MinCopysets: Derandomizing Replication In Cloud Storage

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

WORKING DRAFT

ABSTRACT

Randomized node selection is widely used in large-scale, distributed storage systems to both load balance chunks of data across the cluster and select replica nodes to provide data durability. We argue that while randomized node selection is great for load balancing, it fails to protect data under a common failure scenario. We present MinCopysets, a simple, general-purpose and scalable replication technique to improve data durability while retaining the benefits of randomized load balancing. Our contribution is to decouple the mechanisms used for load balancing from data replication: we use randomized node selection for load balancing but derandomize node selection for data replication. We show that MinCopysets provides significant improvements in data durability. For example in a 1000 node cluster under a power outage that kills 1% of the nodes, it reduces the probability of data loss from 99.7% to 0.02% compared to random replica selection. We implemented MinCopysets on two open source distributed storage systems, RAMCloud and HDFS, and demonstrate that MinCopysets causes negligible overhead on normal storage operations.

1. INTRODUCTION

Randomization is used as a common technique by large-scale distributed storage systems for load balancing and replication. These systems typically partition their data into chunks that are randomly distributed across hundreds or thousands of storage nodes, in order to support parallel access. To provide durability in the face of failures, these chunks are replicated on random nodes. Prominent systems that use randomized data distribution and replication are Hadoop Distributed File System (HDFS) [21], RAMCloud [19] and Windows Azure [9].

Unfortunately, cluster-wide power outages, a common type of data center failure scenario [10, 11, 15, 21], are handled poorly by randomization. This scenario stresses the durability of the system because a certain percentage of machines (0.5%-1%) [10, 21] do not come back to life after power has been restored. Figure 1 shows that once the size of the cluster scales beyond 300 nodes, three well-known distributed storage systems are nearly guaranteed to lose data in this scenario (for more information on this problem see Section 2). This problem has been documented in practice by Yahoo! [21], LinkedIn [10] and Facebook [8], and it theoretically affects systems that use Chord's replication scheme [22], such as Dynamo [13], Project Voldemort [4] and Apache Cassandra [1].

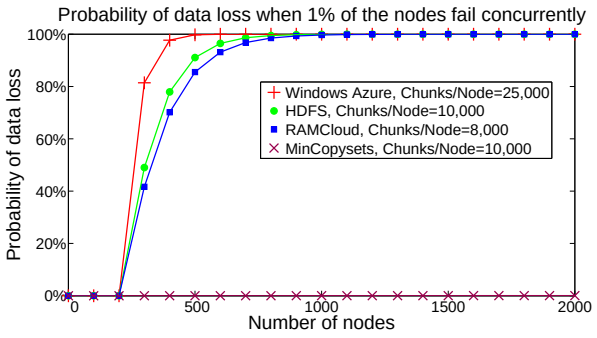

Figure 1: Computed probability of data loss when 1% of the nodes do not survive a restart after a power outage. The node and chunk sizes are based on publicly available sources [9, 10, 19] (see Table 1).

The key insight of this paper is that while randomization is an effective technique for spreading data across the cluster for load balancing, it is a poor mechanism for increasing durability. That is, randomization significantly increases the probability of data loss. The way to provide both load balancing and durability is to *decouple data distribution and data replication*. Data distribution should be random, while data replication should be deterministic.

We present MinCopysets, a simple, general-purpose replication scheme that decouples data distribution and replication. MinCopysets statically splits the nodes into

replication groups. Unlike the current scheme which randomly selects a node to store each chunk's replica, MinCopysets only randomly selects the node for the first replica. The other replicas are deterministically placed on the same replication group as the first replica's node.

Placing the first replica on a random node allows MinCopysets to distribute data widely across the cluster, while deterministically placing all replicas on a fixed replication group reduces the probability of data loss.

As shown in Figure 1, MinCopysets has a very low probability of data loss on large clusters. On a 1000 node cluster under a power outage, MinCopysets reduces the probability of data loss from 99.7% to 0.02%. If we deploy MinCopysets with 3 replicas, it achieves a lower data loss probability than the random replication scheme does with 5 replicas. Furthermore, we will later show that MinCopysets continues to have a very low data loss probability with 3 replicas on clusters of up to 100,000 nodes when 1% of the nodes in the cluster fail simultaneously.

MinCopysets does not change the long-term rate of data loss, but it changes the profile of data loss events in a way that most data center operators and applications will find attractive. With a randomized approach to 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. With MinCopysets data loss events are extremely rare: assuming a power outage occurs once a year, a 5000-node RAMCloud cluster would lose data once every 625 years. The system would lose an entire server's worth of data in this rare event. Since the cost of losing data is relatively independent of the amount of lost data (e.g., reading a backup tape), we argue that MinCopysets creates a superior profile. In this paper we use the term "durability" to refer to the frequency of data loss events, not the amount of data lost.

To show the generality of MinCopysets, we implemented it on two open source distributed storage systems: RAMCloud and HDFS. Our experience shows that adding support for MinCopysets is a relatively small implementation change that causes negligible overhead on normal operations. In addition, MinCopysets complements data locality preferences, failure domains, and network topologies of different storage systems.

The paper is split into the following sections. Section 2 provides the intuition for the paper and the problem. Section 3 discusses the design of MinCopysets. Section 4 provides details on our implementation of MinCopysets on RAMCloud and HDFS and its performance overhead. We discuss other data placement schemes related to MinCopysets, including the replication scheme of DHT systems, Facebook's data placement policy, and data coding schemes in Section 5. Section 6 includes related work, and Section 7 concludes the paper.

2. INTUITION

Cloud applications store and access their data on large-scale distributed storage systems that span thousands of machines [9, 16, 19, 21]. The common architecture of these systems is to partition their data into chunks and distribute the chunks across hundreds and thousands of machines, so that many chunks can be simultaneously read and written.

