-5000	10	Random replication on a small group of nodes, second and third replica reside on the same rack
RAMCloud	8000	100-10000	N-1	Random replication across all nodes
HDFS	10000	100-10000	200	Random replication on a large group of nodes, second and third replica reside on the same rack

Table 1: Replication schemes of data center storage systems. These parameters are estimated based on publicly available data [2, 5, 7, 24, 25]. For simplicity, we fix the HDFS scatter width to 200, since its value varies depending on the cluster and rack size.

If a system has a scatter width of S, each node's data is split uniformly across a group of S other nodes. That is, whenever a particular node fails, S other nodes can participate in restoring the replicas that were lost. Table 1 contains the parameters of the three systems.

We define a set, as a group of R distinct 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. We will show that a large number of distinct copysets increases the probability of losing data under a massive correlated failure. Throughout the paper, we will investigate the relationship between the number of copysets and the system's scatter width.

We define a $permutation$ as an ordered list of all nodes in the cluster. For example, $\{4, 1, 3, 6, 2, 7, 5\}$ is a permutation of a cluster with $N = 7$ nodes.

Finally, random replication is defined as the following algorithm. The first, or primary replica is placed on a random node from the entire cluster. Assuming the primary replica is placed on node i, the remaining $R - 1$ secondary replicas are placed on random machines chosen from nodes $\{i + 1, i + 2, ..., i + S\}$. If $S = N - 1$, the secondary replicas' nodes are chosen uniformly from all the nodes in the cluster [1]..

2.2 Random Replication

The primary reason most large scale storage systems use random replication is that it is a simple replication technique that provides strong protection against uncorrelated failures like individual server or disk failures [13, 25] [2]. These failures happen frequently (thousands of times a year on a large cluster [7, 10, 13]), and are caused by a variety of reasons, including software, hardware and disk failures. Random replication across failure domains (e.g., placing the copies of a chunk on different racks) protects against concurrent failures that happen within a certain domain of nodes, such as racks or network segments. Such failures are quite common and typically occur dozens of times a year [7, 10, 13].

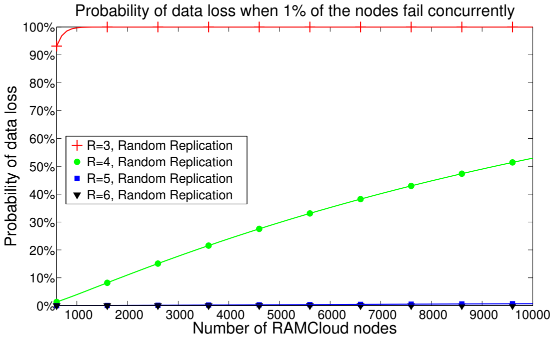

Figure 2: Simulation of the data loss probabilities of a RAMCloud cluster, varying the number of replicas per chunk.

However, multiple groups, including researchers from Yahoo! and LinkedIn, have observed that when clusters with random replication lose power, several chunks of data become unavailable [7, 25], i.e., all three replicas of these chunks are lost. In these events, the entire cluster loses power, and typically 0.5-1% of the nodes fail to reboot [7, 25]. Such failures are not uncommon; they occur once or twice per year in a given data center [7].

Figure 1 shows the probability of losing data in the event of a power outage in the three systems. The figure shows that RAMCloud and HDFS are almost guaranteed to lose data in this event, once the cluster size grows beyond a few hundred nodes. Facebook has a lower data loss probability of about 20% for clusters of 5000 nodes.

Multiple groups have expressed interest in reducing the incidence of data loss, at the expense of losing a larger amount of data at each incident [3, 8, 22]. For example, the Facebook HDFS team has modified the default HDFS implementation to constrain the replication in their deployment to significantly reduce the probability of data loss at the expense of losing more data during each incident [2, 5]. Facebook's Tech Lead of the HBase engineering team, has confirmed this point, as cited above [22]. Robert Chansler, Senior Manager of Hadoop Infrastructure at Linkedin has also confirmed the importance of addressing this issue: "A power-on restart

[1] Our definition of random replication is based on Facebook's design, which selects the replication candidates from a window of nodes around the primary node.

[2] For simplicity's sake, we assume random replication for all three systems, even though the actual schemes are slightly different (e.g., HDFS replicates the second and third replicas on the same rack [25].). We have found there is little difference in terms of data loss probabilities between the different schemes.

of HDFS nodes is a real problem, since it introduces a moment of correlated failure of nodes and the attendant threat that data becomes unavailable. Due to this issue, our policy is to not turn off Hadoop clusters. Administrators must understand how to restore the integrity of the file system as fast as possible, and *an option to reduce the number of instances when data is unavailable–at the cost of increasing the number of blocks recovered at such instances–can be a useful tool since it lowers the overall total down time*" [8].

