DATA DURABILITY IN CLOUD STORAGE SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Asaf Cidon
July 2015

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mendel Rosenblum) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Sachin Katti)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Christos Kozyrakis)

Approved for the University Committee on Graduate Studies

Abstract

This dissertation challenges widely held assumptions about data replication in cloud storage systems. It demonstrates that existing cloud storage techniques are far from being optimal for guarding against different types of node failure events. The dissertation provides novel methodologies for analyzing node failures and designing non-random replication schemes that offer significantly higher durability than existing techniques, at the same storage cost and cluster performance.

Popular cloud storage systems typically replicate their data on random nodes to guard against data loss due to node failures. Node failures can be classified into two categories: independent and correlated node failures. Independent node failures are typically caused by independent server and hardware failures, and occur hundreds of times a year in a cluster of thousands of nodes. Correlated node failures are failures that cause multiple nodes to fail simultaneously and occur a handful of times a year or less. Examples of correlated failures include recovery following a power outage or a large-scale network failure.

The conventional wisdom to guard against node failures is to replicate each node's data three times within the same cluster, and also geo-replicate the entire cluster to a separate location to protect against correlated failures.

The dissertation shows that random replication within a cluster is almost guaranteed to lose data under common scenarios of correlated node failures. 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. The dissertation introduces Copyset Replication, a novel general-purpose replication technique that significantly reduces the frequency of data loss events within a cluster. It also presents an implementation and evaluation of Copyset Replication on two open source cloud storage systems, HDFS and RAMCloud, and shows 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%.

The dissertation also demonstrates that with any replication scheme (including Copyset Replication), using two replicas is sufficient for protecting against independent node failures within a cluster, while using three replicas is inadequate for protecting against correlated node failures. Given that in many storage systems the third or n-th replica was introduced for durability and not for performance, storage systems can change the placement of the last replica to address correlated failures, which are the main vulnerability of cloud storage systems.

The dissertation presents Tiered Replication, a replication scheme that splits the cluster into a primary and backup tier. The first two replicas are stored on the primary tier and are used to recover data in the case of independent node failures, while the third replica is stored on the backup tier and is used for correlated failures. The key insight behind Tiered Replication is that, since the third replicas are rarely read, we can place the backup tier on separate physical infrastructure or a remote location without affecting performance. This separation significantly increases the resilience of the storage system to correlated failures and presents a low cost alternative to geo-replication of an entire cluster. In addition, the Tiered Replication algorithm optimally minimizes the probability of data loss under correlated failures. Tiered Replication can be executed incrementally for each cluster change, which allows it to supports dynamic environments where nodes join and leave the cluster, and it facilitates additional data placement constraints required by the storage designer, such as network and rack awareness. Tiered Replication was implemented on HyperDex, an open-source cloud storage system, and the dissertation demonstrates that it incurs a small performance overhead. Tiered Replication improves the cluster-wide MTTF by a factor of 100,000 in comparison to random replication, without increasing the amount of storage.

Acknowledgements

When I first visited Stanford as a prospective grad student, I was deliberating between doing an MBA and a PhD. It initially seemed like a no-brainer: would I prefer spending long nights partying in exotic locations with circus tigers and fountains of champagne, or sleepless nights in a dank room alone with a computer and a flickering table lamp? Being no fan of masochism, I was convinced I would choose the former.

However, after meeting several Stanford professors, I quickly realized that the opportunity to work with world-class intelligent and creative technologists on cutting-edge research was simply too hard to resist. I realized that a PhD in Stanford is like being mentored by Michael Jackson to throw jump shots or Eminem in freestyle rap. It's a one-in-a-lifetime opportunity. I haven't looked backed since.

First, I'd like to thanks my primary advisors, Mendel Rosenblum and Sachin Katti for providing immense mentorship and support. I have learned valuable life lessons from you both.

Mendel has an extraordinarily lucid thought process. I could be spending hours working on a technical problem or positioning a paper, and in the first 5 minutes of the meeting Mendel would suddenly have a really simple insight that would leave me dumbfounded at how I could spend so much time over-complicating the topic at hand. He is incredibly creative and his depth of understanding of computer systems and technology is truly astonishing. Mendel has also been a personal mentor to me and I greatly enjoyed his advice and insights on my personal and professional life.