In addition, this architecture uses some form of random replication to protect against data loss when node failures inevitably happen. The basic idea is to replicate each data chunk on several (typically three) randomly chosen machines that reside on different racks [7, 16, 19] or failure domains [9]. Random replication is simple and fast, since unlike various coding schemes, it does not require any manipulation of the data for reading and writing the replicas.

Random replication provides strong protection against independent node failures [15, 21]. These failures happen independently and frequently (thousands of times a year on a large cluster [10, 11, 15]), and are caused by a variety of reasons, including software, hardware and disk failures. Random replication across racks or failure domains protects against correlated failures that happen within a certain set of nodes [15]. Such correlated failures are also common [10, 11, 15] and typically occur dozens of times a year due to rack and network failures.

However, random replication fails to protect against cluster-wide failures such as power outages. In these events, the entire cluster loses power, and typically 0.5-1% of the nodes fail to reboot [10, 21]. Such failures are not uncommon; they commonly occur once or twice per year in a given data center [10, 11, 15, 21].

Multiple groups, including researchers from Yahoo! and LinkedIn, have observed that when clusters lose power, the cluster loses several chunks of data [10, 21]. Some companies have deployed ad-hoc solutions to address this problem. For example, the Facebook HDFS team has modified their proprietary HDFS replication implementation to limit replication to constrained groups of nodes [3, 8]. However, their implementation is specific to their own HDFS cluster setup and size. In addition, researchers from Google [15] propose to geo-replicate data across data-centers in order to mitigate this problem. Unfortunately, geo-replication incurs a high latency and bandwidth overhead, because data needs to be written to a different location.

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

System	Node Size	Chunk Size	Nodes in Cluster	Replication Scheme
Windows Azure	75TB	1GB	200-400	Random replication within a small cluster
LinkedIn's HDFS	4TB	128MB	1000-4000	First replica on first rack, other replicas together on second rack
RAMCloud	192GB	8MB	100-100,000	Random replication

Table 1: Parameters of different large scale storage systems. These parameters are estimated based on publicly available details [9, 10, 19].

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, we replicate 10,000 chunks, and we want to ensure that every single one of these chunks will survive the massive failure. Even though each individual chunk is fairly safe, in aggregate across the entire cluster, some chunk is expected to be at risk. That is, the probability of losing all the copies of at least one of the chunks is much higher.

To understand this problem quantitatively, consider the following analysis. In a random replication scheme, each chunk is replicated onto R machines chosen randomly from a cluster with N machines. In the case of F simultaneous machine failures, a chunk that has been replicated on one of these machines will be completely lost if its other replicas are also among those F failed machines. The probability of losing a single chunk is:

$$\frac{\binom{F}{R}}{\binom{N}{R}}$$

Where $\binom{F}{R}$ is the number of combinations of failed replicas and $\binom{N}{R}$ is the maximum number of combinations of replicas. Therefore, the probability of losing at least one chunk in the cluster in the case of F simultaneous failures is given by:

$$1 - \left(1 - \frac{\binom{F}{R}}{\binom{N}{R}}\right)^{N \cdot C}$$

Where $N \cdot C$ is the total number of chunks in the cluster. Figure 1 plots the equation above as a function of N, assuming 1% of the nodes have failed ($F = 0.01N$), with $R = 3$. Unless stated otherwise, Table 1 contains the node and chunk sizes, which we used for all the graphs presented in the paper [1]. For simplicity's sake, throughout the paper we use the random replication scheme for all three systems, even though the systems' actual schemes are slightly different (e.g., HDFS replicates the second and third replicas on the same rack [21]), since we found very little difference in terms of data loss probability between the different schemes. We also assume the systems do not use any data coding (we will show in Section 5.4 that coding doesn't affect the data loss probability in these scenarios).

As we saw in Figure 1 data loss is nearly guaranteed if at least three nodes fail. The reason is that the random replication scheme creates a large number of *copysets*. A copyset is defined as a set of nodes that contain the replicas of a particular data chunk. In other words, *a copyset is a single unit of failure*, i.e., when a copyset fails at least one data chunk is irretrievably lost. With random replication, every new chunk with a very high probability creates a distinct copyset, up to a certain point. This point is determined by the maximum number of copysets possible in a cluster: $\binom{N}{R}$. This number is the total number of possible sets of replicas of nodes. Hence, as the number of chunks in a system grows larger and approaches $\binom{N}{R}$, under large-scale failures at least a few data chunks' copysets will fail, leading to data loss.

The intuition above is the reason why simply changing the parameters of the random replication scheme will not fully protect against simultaneous node failures. In the next two subsections we discuss the effects of such parameter changes.

2.1 Alternative I: Increase the Replication Factor

The first possible parameter change is to increase the number of replicas per chunk. The benefit of this change is that the maximum number of copysets possible in a cluster grows with the replication factor R as $\binom{N}{R}$. However, this is a losing battle, since the amount of data stored in distributed storage systems continues to grow, as clusters become larger and store more chunks. As the number of chunks increases and inevitably approaches $\binom{N}{R}$, the probability of data loss will increase.

Using the same computation from before, we varied the replication factor from 3 to 6 on a RAMCloud cluster. The results of this simulation are shown in Figure 2. As we would expect, increasing the replication factor does indeed increase the durability of the system against simultaneous failures. However, simply increasing the replication factor from 3 to 4, does not seem to provide the necessary resiliency against simultaneous failures on large clusters with thousands of nodes. As a reference, Table 1 shows that HDFS is usually run on clusters with

[1]Windows Azure is currently intended to support only about 200-400 nodes in a cluster (or "stamp"). Figure 1 projects the data loss if Azure continues to use its current replication scheme.