The main reason some data center operators prefer to minimize the frequency of data loss events, is that there is a fixed cost to each incident of data loss that is not proportional to the amount of data lost in each event. The cost of locating and retrieving the data from secondary storage can cause a whole data center operations team to spend a significant amount of time that is unrelated to the amount of data lost [22]. There are also other fixed costs associated with data loss events. In the words of Robert Chansler: "In the case of data loss... [frequently] the data may be recomputed. For re-computation an application typically recomputes its entire data set whenever any data is lost. *This causes a fixed computational cost that is not proportional with the amount of data lost"*. [8]

One trivial alternative for decreasing the probability of data loss is to increase R. In Figure 2 we computed the probability of data loss under different replication factors in RAMCloud. As we would expect, increasing the replication factor increases the durability of the system against correlated failures. However, increasing the replication factor from 3 to 4 does not seem to provide sufficient durability in this scenario. In order to reliably support thousands of nodes in current systems, the replication factor would have to be at least 5. Using $R = 5$ significantly hurts the system's performance and almost doubles the cost of storage.

Our goal in this paper is to decrease the probability of data loss under power outages, without changing the underlying parameters of the system.

3. INTUITION

If we consider each chunk individually, random replication provides high durability even in the face of a power outage. For example, suppose we are trying to replicate a single chunk three times. We randomly select three different machines to store our replicas. If a power outage causes 1% of the nodes in the data center to fail, the probability that the crash caused the exact three machines that store our chunk to fail is only 0.0001%.

However, assume now that instead of replicating just one chunk, the system replicates millions of chunks (each node has 10,000 chunks or more), and needs to ensure that every single one of these chunks will survive the failure. Even though each individual chunk is very safe, in aggregate across the entire cluster, some chunk is expected to be lost. Figure 1 demonstrates this effect: in practical data center configurations, data loss is nearly guaranteed if *any combination of three nodes* fail simultaneously.

We define a copyset as a distinct set of nodes that contain all copies of a given chunk. *Each copyset is a single unit of failure*, i.e., when a copyset fails at least one data chunk is irretrievably lost. Increasing the number of copysets will increase the probability of data loss under a correlated failure, because there is a higher probability that the failed nodes will include at least one copyset. With random replication, almost every new replicated chunk creates a distinct copyset, up to a certain point.

3.1 Minimizing the Number of Copysets

In order to minimize the number of copysets a replication scheme can statically assign each node to a single copyset, and constrain the replication to these pre-assigned copysets. The first or primary replica would be placed randomly on any node (for load-balancing purposes), and the other secondary replicas would be placed deterministically on the first node's copyset.

With this scheme, we will only lose data if all the nodes in a copyset fail simultaneously. For example, with 5000 nodes, this reduces the data loss probabilities when 1% of the nodes fail simultaneously from 99.99% to 0.15%.

However, the downside of this scheme is that it severely limits the system's scatter width. This may cause serious problems for certain storage systems. For example, if we use this scheme in HDFS with $R = 3$, each node's data will only be placed on two other nodes. This means that in case of a node failure, the system will be able to recover its data from only two other nodes, which would significantly increase the recovery time. In addition, such a low scatter width impairs load balancing and may cause the two nodes to be overloaded with client requests.

3.2 Scatter Width

Our challenge is to design replication schemes that minimize the number of copysets given the required scatter width set by the system designer.

To understand how to generate such schemes, consider the following example. Assume our storage system has the following parameters: $R = 3$, $N = 9$ and $S = 4$. If we use random replication, 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 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 created only 6 copysets. Now, if three nodes fail, the probability of data loss is:

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

As we increase N, the relative advantage of creating the minimal number of copysets increases significantly. For example, if we choose a system with $N = 5000$, $R = 3$, $S = 10$ (like Facebook's HDFS deployment), we can design a replication scheme that creates about 8,300 copysets, while random replication would create about 275,000 copysets.

The scheme illustrated above has two important properties that form the basis for the design of Copyset Replication. First, each copyset overlaps with each other copyset by at most one node (e.g., the only overlapping node of copysets $\{4,5,6\}$ and $\{3,6,9\}$ is node 6). This ensures that each copyset increases the scatter width for its nodes by exactly $R-1$. Second, the scheme ensures that the copysets cover all the nodes equally.

Our scheme creates two permutations, and divides them into copysets. Since each permutation increases the scatter width by $R-1$, the overall scatter width will be:

$$S = P(R-1)$$

Where P is the number of permutations. This scheme will create $P\frac{N}{R}$ copysets, which is equal to: $\frac{S}{R-1}\frac{N}{R}$.

The number of copysets created by random replication for values of $S < \frac{N}{2}$ is: $N\binom{S}{R-1}$. This number is equal to the number of primary replica nodes times $R-1$ combinations of secondary replica nodes chosen from a group of S nodes. When S approaches N, the number of copysets approaches the total number of sets, which is equal to $\binom{N}{R}$.

In summary, in a minimal copyset scheme, the number of copysets grows linearly with S, while random replication creates $O(S^{R-1})$ copysets. Figure 3 demonstrates the difference in data loss probabilities as a function of S, between random replication and Copyset Replication, the scheme we develop in the paper.