Sachin is hands-down the most creative person I have ever worked with. The frequency and quality of the ideas he produces is amazing. He is also incredibly sharp, and can grasp and express very broad concepts across very different domains. As a testament to this, I worked with Sachin on research in physical-level wireless communications, mobile software and distributed storage systems. Sachin is a pleasure to interact with on a personal level, and I feel I not only gained an outstanding advisor and mentor, but also a friend for life.

I was extremely fortunate to work with several other professors during my PhD. I had the pleasure to work with John Ousterhout on brainstorming and designing the concept of copysets. I feel like John is a legendary role-model for any computer scientist and I learned a lot from working with him. I had the pleasure to work with

Christos Kozyrakis in the early stages of my PhD. Christos is extremely smart and versatile in his research expertise. He is also a very funny and supportive person, and made the early steps in my PhD much easier and more enjoyable.

One of the greatest pleasures of the PhD program is to work closely with other incredibly talented students. It's a humbling experience to sit in the same room with such sharp colleagues. I worked with Ryan Stutsman and Stephen Rumble on implementing Copyset Replication. Ryan is incredible smart and very funny. I learned a lot from Ryan about politics and theology and really enjoyed his company. I'm sure he is an amazing advisor. Steve is a really talented computer systems researcher and programmer. He is sharp-witted, bright and humble and it was a pleasure working with him. Diego Ongaro is a great friend and has also been a frequent sounding board for my ideas. He is one of the best computer systems engineers I have ever met. He is like a one man software engineering team. I really enjoyed working with Kanthi Nagaraj on our wireless networking project, Flashback. Kanthi is really smart, hardworking, cheerful and a delight to work with. Kanthi gave me a crash course on wireless networking and programmable radios, which are topics I had almost no background in. Tomer London is one of my best friends. We were also research partners (and roommates) during our first year at Stanford. We started brainstorming research ideas even before we started grad school, and have been brainstorming startup ideas ever since. Tomer is incredibly talented, creative and smart.

I also greatly enjoyed collaborating with researchers outside of Stanford during my PhD. I greatly enjoyed working with Emin Gün Sirer and Robert Escriva from Cornell on Tiered Replication. It was a great experience working with Pramod Viswanath from UIUC on the Flashback paper. I learned a lot from the conversations with Pramod about communications and wireless networking. I had very enjoyable brainstorming sessions about storage coding and data placement ideas with Rashmi K. Vinayak from Berkeley.

I want to thank my family and friends. It seems like since the day I was born, my mom destined me to complete my PhD. Ima, I'm officially off the hook now. A secret sauce of my PhD was the immense personal support from my dad. Coaching me about academia, providing suggestions, emotional support, etc. etc. Aba, you

provided amazing support in the ups and downs of PhD life. My mom and dad are the most amazing parents I could ever wish for, and I really think I have had an unfair advantage in life with such incredible role models. My brother, Eyal is going to be a much better engineer and computer scientist than I am. His talent and humbleness are equally stellar. I love you and I have no doubt you will be extremely successful in whatever path you take. Eugenia, thanks for being incredibly supportive and being someone I can turn to whenever I have doubts or questions. I'm sure your dissertation will be way more intelligible and readable than this one. I'd like thank my grandparents, Savta Batya, Saba Yoske and Savta Rina. You are all inspirational to me, and I love you immensely.

Many other people have provided support and feedback for my work, including Jeff Mehlman, Steven Hong, Joseph Koo, Aditya Gudipati, Manu Bansal, Yan Michalevsky, Kiran Joshi, Dinesh Bharadia, Rakesh Misra, Aaron Schulman, David Gal, Lior Gavish, Ed Bugnion, Jacob Leverich, Christina Delimitrou, Daniel Sanchez, Ankita Kejriwal, Kannan Muthukkaruppan, Venkat Venkataramani, Shankar Pasupathy, Deepak Kenchammana, Robert Chansler, Dhruba Borthakur, Murray Stokely and Bernard Wong. I also want to thank Tsachy Weissman for supporting me in the early stages of my PhD and for chairing my oral defense committee.

Throughout my PhD I was supported by the Leonard J. Shustek Stanford Graduate Fellowship.