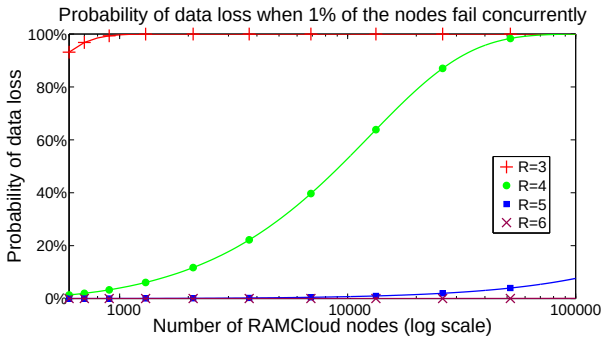

Figure 2: Simulation of the data loss probabilities of a RAMCloud cluster with 8000 chunks per node, varying the number of replicas per chunk.

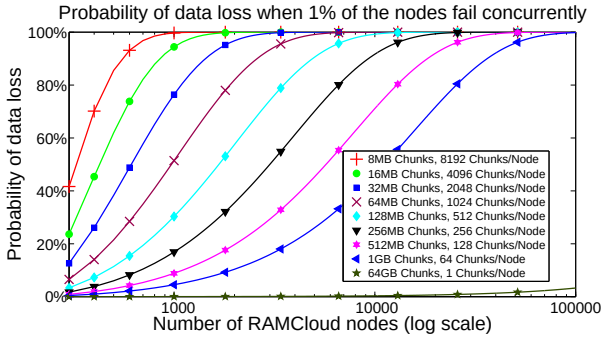

Figure 3: Data loss probabilities of a RAMCloud cluster, varying the number of chunks generated per node.

thousands of nodes [21], while RAMCloud is intended to support clusters of ten thousand nodes or more [19]. In order to support thousands of nodes reliably for current systems, the replication factor would have to be at least 5.

Increasing the replication factor to 5 significantly hurts the system's performance. For each write operation, the system needs to write out more replicas. This increases the network latency and disk bandwidth of write operations.

2.2 Alternative II: Increase the Chunk Size

Instead of increasing the replication factor, we can reduce the number of distinct copysets. A straightforward way is to increase the chunk size, so that for a fixed amount of total data, the total number of chunks in the system and consequently the number of copysets is reduced.

Using the same model as before, we plotted the data loss probability of a RAMCloud cluster with varying chunk sizes in Figure 3. The figure shows that increasing the size of chunks increases the durability of the system. However, we would reach the desired durability of the system only if we increase the chunk size by 3-4 orders of magnitude. If we take this scheme to the extreme, we can make the size of the chunk equal to the size of an entire node. This is in fact the same as implementing disk mirroring (i.e., RAID 1 [20]). In disk mirroring, every storage node has identical backup copies that are used for recovery. Therefore, the number of copysets with disk mirroring is equal to $\frac{N}{R}$. As we can see from Figure 3, disk mirroring significantly reduces the probability of data loss in case of simultaneous failures.

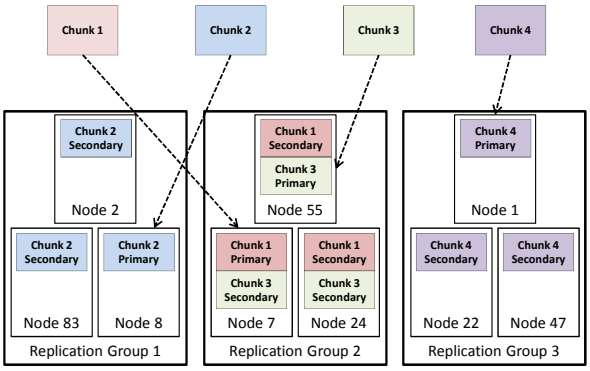

Figure 4: Illustration of the MinCopysets replication scheme. The first replica (primary replica) is chosen at random from the entire cluster. The other replicas (secondary replicas) are replicated on the statically defined replication group of the first replica.

Increasing the size of chunks incurs a significant cost, since it has the undesirable property that data objects are distributed to much fewer nodes. This compromises the parallel operation and load balancing of data center applications. For instance, if we have 10 GBs of data and we use 1 GB chunks, a data center application like MapReduce [12] could operate only across $10 * R$ machines, while if we use 64 MB chunks, MapReduce could run in parallel on $160 * R$ machines. Similarly, this would significantly degrade the performance of a system like RAMCloud, which relies on distributing its data widely for fast crash recovery [19].

Our goal is therefore to design a replication scheme that reduces the probability of any data loss to a negligible value (i.e., close to zero), while retaining the benefits of uniformly distributing chunks across the cluster for parallelization. In the following section, we describe MinCopysets, a scheme that possesses both of these properties.

3. DESIGN

In this section we describe the design of MinCopysets, a replication technique that retains the parallelization benefits of uniformly distributing chunks across the

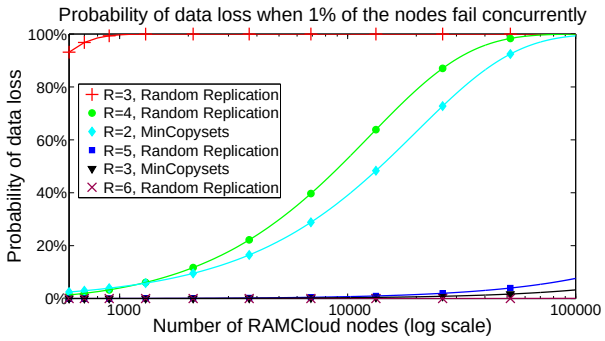

Figure 5: MinCopysets compared with random replication, using different numbers of replicas with 8000 chunks per node.