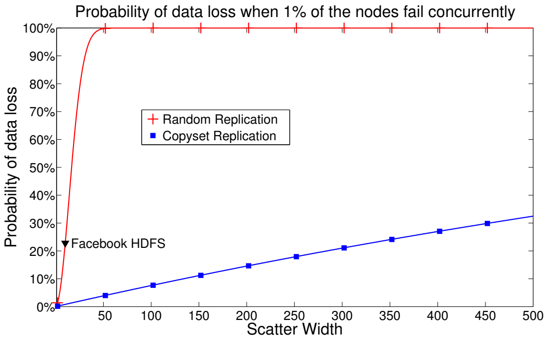

Figure 3: Data loss probability when 1% of the nodes fail simultaneously as a function of S, using $N = 5000$, $R = 3$.

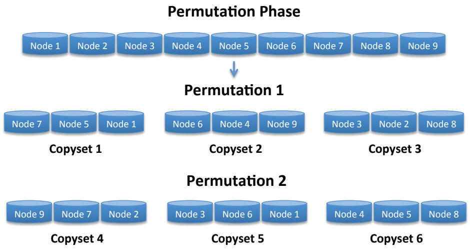

Figure 4: Illustration of the Copyset Replication Permutation phase.

4. DESIGN

In this section we describe the design of a novel replication technique, Copyset Replication, that provides a near optimal trade-off between the scatter width and the number of copysets.

As we saw in the previous section, there exist replication schemes that achieve a linear increase in copysets for a linear increase in S. However, it is not always simple to design the optimal scheme that creates non-overlapping copysets that cover all the nodes. In some cases, with specific values of N, R and S, it has even been shown that no such non-overlapping schemes exist [18, 19]. For a more detailed theoretical discussion see Section 7.1.

Therefore, instead of using an optimal scheme, we propose Copyset Replication, which is close to optimal in practical settings and very simple to implement. Copyset Replication randomly generates permutations and splits each permutation into copysets. We will show that as long as S is much smaller then the number of nodes in the system, this scheme is likely to generate copysets with at most one overlapping node.

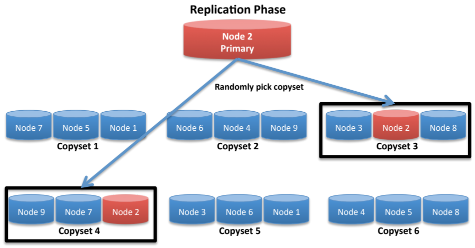

Figure 5: Illustration of the Copyset Replication Replication phase.

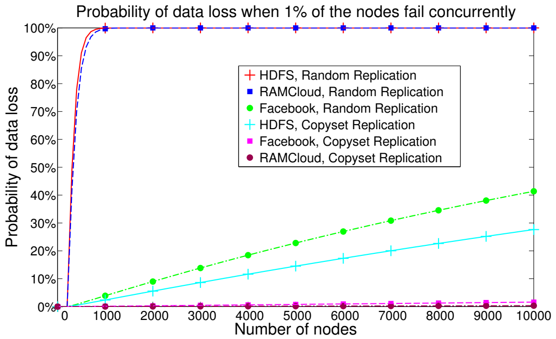

Figure 6: Data loss probability of random replication and Copyset Replication with $R = 3$, using the parameters from Table 1. HDFS has higher data loss probabilities because it uses a larger scatter width ($S = 200$).

Copyset Replication has two phases: Permutation and Replication. The permutation phase is conducted offline, while the replication phase is executed every time a chunk needs to be replicated.

Figure 4 illustrates the permutation phase. In this phase we create several permutations, by randomly permuting the nodes in the system. The number of permutations we create depends on S, and is equal to $P = \frac{S}{R-1}$. If this number is not an integer, we choose its ceiling. Each permutation is split consecutively into copysets, as shown in the illustration. The permutations can be generated completely randomly, or we can add additional constraints, limiting nodes from the same rack in the same copyset, or adding network and capacity constraints. In our implementation, we prevented nodes from the same rack from being placed in the same copyset by simply reshuffling the permutation until all the constraints were met.

In the replication phase (depicted by Figure 5) the system places the replicas on one of the copysets generated in the permutation phase. The first or primary replica can be placed on any node of the system, while the other replicas (the secondary replicas) are placed on the nodes of a randomly chosen copyset that contains the first node.

Copyset Replication is agnostic to the data placement policy of the first replica. Different storage systems have certain constraints when choosing their primary replica nodes. For instance, in HDFS, if the local machine has enough capacity, it stores the primary replica locally, while RAMCloud uses an algorithm for selecting its primary replica based on Mitzenmacher's randomized load balancing [23]. The only requirement made by Copyset Replication is that the secondary replicas of a chunk are always placed on one of the copysets that contains the primary replica's node. This constrains the number of copysets created by Copyset Replication.

4.1 Durability of Copyset Replication

Figure 6 is the central figure of the paper. It compares the data loss probabilities of Copyset Replication and random replication using 3 replicas with RAMCloud, HDFS and Facebook. For HDFS and Facebook, we plotted the same S values for Copyset Replication and random replication. In the special case of RAMCloud, the recovery time of nodes is not related to the number of permutations in our scheme, because disk nodes are recovered from the memory across all the nodes in the cluster and not from other disks. Therefore, Copyset Replication with with a minimal $S = R - 1$ (using $P = 1$) actually provides the same node recovery time as using a larger value of S. Therefore, we plot the data probabilities for Copyset Replication using $P = 1$.