Contents

Acknowledgements **iii**

1 Introduction **1**

1.1 Introduction 2

1.2 Contributions 3

1.2.1 Model for Computing MTTF Across an Entire Cloud Storage Cluster 3

1.2.2 Copyset Framework and the Shortcomings of Random Replication 5

1.2.3 Copyset Replication and Tiered Replication 6

1.2.4 The Relationship of Copysets with BIBD 7

1.3 How to Read This Dissertation? 8

2 A Framework for Replication: Copysets and Scatter Width **9**

2.1 Introduction 10

2.2 The Problem 14

2.2.1 Definitions 14

2.2.2 Random Replication 15

2.3 Intuition 17

2.3.1 Minimizing the Number of Copysets 18

2.3.2 Scatter Width 19

2.4 Design 22

2.4.1 Durability of Copyset Replication 25

2.4.2 Optimality of Copyset Replication 29

2.4.3 Expected Amount of Data Lost 30
2.5 Evaluation 32
2.5.1 HDFS Implementation 32
2.5.2 HDFS Evaluation 33
2.5.3 Implementation of Copyset Replication in RAMCloud 35
2.5.4 Evaluation of Copyset Replication on RAMCloud 36
2.6 Discussion 38
2.6.1 Copysets and Coding 38
2.6.2 Graceful Power Downs 38
2.7 Related Work 40
2.7.1 Combinatorial Design Theory 40
2.7.2 DHT Systems 42
2.7.3 Data Center Storage Systems 42

3 The Peculiar Case of the Last Replica 44
3.1 Introduction 45
3.2 Motivation 48
3.2.1 Analysis of Independent Node Failures 48
3.2.2 Analysis of Correlated Node Failures 55
3.2.3 The Peculiar Case of the Nth Replica 56
3.3 Design 59
3.3.1 Analysis of Tiered Replication 63
3.3.2 Dynamic Cluster Changes 65
3.3.3 Additional Constraints 66
3.3.4 Analysis of Additional Constraints 67
3.4 Implementation 69
3.4.1 Performance Benchmarks 70
3.4.2 Write Latency 71
3.4.3 Recovery Evaluation 71
3.5 Related Work 72
3.6 Appendix 74

4 Conclusions **75**

4.1 Conclusions 76

Bibliography **78**

List of Tables

2.1 Replication schemes of data center storage systems. These parameters are estimated based on publicly available data [10, 75, 8, 3, 60]. For simplicity, we fix the HDFS scatter width to 200, since its value varies depending on the cluster and rack size. 14

2.2 Comparison of recovery time of a 100 GB node on a 39 node cluster. Recovery time is measured after the moment of failure detection. . . . 33

2.3 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. 34

2.4 Comparison of backup recovery performance on RAMCloud with Copyset Replication. Recovery time is measured after the moment of failure detection. 37

3.1 Probability of simultaneous node failures due to independent node failures under different cluster sizes. The model uses $S = 10$, $R = 3$, an average node MTTF of 10 years and a node recovery time of 3 minutes. 51

3.2 Tiered Replication algorithm's variables and helper functions. 59

List of Figures

2.1 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 [75, 8, 10, 60] (see Table 2.1). 11

2.2 Simulation of the data loss probabilities of a RAMCloud cluster, varying the number of replicas per chunk. 16

2.3 Data loss probability when 1% of the nodes fail simultaneously as a function of S, using $N = 5000$, $R = 3$. 21

2.4 Illustration of the Copyset Replication Permutation phase. 22

2.5 Illustration of the Copyset Replication Replication phase. 23

2.6 Data loss probability of random replication and Copyset Replication with $R = 3$, using the parameters from Table 2.1. HDFS has higher data loss probabilities because it uses a larger scatter width ($S = 200$). 24

2.7 Data loss probability of random replication and Copyset Replication in RAMCloud. 25

2.8 Data loss probability of random replication and Copyset Replication in HDFS. 26

2.9 Data loss probability of random replication and Copyset Replication in Facebook's implementation of HDFS. 27

2.10 Data loss probability on Facebook's HDFS cluster, with a varying percentage of the nodes failing simultaneously. 28

2.11 Comparison of the average scatter width ofCopyset Replication to the optimal scatter width in a 5000-node cluster. 29