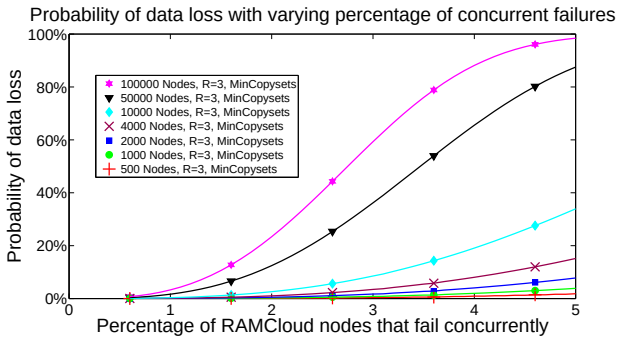

Figure 6: Probability of data loss, when we vary the percentage of simultaneously failed nodes beyond the reported 1% of typical simultaneous failures. Each node has 8000 chunks per node.

entire cluster, while ensuring low data loss probabilities even in the presence of cluster-wide failures. Previous random replication schemes randomly select a node for each one of their chunks' replicas. Random replication causes these systems to have a large number of copysets and become vulnerable to simultaneous node failures. MinCopysets's key insight is to decouple the placement of the first replica, which provides uniform data distribution, from the placement of the other replicas, which provide durability, without increasing the number of copysets.

The idea behind MinCopysets is simple. The servers are divided statically into replication groups (each server belongs to a single group). When a chunk has to be replicated, a primary node is selected randomly for its first replica, and the other replicas are stored deterministically on secondary nodes, which are the other nodes of the primary node's replication group. Figure 4 illustrates the design of our new replication scheme. The primary node is chosen at random, while the secondary nodes are always the members of the primary node's replication group.

MinCopysets decouples data distribution from chunk replication. Similar to the random replication scheme, it enables data center applications to operate in parallel on large amounts of data, because it distributes chunks uniformly across the data center. In addition, similar to disk mirroring, it limits the number of copysets generated by the cluster, because each node is only a member of a single replication group. Therefore, the number of copysets remains constant and is equal to $\frac{N}{R}$. Unlike random replication, the number of copysets does not increase with the number of chunks in the system.

In the next few subsections, we will go into further detail about the properties of MinCopysets.

3.1 Durability of MinCopysets

Figure 5 is the central figure of this paper. It compares the durability of the MinCopysets and random replication with different replication factors. We can make several interesting observations from the graph. First, on a 1000 node cluster under a power outage, MinCopysets reduces the probability of data loss from 99.7% to 0.02%. Second, if a system uses MinCopysets with 3 replicas, it has lower data loss probabilities than a system using random replication with 5 replicas, and a system using MinCopysets with 2 replicas has lower probabilities than random replication with 4 replicas. In other words, MinCopysets has significantly lower data loss probabilities than random replication with more replicas. Third, MinCopysets with 3 replicas can scale up to 100,000 nodes, which means that MinCopysets can support the cluster sizes of all widely used large-scale distributed storage systems (see Table 1).

The typical number of simultaneous failures observed in commercial data centers is 0.5-1% of the nodes in the cluster [21]. Figure 6 depicts the probability of data loss as we increase the percentage of simultaneous failures much beyond the reported 1%. Note that commercial storage systems commonly operate in the range of 200-4000 nodes per cluster (e.g., see Table 1 and GFS [16]). For these cluster sizes MinCopysets prevents data loss with a high probability, even in the very unlikely and hypothetical scenario where 4% of the nodes fail simultaneously.

3.2 Compatibility with Existing Systems

An important property of MinCopysets is that it does not require changing the architecture of the storage system. Unlike the alternative solutions that we discussed in Section 2, MinCopysets frees the storage system to use chunks of any size with any replication factor.

In addition, MinCopysets provides the storage system with the freedom to decide how to split the nodes into

replication groups. For example, groups can be formed from nodes that belong to different failure domains, in order to prevent simultaneous failures when an entire failure domain (e.g. rack, part of the network) fails.

Furthermore, large scale storage systems have additional constraints when choosing their primary replica. For instance, in HDFS, if the local machine has enough capacity, it stores the primary replica locally, while RAMCloud uses a semi-randomized approach for selecting its primary replica, based on Mitzenmacher's randomized load balancing algorithms [18]. MinCopysets can support any primary replica selection scheme, as long as the secondary replicas are always part of the same replication group as the primary.

The new scheme can be implemented as a modification of the random replication scheme. In order to enable MinCopysets, we need to add support for a centralized function that will split the nodes into groups, and modify the existing replication scheme to adhere to these groups. Fortunately, most large scale storage systems have a highly available centralized server that is in charge of managing the nodes in the cluster (e.g., Windows Azure's Stream Manager, HDFS' NameNode, RAMCloud's Coordinator). As we will demonstrate in Section 4, it was relatively straightforward to modify the RAMCloud and HDFS random replication scheme to support MinCopysets.

3.3 Node Recovery Overhead

For each chunk, random replication independently chooses nodes for each of its replicas. In contrast, MinCopysets chooses an entire replication group, which consists of a fixed set of nodes. In case of a node failure, the entire replication group is impacted. This slightly complicates node recovery.

There are several ways to solve this problem. For example, the coordinator does not have to assign replication groups to all backups, and keep a small number of backups as 'reserve' nodes. Another alternative is to allow nodes to be members of several replication groups (e.g. 2 or 3). In the latter scheme, when a node fails, its group's remaining nodes can form a new replication group with a node that may already be a member of another replication group.

The brute force alternative, which is the one we implemented, is to re-replicate the entire replication group when one of the nodes in the group fails. The nodes that didn't fail and belonged to the original replication group can be then reassigned to a new replication group once enough unassigned nodes are available.

This approach increases the network and disk bandwidth during backup recovery. In addition, the backup nodes that were part of the failed replication group have to clean up their old data after it has been re-replicated. This would consume additional CPU and disk bandwidth. Note that this overhead is a result of a conscious design choice to simplify the management of replicas at the expense of added recovery bandwidth, and is not an inherent overhead of MinCopysets.