We can make several interesting observations. Copyset Replication reduces the probability of data loss under power outages for RAMCloud and Facebook to close to zero, but does not improve HDFS as significantly. For a 5000 node cluster under a power outage, Copyset Replication reduces RAMCloud's probability of data loss from 99.99% to 0.15%. For Facebook, that probability is reduced from 22.8% to 0.78%. In the case of HDFS, since the scatter width is large ($S = 200$), Copyset Replication significantly improves the data loss probability, but not enough so that the probability of data loss becomes close to zero.

Figure 7 depicts the data loss probabilities of 5000 node RAMCloud, HDFS and Facebook clusters. We can observe that the reduction of data loss caused by Copyset Replication is equivalent to increasing the number of replicas. For example, in the case of RAMCloud, if the system uses Copyset Replication with 3 replicas, it has lower data loss probabilities than random replication with 5 replicas. Similarly, Copyset Replication with 3 replicas has the same the data loss probability as random replication with 4 replicas in a Facebook cluster.

The typical number of simultaneous failures observed in data centers is 0.5-1% of the nodes in the cluster [25]. Figure 8 depicts the probability of data loss in Face-

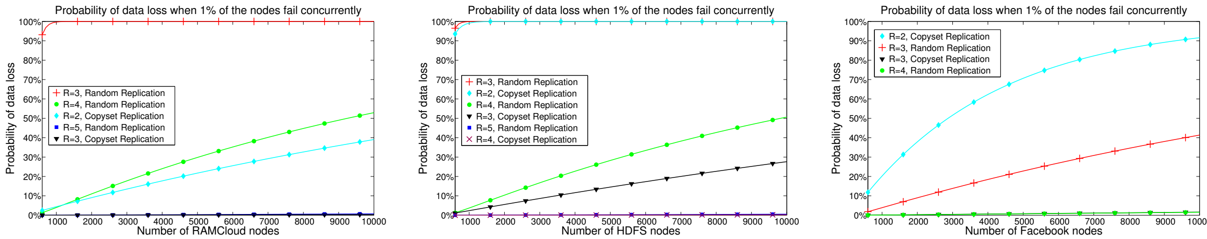

Figure 7: Data loss probability of random replication and Copyset Replication in different systems.

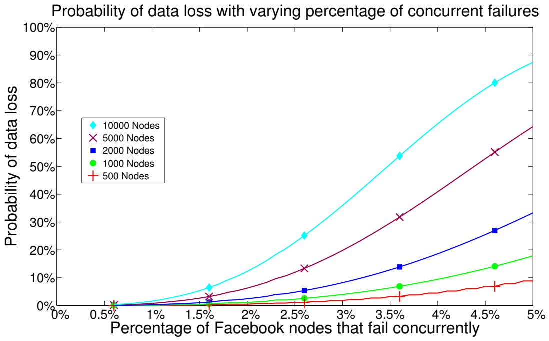

Figure 8: Data loss probability on Facebook's HDFS cluster, with a varying percentage of the nodes failing simultaneously.

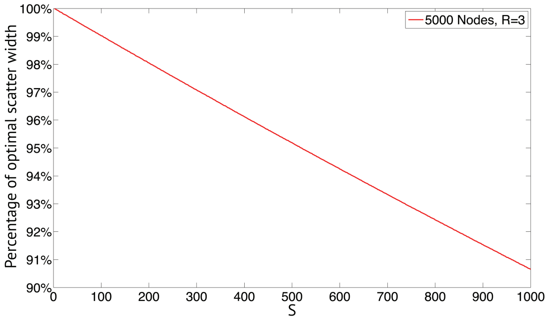

Figure 9: Comparison of the average scatter width ofCopyset Replication to the optimal scatter width in a 5000-node cluster.

book's HDFS system as we increase the percentage of simultaneous failures much beyond the reported 1%. Note that Facebook commonly operates in the range of 1000-5000 nodes per cluster (e.g., see Table 1). For these cluster sizes Copyset Replication prevents data loss with a high probability, even in the scenario where 2% of the nodes fail simultaneously.

4.2 Optimality of Copyset Replication

Copyset Replication is not optimal, because it doesn't guarantee that all of its copysets will have at most one overlapping node. In other words, it doesn't guarantee that each node's data will be replicated across exactly S different nodes. Figure 9 depicts a monte-carlo simulation that compares the average scatter width achieved by Copyset Replication as a function of the maximum S if all the copysets were non-overlapping for a cluster of 5000 nodes.

The plot demonstrates that when S is much smaller than N, Copyset Replication is more than 90% optimal. For RAMCloud and Facebook, which respectively use $S = 2$ and $S = 10$, Copyset Replication is nearly optimal. For HDFS we used $S = 200$, and in this case Copyset Replication provides each node an average of 98% of the optimal bandwidth, which translates to $S = 192$.