2.12 Expected amount of data lost as a percentage of the data in the cluster. 30

3.1 Markov chain of data loss due to independent node failures. Each state represents the number of nodes that are down simultaneously. 49
3.2 MTTF due to independent and correlated node failures of a cluster with a scatter width of 10. 52
3.3 MTTF due to independent and correlated node failures of a cluster with 4000 nodes. 53
3.4 MTTF due to independent and correlated node failures of a cluster with a scatter width of 10. 61
3.5 MTTF due to independent and correlated node failures of a cluster with a scatter width of 10. 62
3.6 Adding constraints on Tiered Replication increases the number of copy-sets, on a cluster with $N = 4000$, $R - 3$ and $S = 10$. 66
3.7 Tiered Replication throughput under YCSB benchmark. Each bar represents the throughput under a different YCSB workload. 70
3.8 Tiered Replication write latency under YCSB benchmark. 71

Chapter 1

Introduction

1.1 Introduction

Cloud storage systems are used by web service providers, such as Google, Facebook and Amazon, to centrally store, process and backup massive amounts of data. These storage systems store petabytes of data across thousands of servers. Such storage systems are utilized to store web mail attachments, search engine indexes and social network profile data.

Unlike traditional storage systems, which guard against data loss by utilizing expensive fault-tolerant hardware systems, cloud storage systems use a smart distributed software layer to handle data replication and recovery, across a large number of inexpensive commodity servers that frequently fail.

The unprecedented scale of these storage systems coupled with the need to tolerate frequent commodity server hardware failures, create novel research questions about how to effectively guard against node failurs. The dissertation provides two motivating examples for such research questions.

First, cloud storage system operators, such as Yahoo! [75], LinkedIn [10] and Facebook [8] have noticed that once the cluster scales beyond several thousands of nodes, large scale correlated failures, such as power outages or large network failures, are almost guaranteed to cause data loss. This type of data loss incident only occurs once these storage system reach a scale of thousands of nodes.

Second, cloud storage systems use data replication in order to guard against data loss. When designing cloud storage systems, many storage designers need to answer a simple question: how many replicas would be sufficient in order to protect against data loss? In order to answer this question, we need to understand the underlying causes of data loss in cloud storage systems, and how the number of replicas and the placement of the replica would affect the mean time to failure (MTTF) of the entire storage system.

Motivated by these problems and other common issues encountered by such systems, this dissertation propose a novel framework for modeling and analyzing node failures in cloud storage systems. Based on this framework, it provides a design and implementation of two novel replication techniques, Copyset Replication and Tiered

Replication, that optimally protect against data loss in storage systems that span any number of nodes.

1.2 Contributions

At a high level, this dissertation demonstrates that widely used replication and data placement techniques are not tuned to the failure scenarios that occur in modern cloud storage systems, and provides a framework for analyzing node failures and techniques that optimally address these failures. The dissertation presents implementations of these techniques and demonstrates that they offer much higher resilience at the same or at lower cost than existing replication schemes. The specific contributions of this dissertation are described below.

1.2.1 Model for Computing MTTF Across an Entire Cloud Storage Cluster

An important contribution of this dissertation is that it is the first to analyze the MTTF (Mean Time To Failure) of an entire cloud storage cluster.

In the distributed storage community, node failures are traditionally classified into two types of failures: independent and correlated node failures [8, 10, 28, 6, 58, 87]. For many years, researchers have been analyzing the MTTF of a single node due to independent node failures, by analyzing the fault tolerance of disk systems [70, 71, 57, 80, 74, 32, 62, 88, 30]. There is a much more limited body of work on analyzing correlated failures in a cloud storage setting, with thousands of commodity servers. There are several prior studies on smaller clusters, consisting of hundreds of nodes [59, 40, 61, 68, 81, 86], but very few studies on modern web-scale storage systems. Google researchers have studied the MTTF of a single piece of data in a cloud storage clustered environment, which is primarily affected by correlated failures [28, 31, 19]. Similarly, LinkedIn released a report about the availability of their HDFS cluster under correlated and independent node failures [10].

The problem with these prior models is that they only allow storage designers to

analyze the MTTF from the point of view of a single node or a single data chunk. Knowing the MTTF of a single node or chunk does not allow storage designers to determine the MTTF of an entire cluster. To illustrate the difference between the MTTF of a single data chunk and the MTTF of an entire cluster, consider the following example. Assume a chunk has an MTTF of 100 years. If we have a cluster with 100 chunks across several nodes, the cluster's MTTF would be 100 years if all the data chunks fail at the exactly same time. The cluster-wide MTTF could also be 1 year (i.e., 100 times lower), if each chunk fails at different intervals. The cluster-wide failure model (i.e., the correlation and frequency of node failures), and the placement of data across the cluster would explain the differences in the cluster-wide MTTF.