3.4 Supporting Multiple Replication Factors

GFS, Azure and HDFS allow users to define different replication factors for each chunk. MinCopysets supports multiple replication factors by maintaining multiple lists of replication groups for each replication factor. When it makes its replica placement decisions, the system simply uses the list that matches the chunk's replication factor.

Since MinCopysets significantly reduces the probability of data loss in case of simultaneous failures, it is probably unnecessary, in most cases, to replicate chunks using a replication factor larger than 3.

4. EVALUATION

MinCopysets is a general-purpose, scalable replication scheme that can be implemented on a wide range of distributed storage systems that distribute chunks randomly across the data center. In this section, we describe our implementation of MinCopysets in RAMCloud and HDFS. We also provide the results of our experiments on the impact of MinCopysets on RAMCloud's performance.

We found that it was simple to add support for MinCopysets to RAMCloud and HDFS. Not including unit tests, the change to the original RAMCloud source code amounted to about 600 lines of C++ and the HDFS change was about 200 lines of Java.

4.1 Implementation of MinCopysets in RAMCloud

RAMCloud is a memory-based persistent distributed storage system. The main goal of RAMCloud is to provide low latency access; a read RPC of a 100 byte object can be serviced in approximately 5 microseconds, end-to-end.

In order to keep data persistent, RAMCloud stores one copy of each data object in memory, while saving additional copies on three disk-based backup nodes. Due to the low latency requirements of RAMCloud, one of the main challenges of the system is to provide fast crash recovery, in case the main memory copy of the data is lost. To this end, RAMCloud splits its data into small 8MB chunks, which are scattered uniformly across the entire data center. When one of the in-memory nodes is lost, the data is recovered from the entire cluster in parallel [19].

4.1.1 Architecture

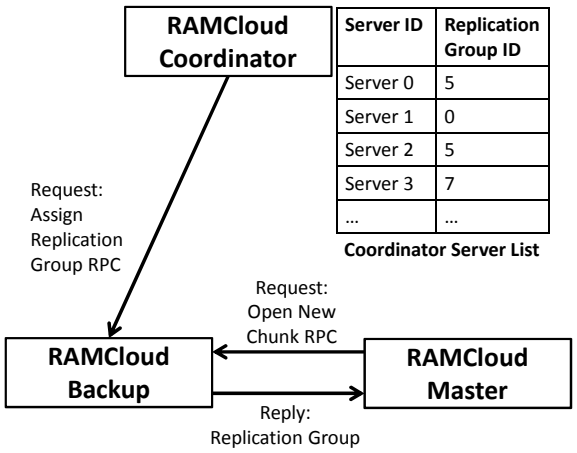

Figure 7: Illustration of the MinCopysets implementation on RAMCloud. The coordinator assigns each backup to a single replication group. When the master creates a new chunk, the backup informs the master of the members of its replication group.

Figure 7 illustrates the architecture of our implementation. We implemented MinCopysets on the RAMCloud coordinator, master and backup.

The RAMCloud coordinator is a highly available server that is in charge of managing the cluster. It keeps an up-to-date list of all the servers in the cluster and their addresses. It also controls the registration of new servers, decides whether a node has failed and coordinates node recoveries.

The RAMCloud master services client requests from the in-memory copy of the data. The master is also in charge of selecting the backup servers to store the persistent disk based copies of the data.

The RAMCloud backups are in charge of storing data persistently. They are largely passive: in normal operation, their only function is to receive write RPCs from masters and write the data to disk. Backups read data only when a master server needs to be recovered.

In our implementation, the coordinator is in charge of assigning the backups to replication groups. The master then replicates its chunks on replication groups.

The coordinator controls the assignment of backup nodes to replication groups because it has the full authority to decide which nodes are part of the cluster. In order to support MinCopysets, we added a new field to each entry of the coordinator's server list, which contains the replication group ID.

When a new backup joins the cluster, the coordinator iterates through the server list, to see whether there are three backups on three different racks that are not assigned to a replication group. The coordinator assigns these three backups a replication group, by sending them an RPC, which informs them of their group.

When a RAMCloud master tries to create a new chunk, it first selects a primary backup as it did in the original random replication scheme using Mitzenmacher's randomized load balancing algorithm [18]. If the backup has been assigned a replication group, it will accept the master's write request and respond with the other members in its replication group. The master will then replicate the other two copies of the chunk to the two remaining members of the group. If the backup did not accept the master's write request, the master will retry its write RPC on a new primary backup.

We describe several additional issues we had to solve in our implementation below.

4.1.2 Backup Recovery

MinCopysets introduces some changes to backup recovery. For the simplicity of the implementation, every time a backup node fails, we re-replicate its entire replication group. This approach increases the disk and network bandwidth during recovery.

After a node fails, the coordinator changes the replication group ID of all the nodes in its replication group to **limbo**. Limbo backups can still service read requests for the purpose of master recovery but cannot accept new chunks. Therefore, when a master tries to create a new chunk on a limbo backup, the backup will refuse its write request. Limbo backups are treated by the coordinator as new backups that have not yet been assigned a replication group ID. Once there are enough unassigned or limbo nodes to form a new group, the coordinator will assign them a new replication group ID, and they can begin servicing write requests again, even if they still store leftover data from the old replication group.

In addition, all RAMCloud masters are notified of the node's failure. As a result, any RAMCloud master that had data stored on the node's group tries to re-replicate its data on a new replication group. If a backup has data remaining from its old replication group, it won't garbage collect the data until the masters have re-replicated it entirely on a new group.