4.3 Expected Amount of Data Lost

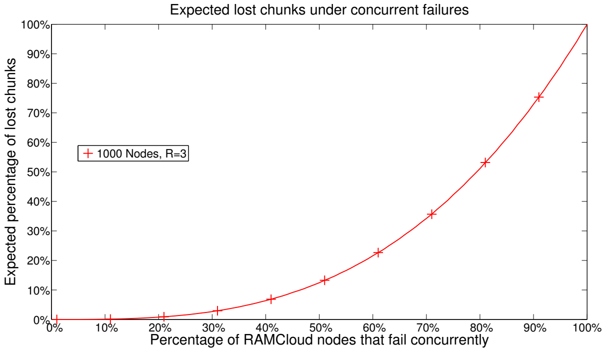

Figure 10: Expected amount of data lost as a percentage of the data in the cluster.

Copyset Replication trades off the probability of data loss with the amount of data lost in each incident. The expected amount of data lost remains constant regardless of the replication policy. Figure 10 shows the amount of data lost as a percentage of the data in the cluster.

Therefore, a system designer that deploys Copyset Replication should expect to experience much fewer events of data loss. However, each one of these events will lose a larger amount of data. In the extreme case, if we are using Copyset Replication with $S = 2$ like in RAMCloud, we would lose a whole node's worth of data at every data loss event.

5. EVALUATION

Copyset Replication is a general-purpose, scalable replication scheme that can be implemented on a wide range of data center storage systems and can be tuned to any scatter width. In this section, we describe our implementation of Copyset Replication in HDFS and RAMCloud. We also provide the results of our experiments on the impact of Copyset Replication on both systems' performance.

5.1 HDFS Implementation

The implementation of Copyset Replication on HDFS was relatively straightforward, since the existing HDFS replication code is well-abstracted. Copyset Replication is implemented entirely on the HDFS NameNode, which serves as a central directory and manages replication for the entire cluster.

The permutation phase of Copyset Replication is run when the cluster is created. The user specifies the scatter width and the number of nodes in the system. After all the nodes have been added to the cluster, the NameNode creates the copysets by randomly permuting the list of nodes. If a generated permutation violates any rack or network constraints, the algorithm randomly reshuffles a new permutation.

In the replication phase, the primary replica is picked using the default HDFS replication.

5.1.1 Nodes Joining and Failing

In HDFS nodes can spontaneously join the cluster or crash. Our implementation needs to deal with both cases.

When a new node joins the cluster, the NameNode randomly creates $\frac{S}{R-1}$ new copysets that contain it. As long as the scatter width is much smaller than the number of nodes in the system, this scheme will still be close to optimal (almost all of the copysets will be non-overlapping). The downside is that some of the other nodes may have a slightly higher than required scatter width, which creates more copysets than necessary.

When a node fails, for each of its copysets we replace it with a randomly selected node. For example, if the original copyset contained nodes $\{1, 2, 3\}$, and node 1 failed, we re-replicate a copy of the data in the original copyset to a new randomly selected node. As before, as long as the scatter width is significantly smaller than the number of nodes, this approach creates non-overlapping copysets.

5.2 HDFS Evaluation

We evaluated the Copyset Replication implementation on a cluster of 39 HDFS nodes with 100 GB of SSD storage and a 1 GB ethernet network. Table 2 compares the recovery time of a single node using Copyset Replication and random replication. We ran each recovery five times.

Replication	Recovery Time (s)	Minimal Scatter Width	Average Scatter Width	# Copysets
Random Replication	600.4	2	4	234
Copyset Replication	642.3	2	4	13
Random Replication	221.7	8	11.3	2145
Copyset Replication	235	8	11.3	77
Random Replication	139	14	17.8	5967
Copyset Replication	176.6	14	17.8	147
Random Replication	108	20	23.9	9867
Copyset Replication	127.7	20	23.9	240

Table 2: Comparison of recovery time of a 100 GB node on a 39 node cluster. Recovery time is measured after the moment of failure detection.

As we showed in previous sections, Copyset Replication has few overlapping copysets as long as S is significantly smaller than N. However, since our experiment uses a small value of N, some of the nodes did not have sufficient scatter width due to a large number of overlapping copysets. In order to address this issue, our Copyset Replication implementation generates additional permutations until the system reached the minimal desired scatter width for all its nodes. The additional permutations created more copysets. We counted the average number of distinct copysets. As the results show, even with the extra permutations, Copyset Replication still has orders of magnitude fewer copysets than random replication.

To normalize the scatter width between Copyset Replication and random replication, when we recovered the data with random replication we used the average scatter width obtained by Copyset Replication.

The results show that Copyset Replication has an overhead of about 5-20% in recovery time compared to random replication. This is an artifact of our small cluster size. The small size of the cluster causes some nodes to be members of more copysets than others, which means they have more data to recover and delay the overall recovery time. This problem would not occur if we used a realistic large-scale HDFS cluster (hundreds to thousands of nodes).