This dissertation is the first to provide frameworks for computing the MTTF for an entire cluster, or in other words, calculating how frequently data loss occurs across an entire data center with thousands of nodes. It is important for storage designers to be able to model and understand the MTTF and of an entire cluster, because it affects the overall reliability and availability of the applications utilizing the storage system. Some applications may lose availability at every occurance of data loss. In addition, since each data loss event in a cloud storage system might incur a high fixed cost, because the data loss incident necessitates manual location and recovery of data, some storage designers prefer to minimize the overall MTTF of the cluster, even if each failure incident event may result in a higher average data loss [54].

The dissertation provides a relatively simple model for calculating the cluster-wide MTTF for both independent and correlated node failures. It shows how the parameters of the cluster, including the node recovery time, replication technique and MTTF of each individual machine affect the overall MTTF of the cluster. It also qualitatively demonstrates why both Google and LinkedIn have found that correlated node failures are a much greater cause of data loss in cloud storage systems than independent node failures.

1.2.2 Copyset Framework and the Shortcomings of Random Replication

Based on the cluster-wide MTTF model, the dissertation demonstrates that data placement and replication is one of the most important tools for storage system designers to control the cluster-wide MTTF. Prior work has shown that random replication suffers from a high rate of data loss under correlated failures [3, 75, 10]. The dissertation provides a framework that helps explain why random replication is prone to a high probability of data loss, and furthermore shows that random replication, which is used by an overwhelming majority of cloud storage systems, is a very poor data placement policy for minimizing cluster-wide MTTF.

The work shows that in order to minimize cluster-wide MTTF, storage system designers must minimize the number of sets of nodes containing all replicas of data chunks, or, minimize the number of copysets. This dissertation demonstrates that the number of copysets is one of the most important factors determining cluster-wide MTTF. It shows that random replication, or other schemes that uniformly scatter replicas across the cluster, such as consistent hashing based techniques [41, 79, 47, 20, 18, 67, 65], will produce the maximum number of copysets if the number of chunks in the storage systems is very high.

The work also introduces the concept of scatter width, which is the number of candidate nodes that can store each node's replica. Scatter width is a function of node recovery time, because it determines how many nodes will participate in recovering the data of a failed node. This work describes the relationship between copysets and scatter width, and shows that with random replication the number of copysets increases as a polynomial function of the scatter width.

The thesis provides a benchmark for the optimal relationship between the number of copysets and the scatter width, and shows that in optimal schemes the number of copysets increases as a linear function of the scatter width.

1.2.3 Copyset Replication and Tiered Replication

The thesis presents the design of two new practical replication schemes, Copyset Replication and Tiered Replication, that unlike randomized replication schemes, minimize the number of copysets for a given scatter width.

Copyset Replication is a novel scheme that allows the storage system designer to constrain the number of copysets, as a function of the scatter width, within a single cluster. If the storage system requires increased scatter width, Copyset Replication will create a new permutation of copysets. The work demonstrates that Copyset Replication achieves a near-optimal linear relationship between the number of copysets and the scatter width.

Copyset Replication is implemented on two open-source systems, HDFS and RAMCloud, and the work demonstrates that it causes a minimal overhead on normal performance, while reducing the probability of data loss under correlated failures by orders of magnitude. 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%.

Copyset Replication is only focused on replication within the same cluster and physical location. However, many storage system designers employ geo-replication as a technique to further increase the MTTF of the storage system, by replication an entire cluster's data to a second, remote cluster [28, 52, 24, 49, 51, 45, 91, 4]. This effectively doubles the storage system's storage size.

Tiered Replication is a novel scheme that provides almost the same level of durability of geo-replication, at a greatly reduced price. The work demonstrates that in practical settings, the last (typically the third) replica is not necessary for preventing data loss due to independent node failures, due to the very low probability that independent node failures will affect several nodes simultaneously. Therefore, the last replica can be utilized for "geo-replication" of a single replica, or in other words, the last replica can be placed on a separate or remote data center.

In addition, unlike previous random replication schemes, Tiered Replication also minimizes the number of copysets per scatter width, by greedily creating unique

replication sets that have very few overlaps. Therefore, by combining light-weight geo-replication with a minimal number of copysets, Tiered Replication improves the cluster-wide MTTF by a factor of 100,000 in comparison to random replication and by a factor of 100 compared to Copyset Replication, without increasing the amount of storage.