The backup recovery is not part of the critical path of RAMCloud. Therefore, the extra time it takes to fully re-replicate the replicas on the new replication group does not affect the system's performance. The only overhead is the extra consumed bandwidth. Note that backup recovery tasks can be de-prioritized in comparison to more important real-time operations like client read and write requests and operations related to master recovery. In addition, we assume that failures are relatively rare events.

4.1.3 Other Issues

In the original random replication scheme, it is straightforward for a master to create a new chunk on a

backup. It sends write RPCs in parallel to three backups, and if one of the RPCs fails for some reason, it retries and creates the chunk on a different backup. The RPC could fail due to a node failure or because the backup doesn't have enough disk capacity to allocate for the new chunk. With MinCopysets, if any one of the replication group nodes cannot write the new chunk, all three RPCs need to be aborted. Otherwise there may be inconsistencies in RAMCloud's log.

A naïve way to tackle this problem is to declare the node that didn't accept the RPC as a failed node. Since backup recovery now operates on entire replication groups, the data on the failed node's group would simply be re-replicated as new chunks on a new replication group which would solve any outstanding consistency issues.

The problem with this approach is that RPCs can fail because nodes do not have sufficient disk capacity. If we treat any node that reaches the limits of its disk capacity as a failed node, the system would generate false positive backup recoveries. This could lead to a systematic collapse since each crash reduces the total amount of storage available to the system which, in turn, increases the likelihood of nodes running out of space.

In order to prevent this problem, we implemented an unwind algorithm for new chunks. Anytime a master tries to create a new chunk in one of the backups and it fails because the backup does not have enough disk capacity, the master sends an unwind RPC to all of the backups in the replication group, and retries replication on a different group. If, for some reason, the unwind RPC fails on one of the nodes, we treat it as a node failure and initiate backup recovery on the entire replication group.

4.2 HDFS Implementation

To demonstrate the general-purpose nature of MinCopysets, we describe our implementation of MinCopysets in HDFS.

The implementation of MinCopysets on HDFS is similar to RAMCloud's implementation. The main difference is that in HDFS the NameNode, which controls all file system metadata, dictates the placement of every chunk replica. In contrast, replica placement decisions are completely decentralized in RAMCloud.

4.2.1 Architecture

The centralized NameNode simplified the implementation of MinCopysets on HDFS. For normal operations, we only needed to modify the NameNode to assign new DataNodes (storage nodes) to replication groups and choose replica placements based on these group assignments.

The HDFS replication code is well-abstracted. We added support for assigning DataNodes to replication groups, choosing replica DataNodes based on replication groups, and re-replicating blocks to other groups after a DataNode failure.

4.2.2 Chunk and Network Load Balancing

MinCopysets complicates chunk rebalancing in HDFS. Rebalancing is the act of migrating replicas to different storage servers to more evenly spread data across DataNodes. This is desirable, for instance, when additional DataNodes are added to the system. With MinCopysets a single replica of a chunk cannot simply be moved to another node belonging to a different replication group. Instead, a new replication group must be chosen and each replica of the chosen chunk must be moved to a member of the new set.

Another potential issue in the MinCopysets HDFS implementation is reduced network load balancing caused by HDFS' pipelined replication. In HDFS, DataNodes replicate data on a pipeline from one node to the next, in an order that minimizes the network distance from the client to the last DataNode. With the MinCopysets implementation, since replication groups are chosen statically, the system might over-utilize a relatively small number of links that connect the replication group's nodes which may result in network bottlenecks.

In order to solve the network load balancing issues and to support MinCopysets on such systems, the NameNode could take the network topology into account when assigning the nodes to different replication groups. In addition, since certain network links can become congested, the central node could periodically load balance certain replication groups (i.e., reassign certain nodes to different groups). Since we still need to evaluate the network load balancing issue in HDFS, we have not yet implemented these solutions.

4.3 Performance of MinCopysets on RAMCloud

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

As expected, we found that using MinCopysets incurs almost no overhead on normal RAMCloud operations and on master recovery, while the overhead of backup recovery was higher as we expected. We provide the results below.

4.3.1 Normal Client Operations

In order to determine the overhead of MinCopysets replication under normal client operations, we ran a performance test suite. The test runs several master servers along with multiple backup servers. Up to 10 clients try

System	Recovery Data	Recovery Time
Without MinCopysets	1256 MB	0.73 s
With MinCopysets	3648 MB	1.10 s

Table 2: Comparison of backup recovery performance on RAMCloud with MinCopysets. Recovery time is measured after the moment of failure detection.

to write and read objects of various sizes (100 bytes to 1 MB) to and from the masters. We ran these tests 100 times, and found no difference between the performance of RAMCloud with the MinCopysets and with the random replication scheme.

4.3.2 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 the MinCopysets and random replication schemes. As expected, we didn't observe any difference in the time it took to recover the master node in both scheme.

However, when we ran the benchmark again using 10 backups instead of 39, we observed MinCopysets took 11% more time to recover the master node than the random replication scheme. Due to the fact that MinCopysets 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. As expected, the overhead gets smaller as the grows larger. Since we assume that RAMCloud is typically deployed on large scale clusters, the master recovery overhead is usually negligible.

4.3.3 Backup Recovery

In order to evaluate the overhead of MinCopysets on backup recovery, we ran an experiment where a single backup crashes on a RAMCloud cluster with 39 masters and 72 backups, storing a total of 33 GB of data. Table 2 presents the results. Since masters re-replicate data in parallel, recovery from a backup failure only takes 51% longer using MinCopysets, 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 MinCopysets, and results from our design choice to reduce the complexity of the coordinator at the expense of higher backup recovery overhead.