5.2.1 Hot Spots

One of the main advantages of random replication is that it can prevent a particular node from becoming a 'hot spot', by scattering its data uniformly across a random

Scatter Width	Mean Load	75th % Load	99th % Load	Max Load
10	10%	10%	10%	20%
20	5%	5%	5%	10%
50	2%	2%	2%	6%
100	1%	1%	2%	3%
200	0.5%	0.5%	1%	1.5%
500	0.2%	0.2%	0.4%	0.8%

Table 3: The simulated load in a 5000-node HDFS cluster with $R = 3$, using Copyset Replication. With Random Replication, the average load is identical to the maximum load.

set of nodes. If the primary node gets overwhelmed by read requests, clients can read its data from the nodes that store the secondary replicas.

We define the load $L(i, j)$ as the percentage of node i's data that is stored as a secondary replica in node j. For example, if $S = 2$ and node 1 replicates all of its data to nodes 2 and 3, then $L(1, 2) = L(1, 3) = 0.5$, i.e., node 1's data is split evenly between nodes 2 and 3.

The more we spread the load evenly across the nodes in the system, the more the system will be immune to hot spots. Note that the load is a function of the scatter width; if we increase the scatter width, the load will be spread out more evenly. We expect that the load of the nodes that belong to node i's copysets will be $dfrac1S$. Since Copyset Replication guarantees the same scatter width of random replication, it should also spread the load uniformly and be immune to hot spots with a sufficiently high scatter width.

In order to test the load with Copyset Replication, we ran a monte carlo simulation of data replication in a 5000-node HDFS cluster with $R = 3$.

Table 3 shows the load we measured in our monte carlo experiment. Since we have a very large number of chunks with random replication, the mean load is almost identical to the worst-case load. With Copyset Replication, the simulation shows that the 99th percentile loads are 1-2 times and the maximum loads 1.5-4 times higher than the mean load. Copyset Replication incurs higher worst-case loads because the permutation phase can produce some copysets with overlaps.

Therefore, if the system's goal is to prevent hot spots even in a worst case scenario with Copyset Replication, the system designer should increase the system's scatter width accordingly.

5.3 Implementation of Copyset Replication in RAMCloud

The implementation of Copyset Replication on RAMCloud was similar to HDFS, with a few small differences. Similar to the HDFS implementation, most of the code was implemented on RAMCloud's coordinator, which serves as a main directory node and also assigns nodes to replicas.

In RAMCloud, the main copy of the data is kept in a master server, which keeps the data in memory. Each master replicates its chunks on three different backup servers, which store the data persistently on disk.

The Copyset Replication implementation on RAMCloud only supports a minimal scatter width ($S = R - 1 = 2$). We chose a minimal scatter width, because it doesn't affect RAMCloud's node recovery times, since the backup data is recovered from the master nodes, which are spread across the cluster.

Another difference between the RAMCloud and HDFS implementations is how we handle new backups joining the cluster and backup failures. Since each node is a member of a single copyset, if the coordinator doesn't find three nodes to form a complete copyset, the new nodes will remain idle until there are enough nodes to form a copyset.

When a new backup joins the cluster, the coordinator checks whether there are three backups that are not assigned to a copyset. If there are, the coordinator assigns these three backups to a copyset.

In order to preserve $S = 2$, every time a backup node fails, we re-replicate its entire copyset. Since backups don't service normal reads and writes, this doesn't affect the sytem's latency. In addition, due to the fact that backups are recovered in parallel from the masters, re-replicating the entire group doesn't significantly affect the recovery latency. However, this approach does increase the disk and network bandwidth during recovery.

5.4 Evaluation of Copyset Replication on RAMCloud

We compared the performance of Copyset Replication with random replication under three scenarios: normal RAMCloud client operations, a single master recovery and a single backup recovery.

As expected, we could not measure any overhead of using Copyset Replication on normal RAMCloud operations. We also found that it does not impact master recovery, while the overhead of backup recovery was higher as we expected. We provide the results below.

5.4.1 Master Recovery

One of the main goals of RAMCloud is to fully recover a master in about 1-2 seconds so that applications experience minimal interruptions. In order to test master recovery, we ran a cluster with 39 backup nodes and 5 master nodes. We manually crashed one of the master servers, and measured the time it took RAMCloud to recover its data. We ran this test 100 times, both with Copyset Replication and random replication. As

Replication	Recovery Data	Recovery Time
Random Replication	1256 MB	0.73 s
Copyset Replication	3648 MB	1.10 s

Table 4: Comparison of backup recovery performance on RAMCloud with Copyset Replication. Recovery time is measured after the moment of failure detection.

expected, we didn't observe any difference in the time it took to recover the master node in both schemes.

However, when we ran the benchmark again using 10 backups instead of 39, we observed Copyset Replication took 11% more time to recover the master node than the random replication scheme. Due to the fact that Copyset Replication divides backups into groups of three, it only takes advantage of 9 out of the 10 nodes in the cluster. This overhead occurs only when we use a number of backups that is not a multiple of three on a very small cluster. Since we assume that RAMCloud is typically deployed on large scale clusters, the master recovery overhead is negligible.