The work presents the implementation of Tiered Replication on HyperDex, an open-source cloud storage system, and shows that it incurs a minimal overhead on normal storage operations. In addition, since Tiered Replication relies on an incremental greedy algorithm, it has better support for dynamic cluster changes and network topology constraints than Copyset Replication.

The idea of leveraging the high MTTF under independent node failures to place the last replica on a separate storage infrastructure is novel, and unlike traditional geo-replication does not double the storage cost of the system. This idea can is widely applicable to a wide variety of storage systems.

1.2.4 The Relationship of Copysets with BIBD

Both Copyset Replication and Tiered Replication provide a linear relationship between the number of copysets and the scatter width. However, both of these replication techniques do not provide an optimally minimal number of copysets.

This dissertation is the first to demonstrate that BIBD (Balanced Incomplete Block Designs), a field of combinatorial theory, can be a potential technique to optimally solve the copyset minimization problem (and also optimally minimze the cluster-wide MTTF) for a given scatter width. BIBD schemes have been used for a variety of applications in the past, including agriculture experiments [21, 16, 89], social sciences [85, 26], RAID storage systems [35, 73] and network fabric interconnects [36, 55]. The dissertation shows that finding the minimal number of copysets for a scatter width that is exactly equal to the number of nodes in the cluster, is similar to a BIBD scheme with a λ parameter equal to 1.

Since most practical cloud storage systems require a scatter width that is smaller than the number of nodes in the cluster, existing BIBD schemes do not allow us to

construct practical replication techniques. Existing BIBD schemes only provide integer λ values. Therefore, this dissertation introduces motivation for a new promising area of potential future theoretical research, of creating BIBD schemes with λ value that are smaller than 1.

1.3 How to Read This Dissertation?

The chapters in this dissertation are ordered from a pedagogical point of view. It starts with Copyset Replication (based on work published in Usenix ATC 2013 [14]), described in Chapter 2, which describes the copyset framework, provides a model for analyzing correlated failures and introduces Copyset Replication, a simple technique to minimize the number of copysets for any scatter width. Tiered Replication, described in Chapter 3 (based on work published in Usenix ATC 2015 [13]), extends the failure model for independent node failures, and provides techniques to further increase the MTTF under correlated failures, by geo-replicating a single replica (instead of an entire cluster). Finally, in Chapter 4, the results of the dissertation are summarized and the chapter provides an outlook for future work on cloud storage durability.

Chapter 2

A Framework for Replication: Copysets and Scatter Width

2.1 Introduction

Random replication is used as a common technique by data center storage systems, such as Hadoop Distributed File System (HDFS) [75], RAMCloud [60], Google File System (GFS) [29] and Windows Azure [9] 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 [19, 28, 75, 10], are handled poorly by random replication. This scenario stresses the availability of the system because a non-negligible percentage of nodes (0.5%-1%) [75, 10] 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 2.1 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! [75], LinkedIn [10] and Facebook [8]. 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" [54]. Other data center operators have reported similar experiences [11].