5. DISCUSSION

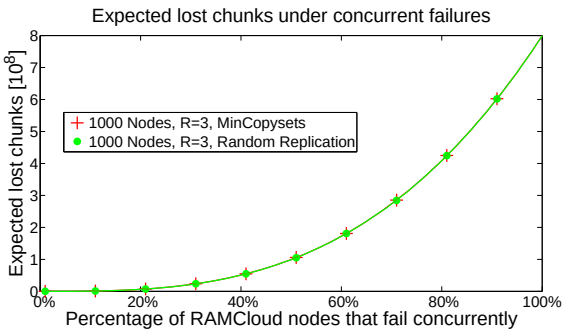

Figure 8: The expected number of chunks lost in a 1000 node RAMCloud cluster with 8000 chunks per node, when we vary the number of concurrent failures.

This section discusses several additional issues related to MinCopysets. First, we describe the difference between data loss probability and the actual amount of data lost. Second, we discuss how the notion of copysets can be applied to DHT systems. Third, we show some alternative replication schemes that have higher data loss probabilities than MinCopysets, but can reduce the overhead of node recovery. Finally, we discuss how coding schemes relate to the number of copysets.

5.1 Amount of Data Lost

MinCopysets trades off the probability of failure with the amount of data lost in each failure. In other words, MinCopysets reduces the probability of losing any data during simultaneous failures, but increases the amount lost when data loss does occur. In order to investigate this property, we ran a monte carlo simulation that compares the expected number of lost chunks with random replication and MinCopysets under concurrent node failures.

The results of this simulation are shown in Figure 8. The graph shows that the expected number of lost chunks is almost identical in both schemes. Therefore, a system designer that deploys MinCopysets should expect to experience much fewer events of data loss. However, each one of these events will lose a large amount of data (i.e. at least a whole node's worth of data). For example, if we assume a power outage that causes 1% of the nodes to fail occurs once a year, a 5000 node RAMCloud cluster with random replication will lose about 344 MB every year. In contrast, MinCopysets will take on average approximately 625 years to lose data. In the case of this very rare event, 64 GB of data will be lost.

Since each data loss event can incur certain fixed costs (e.g. rolling out magnetic tape archives to recover the lost data), most storage systems would probably prefer a significantly lower occurrence of data loss at the expense

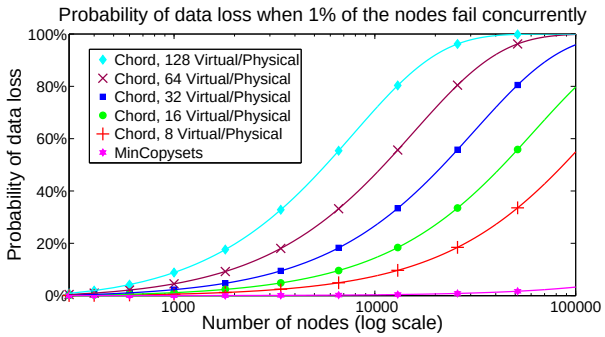

Figure 9: The probability of data loss in a cluster of nodes using Chord's replication scheme, with a varying number of virtual nodes per physical node. Each physical node has 8000 chunks.

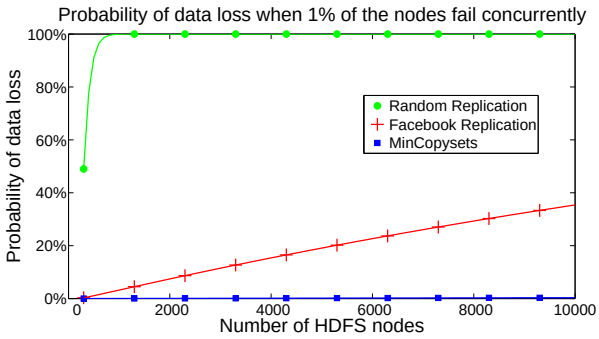

Figure 10: Simulation of Facebook's HDFS replication scheme, on a cluster with 10,000 chunks per node.

of losing a higher amount of data each time.

5.2 Copysets and Distributed Hash Tables

Dynamo [13], Project Voldemort [4] and Apache Cassandra [1] are all key-value storage systems that use hash key partitioning schemes similar to Distributed Hash Tables (DHT) such as Chord [22]. These typically assign each physical node multiple virtual nodes [2]. Each virtual node is assigned a range in the hash space. The replicas of the data are stored on a sequence of virtual nodes, according to their hash space. DHT based systems use multiple virtual nodes per physical node in order to distribute the key ranges across multiple nodes to ensure load balancing. The original Chord paper proposes to run $O(logN)$ virtual nodes, where N is the number of physical nodes.

This presents an interesting trade off with regards to copysets. As long as the number of virtual nodes per physical node is small, each node belongs to a relatively small number of replication groups, because the replicas are always distributed to the same sequence of virtual nodes. However, using a small number of virtual nodes may have a negative impact on load balancing.

Therefore, DHT replication balances between data distribution and data loss probability, similar to the random replication scheme. To demonstrate this effect, we computed the data loss probabilities under a power outage in a cluster using a DHT based replication scheme. The results are presented in Figure 9.

The takeway from this graph is that increasing the number of virtual nodes for load balancing increases the probability of data loss in the face of concurrent node failures. Therefore, unlike MinCopysets, the DHT replication scheme couples load balancing and durability.

[2]Voldemort uses a key partitioning similar to virtual nodes. Virtual node support is currently being implemented for Cassandra [5].

5.3 Other Alternatives to Contain Randomized Replication

MinCopysets is one of many possible schemes that can constrain the number of copysets, while still enabling replicas to be distributed widely in the cluster. For example, instead of having only one replication group per node, we could allow nodes to belong to a larger, fixed number of replication groups.

If nodes can belong to multiple groups, the overhead of node recovery can be reduced, as described in Section 3.3. In fact, the replication scheme that Facebook implemented in their HDFS cluster uses a small number of replication groups per node [3, 8]. In their proprietary implementation of Hadoop, they constrain the placement of replicas into smaller groups, by defining a window of 2 racks and 5 machines around the first replica's machine. The second and third replica are chosen at random from this window.