5.4.2 Backup Recovery

In order to evaluate the overhead of Copyset Replication on backup recovery, we ran an experiment in which a single backup crashes on a RAMCloud cluster with 39 masters and 72 backups, storing a total of 33 GB of data. Table 4 presents the results. Since masters re-replicate data in parallel, recovery from a backup failure only takes 51% longer using Copyset Replication, compared to random replication. As expected, our implementation approximately triples the amount of data that is re-replicated during recovery. Note that this additional overhead is not inherent to Copyset Replication, and results from our design choice to strictly preserve a minimal scatter width at the expense of higher backup recovery overhead.

6. DISCUSSION

This section discusses how coding schemes relate to the number of copysets, and how Copyset Replication can simplify graceful power downs of storage clusters.

6.1 Copysets and Coding

Some storage systems, such as GFS, Azure and HDFS, use coding techniques to reduce storage costs. These techniques generally do not impact the probability of data loss due to simultaneous failures.

Codes are typically designed to compress the data rather than increase its durability. If the coded data is distributed on a very large number of copysets, multiple simultaneous failures will still cause data loss.

In practice, existing storage system parity code implementations do not significantly reduce the number of copysets, and therefore do not impact the profile of data loss. For example, the HDFS-RAID [1, 11] implementation encodes groups of 5 chunks in a RAID 5 and mirroring scheme, which reduces the number of distinct copysets by a factor of 5. While reducing the number of copysets by a factor of 5 reduces the probability of data loss, Copyset Replication still creates two orders of magnitude fewer copysets than this scheme. Therefore, HDFS-RAID with random replication is still very likely lose data in the case of power outages.

6.2 Graceful Power Downs

Data center operators periodically need to gracefully power down parts of a cluster [4, 10, 13]. Power downs are used for saving energy in off-peak hours, or to conduct controlled software and hardware upgrades.

When part of a storage cluster is powered down, it is expected that at least one replica of each chunk will stay online. However, random replication considerably complicates controlled power downs, since if we power down a large group of machines, there is a very high probability that all the replicas of a given chunk will be taken offline. In fact, these are exactly the same probabilities that we use to calculate data loss. Several previous studies have explored data center power down in depth [17, 21, 27].

If we constrain Copyset Replication to use the minimal number of copysets (i.e., use Copyset Replication with $S = R - 1$), it is simple to conduct controlled cluster power downs. Since this version of Copyset Replication assigns a single copyset to each node, as long as one member of each copyset is kept online, we can safely power down the remaining nodes. For example, a cluster using three replicas with this version of Copyset Replication can effectively power down two-thirds of the nodes.

7. RELATED WORK

The related work is split into three categories. First, replication schemes that achieve optimal scatter width are related to a field in mathematics called combinatorial design theory, which dates back to the 19th century. We will give a brief overview and some examples of such designs. Second, replica placement has been studied in the context of DHT systems. Third, several data center storage systems have employed various solutions to mitigate data loss due to concurrent node failures.

7.1 Combinatorial Design Theory

The special case of trying to minimize the number of copysets when $S = N - 1$ is related to combinatorial design theory. Combinatorial design theory tries to answer questions about whether elements of a discrete finite set can be arranged into subsets, which satisfy certain "balance" properties. The theory has its roots in recreational mathematical puzzles or brain teasers in the 18th and 19th century. The field emerged as a formal area of mathematics in the 1930s for the design of agricultural experiments [12]. Stinson provides a comprehensive survey of combinatorial design theory and its applications. In this subsection we borrow several of the book's definitions and examples [26].

The problem of trying to minimize the number of copysets with a scatter width of $S = N - 1$ can be expressed a Balanced Incomplete Block Design (BIBD), a type of combinatorial design. Designs that try to minimize the number of copysets for any scatter width, such as Copyset Replication, are called unbalanced designs.

A combinatorial design is defined a pair (X, A), such that X is a set of all the nodes in the system (i.e., $X = \{1, 2, 3, ..., N\}$) and A is a collection of nonempty subsets of X. In our terminology, A is a collection of all the copysets in the system.

Let N, R and λ be positive integers such that $N > R \geq 2$. A (N, R, λ) BIBD satisfies the following properties:

1. $|A| = N$
2. Each copyset contains exactly R nodes
3. Every pair of nodes is contained in exactly λ copysets

When $\lambda = 1$, the BIBD provides an optimal design for minimizing the number of copysets for $S = N - 1$.

For example, a $(7, 3, 1) BIBD$ is defined as:

$X = \{1, 2, 3, 4, 5, 6, 7\}$
$A = \{123, 145, 167, 246, 257, 347, 356\}$

Note that each one of the nodes in the example has a recovery bandwidth of 6, because it appears in exactly three non-overlapping copysets.

Another example is the $(9, 3, 1) BIBD$:

$X = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
$A = \{123, 456, 789, 147, 258, 369, 159, 267, 348, 168, 249, 357\}$

There are many different methods for constructing new BIBDs. New designs can be constructed by combining other known designs, using results from graph and coding theory or in other methods [20]. The Experimental Design Handbook has an extensive selection of design examples [9].