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

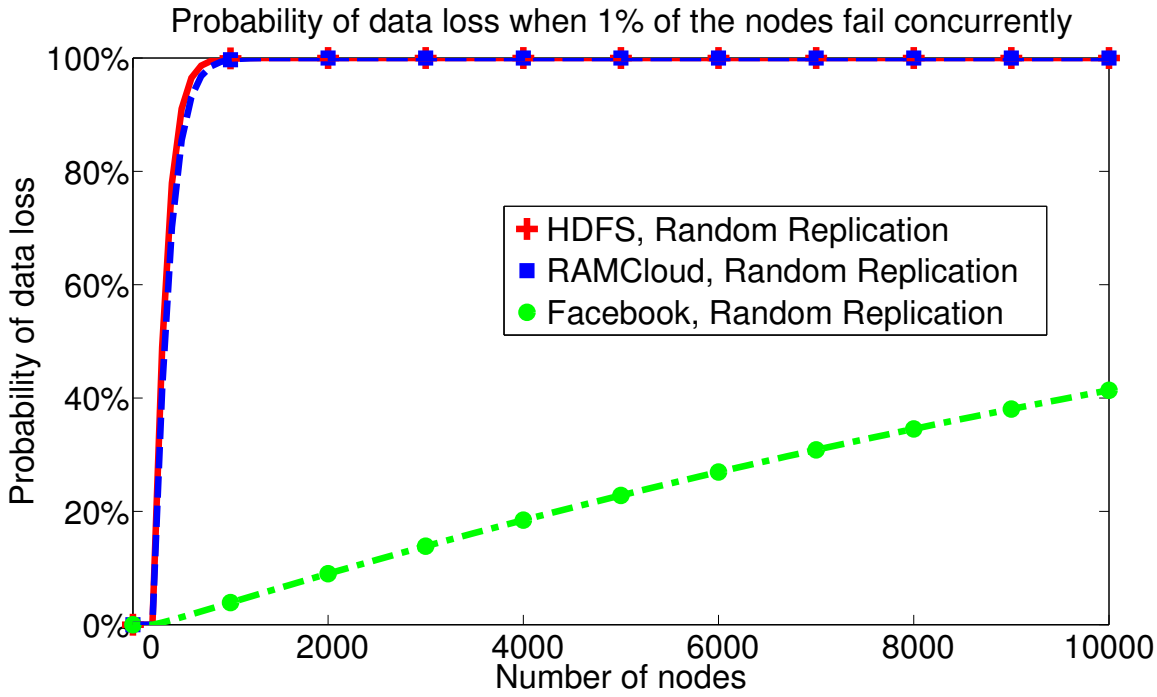

Figure 2.1: Computed probability of data loss with $R = 3$ when 1% of the nodes do not survive a power outage. The parameters are based on publicly available sources [75, 8, 10, 60] (see Table 2.1).

losses since they are less likely to tax storage nodes and fabric with spikes in data reconstruction traffic. Other 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." [62]

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 dissertation 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 creates 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 [78].

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 dissertation is split into the following sections. Section 2.2 presents the problem. Section 2.3 provides the intuition for our solution. Section 3.3 discusses the design of Copyset Replication. Section 2.5 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 2.6, while Section 3.5 analyzes related work.

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 2.1: Replication schemes of data center storage systems. These parameters are estimated based on publicly available data [10, 75, 8, 3, 60]. For simplicity, we fix the HDFS scatter width to 200, since its value varies depending on the cluster and rack size.

2.2 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.

2.2.1 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. 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 2.1 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].

2.2.2 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 [75, 28] [2]. These failures happen frequently (thousands of times a year on a large cluster [19, 28, 10]), 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 [19, 28, 10].

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 [75, 10], 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

[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 [75].). We have found there is little difference in terms of data loss probabilities between the different schemes.