MinCopysets and has a lower probability of data loss than Facebook's scheme. To demonstrate this point, we simulated Facebook's scheme and compared it with MinCopysets, using typical HDFS chunk and node size parameters. Figure 10 presents the results of the simulation. As we can see, while Facebook's more relaxed scheme significantly improves the probabilities of the original random replication scheme, it is more exposed to concurrent failures on large clusters than MinCopysets. The advantage of using MinCopysets in comparison with schemes like Facebook's is that MinCopysets is more general-purpose. With MinCopysets, system designers do not have to worry about optimizing the "replication window" to match their specific cluster, chunk and node sizes.

5.4 Copysets and Coding

Some storage systems, such as GFS, Azure and HDFS, use coding techniques to reduce space usage or tolerate more failures. We have found that these techniques do

not impact the probability of data loss when using random replication, and the MinCopysets solution can be applied together with data coding.

Codes are typically designed to allow chunks to be resilient against failures of two nodes using a reduced amount of storage, but not against three failures or more. If the coded data is still 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 affect the number of copysets. For example, the HDFS-RAID [2, 14] implementation encodes groups of 5 chunks in a RAID 5 and mirroring scheme, which reduces the number of non-distinct copysets by a factor of 5. However, this is exactly equivalent to increasing the chunk size by a factor of 5, and we showed before (in Fig 3) that such a reduction does not significantly decrease the data loss probabilities of the system.

6. RELATED WORK

The related work is split into two categories. The first category is large scale storage systems that intentionally constrain the placement of replicas to prevent data loss due to concurrent node failures. The second category is RAID mirroring schemes, which were not originally designed to support the scale of Big Data applications.

6.1 Different Placement Policies

Similar to MinCopysets, Facebook's proprietary HDFS implementation constrains the placement of replicas into smaller groups, to protect against concurrent failures [3, 8]. As we discussed in the previous section, while this scheme improves the original random replication scheme, it is much more exposed to concurrent failures than MinCopysets and is not general-purpose.

Ford et al. from Google [15] 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 is a fail-safe way to ensure data durability under a power outage. However, geo-replication incurs high latency and bandwidth costs. In addition, not all storage providers have the capability to support geo-replication.

In contrast to geo-replication, MinCopysets can mitigate the probability of data loss under concurrent failures without moving data to a separate location.

6.2 RAID

Another replication scheme that is related to MinCopysets is RAID (for a detailed overview of RAID technology, see IBM's guide [6]). RAID 1 [20] is the basic disk mirroring scheme. In RAID 1, each disk is fully mirrored on a set of additional disks, in order to provide higher durability. This scheme was originally designed as a hardware solution for a single machine. In order to extend RAID 1 to multiple disks, RAID 1+0 or RAID 10 is designed as a stripe of mirrors. In this scheme, odd numbered chunks are replicated on one mirror, while even numbered chunks are replicated on the second mirror. This scheme was extended further to RAID 100 and RAID 1000. In these schemes, chunks are split into even more mirror groups.

As an extension to standard RAID systems, RAID-x [17] was designed as a distributed software RAID scheme for small clusters. Similar to our paper, it also identifies the need to decouple the distribution and data loss probability of data, using a combination of striping and mirroring.

While MinCopysets is inspired by RAID's concept of mirroring, in practice the two schemes are very different. MinCopysets operates on a data center scale, while these RAID schemes are designed to serve small clusters of large servers. In addition, the RAID scheme is rigid. Nodes cannot join or leave mirror groups, and each chunk is deterministically replicated on a certain mirror group. In contrast, MinCopysets can choose the first replica at random from the entire cluster, and nodes can be divided into replication groups using different flexible policies.

7. CONCLUSION

Conventional wisdom holds that randomization is a general-purpose technique for solving a variety of problems in large-scale distributed systems. This paper questions this assumption, and shows that randomization leads to poor data durability. In particular, we show that existing widely deployed systems such as HDFS and Azure that use random replication can lose data under common events such as power outages.

This paper presents MinCopysets, a simple general-purpose scalable replication scheme, that derandomizes data replication in order to achieve better data durability properties. MinCopysets decouples the mechanisms used for data distribution and durability. It allows system designers to use randomized node selection for data distribution to reap the benefits of parallelization and load balancing, while using deterministic replica selection to significantly improve data durability. We showed that with MinCopysets, systems can use only 3 replicas yet achieve the same data loss probabilities as systems using random replication with 5 replicas, and can safely scale up to 100,000 nodes. With relatively straightforward implementations, we added support for MinCopysets to RAMCloud and HDFS. MinCopysets does not introduce any significant overhead on normal storage operations, and can support any data locality or network

topology requirements of the underlying storage system.

References

[1] Cassandra. http://cassandra.apache.org/.

[2] Hdfs raid. http://wiki.apache.org/hadoop/HDFS-RAID.

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

[4] Project voldemort, a distributed database. http://project-voldemort.com/.

[5] Support multiple non-consecutive tokens per host (virtual nodes). https://issues.apache.org/jira/browse/CASSANDRA-4119.

[6] Understanding raid technology. http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp?topic=/diricinfo/fqy0_cselraid_copy.html.

[7] D. Borthakur. Hdfs block replica placement in your hands now?, 2009. http://hadoopblog.blogspot.com/2009/09/hdfs-block-replica-placement-in-your.html.

[8] 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.

[9] 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.

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

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

[12] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, Oct. 2007.

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

[15] 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.

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

[17] K. Hwang, H. Jin, and R. S. Ho. Orthogonal striping and mirroring in distributed raid for i/o-centric cluster computing. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):26–44, Jan. 2002.

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

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

[20] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, SIGMOD '88, pages 109–116, New York, NY, USA, 1988. ACM.

[21] 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.

[22] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.