However, there is no single technique that can produce optimal BIBDs for any combination of N and R. Moreover, there are many negative results, i.e., researchers that prove that no optimal designs exists for a certain combination of N and R [18, 19].

Due to these reasons, and due to the fact that BIBDs do not solve the copyset minimization problem for any scatter width that is not equal to $N - 1$, it is not practical to use BIBDs for creating copysets in data center storage systems. This is why we chose to utilize Copyset Replication, a non-optimal design based on random permutations that can accommodate any scatter width. However, BIBDs do serve as a useful benchmark to measure how optimal Copyset Replication in relationship to the optimal scheme for specific values of S, and the novel formulation of the problem for any scatter width is a potentially interesting future research topic.

7.2 DHT Systems

There are several prior systems that explore the impact of data placement on data availability in the context of DHT systems.

Chun et al. [15] identify that randomly replicating data across a large "scope" of nodes increases the probability of data loss under simultaneous failures. They investigate the effect of different scope sizes using Carbonite, their DHT replication scheme. Yu et al. [28] analyze the performance of different replication strategies when a client requests multiple objects from servers that may fail simultaneously. They propose a DHT replication scheme called "Group", which constrains the placement of replicas on certain groups, by placing the secondary replicas in a particular order based on the key of the primary replica. Similarly, Glacier [16] constrains the random spread of replicas, by limiting each replica to equidistant points in the keys' hash space.

None of these studies focus on the relationship between the probability of data loss and scatter width, or provide optimal schemes for different scatter width constraints.

7.3 Data Center Storage Systems

Facebook's proprietary HDFS implementation constrains the placement of replicas to smaller groups, to protect against concurrent failures [2, 5]. Similarly, Sierra randomly places chunks within constrained groups in order to support flexible node power downs and data center power proportionality [27]. As we discussed previously, both of these schemes, which use random replication within a constrained group of nodes, generate orders of magnitude more copysets than Copyset Replication with the same scatter width, and hence have a much higher probability of data loss under correlated failures.

Ford et al. from Google [13] analyze different failure loss scenarios on GFS clusters, and have proposed geo-replication as an effective technique to prevent data loss under large scale concurrent node failures. Geo-replication across geographically dispersed sites is a fail-safe way to ensure data durability under a power outage.

However, not all storage providers have the capability to support geo-replication. In addition, even for data center operators that have geo-replication (like Facebook and LinkedIn), losing data at a single site still incurs a high fixed cost due to the need to locate or recompute the data. This fixed cost is not proportional to the amount of data lost [8, 22].

8. ACKNOWLEDGEMENTS

We would like to thank David Gal, Diego Ongaro, Israel Cidon, Rashmi Vinayak and Shankar Pasupathy for their valuable feedback. We would also like to thank our shepherd, Bernard Wong, and the anonymous reviewers for their comments. Asaf Cidon is supported by the Leonard J. Shustek Stanford Graduate Fellowship. This work was supported by the National Science Foundation under Grant No. 0963859 and by STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] HDFS RAID. `http://wiki.apache.org/hadoop/HDFS-RAID`.

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

[3] L. A. Barroso. Personal Communication, 2013.

[4] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, Dec. 2007.

[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] R. J. Chansler. Personal Communication, 2013.

[9] W. Cochran and G. Cox. Experimental designs . 1957.

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

[11] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson. DiskReduce: Replication as a prelude to erasure coding in data-intensive scalable computing, 2011.

[12] R. Fisher. An examination of the different possible solutions of a problem in incomplete blocks. *Annals of Human Genetics*, 10(1):52–75, 1940.

[13] 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.

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

[15] B. gon Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *IN PROC. OF NSDI*, pages 45–58, 2006.

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

[17] D. Harnik, D. Naor, and I. Segall. Low power mode in cloud storage systems.

[18] S. Houghten, L. Thiel, J. Janssen, and C. Lam. There is no (46, 6, 1) block design*. *Journal of Combinatorial Designs*, 9(1):60–71, 2001.

[19] P. Kaski and P. Östergård. There exists no (15, 5, 4) RBIBD. *Journal of Combinatorial Designs*, 9(3):227–232, 2001.

[20] J. Koo and J. Gill. Scalable constructions of fractional repetition codes in distributed storage systems. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 1366–1373. IEEE, 2011.

[21] J. Leverich and C. Kozyrakis. On the energy (in)efficiency of hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44(1):61–65, Mar. 2010.

[22] K. Mathukkaruppan. Personal Communication, 2012.

[23] M. D. Mitzenmacher. The power of two choices in randomized load balancing. Technical report, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 1996.

[24] D. Ongaro, S. M. Rumble, R. Stutsman, J. K. Ousterhout, and M. Rosenblum. Fast crash recovery in RAMCloud. In *SOSP*, pages 29–41, 2011.

[25] 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.

[26] D. Stinson. *Combinatorial designs: construction and analysis*. Springer, 2003.

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

[28] H. Yu, P. B. Gibbons, and S. Nath. Availability of multi-object operations. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*, NSDI'06, pages 16–16, Berkeley, CA, USA, 2006. USENIX Association.