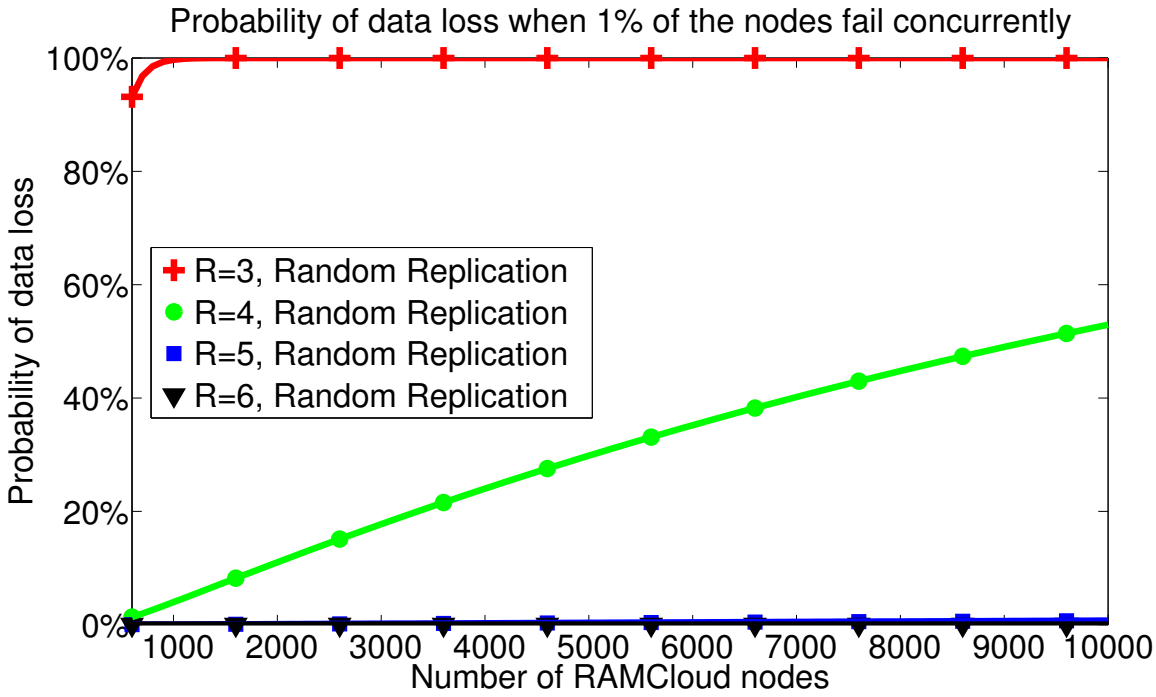

Figure 2.2: Simulation of the data loss probabilities of a RAMCloud cluster, varying the number of replicas per chunk.

reboot [75, 10]. Such failures are not uncommon; they occur once or twice per year in a given data center [10].

Figure 2.1 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 [54, 11, 62]. 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 [8, 3]. Facebook's Tech Lead of the HBase engineering team has confirmed this point, as cited above [54]. Robert Chansler, Senior Manager of Hadoop Infrastructure at Linkedin has also confirmed the importance of addressing this issue: "A power-on restart 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*". [11]

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 [54]. 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*". [11]

One trivial alternative for decreasing the probability of data loss is to increase R. In Figure 2.2 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.

2.3 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 2.1 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.

2.3.1 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.

2.3.2 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 2.3 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.

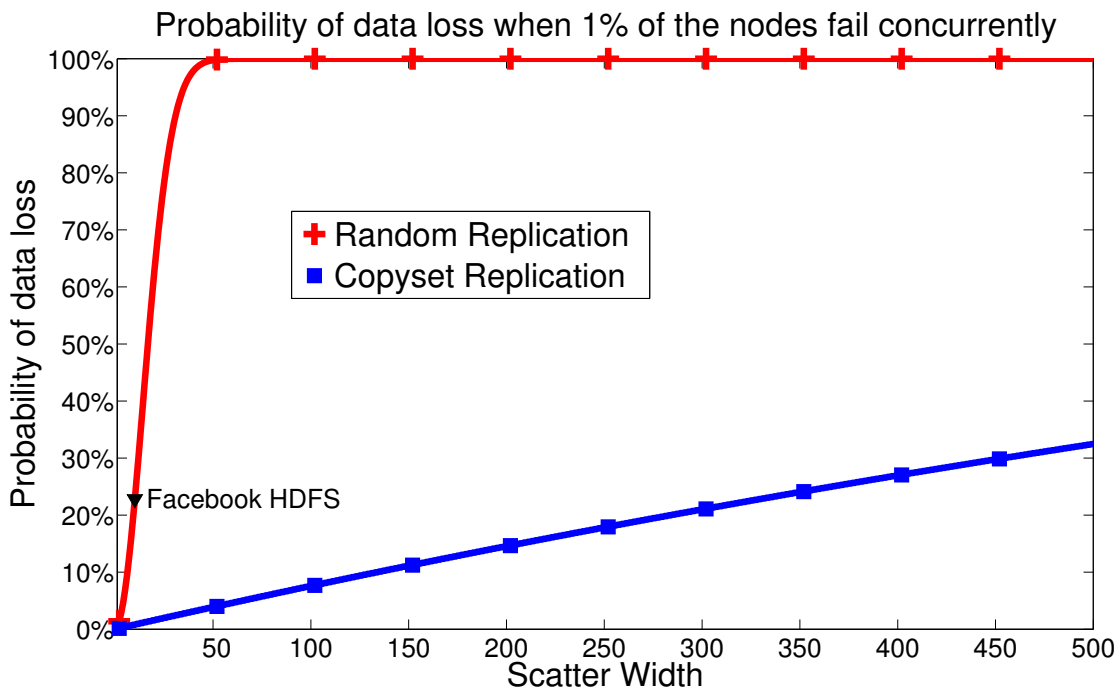

Figure 2.3: Data loss probability when 1% of the nodes fail simultaneously as a function of S, using $N = 5000$, $R = 3$.

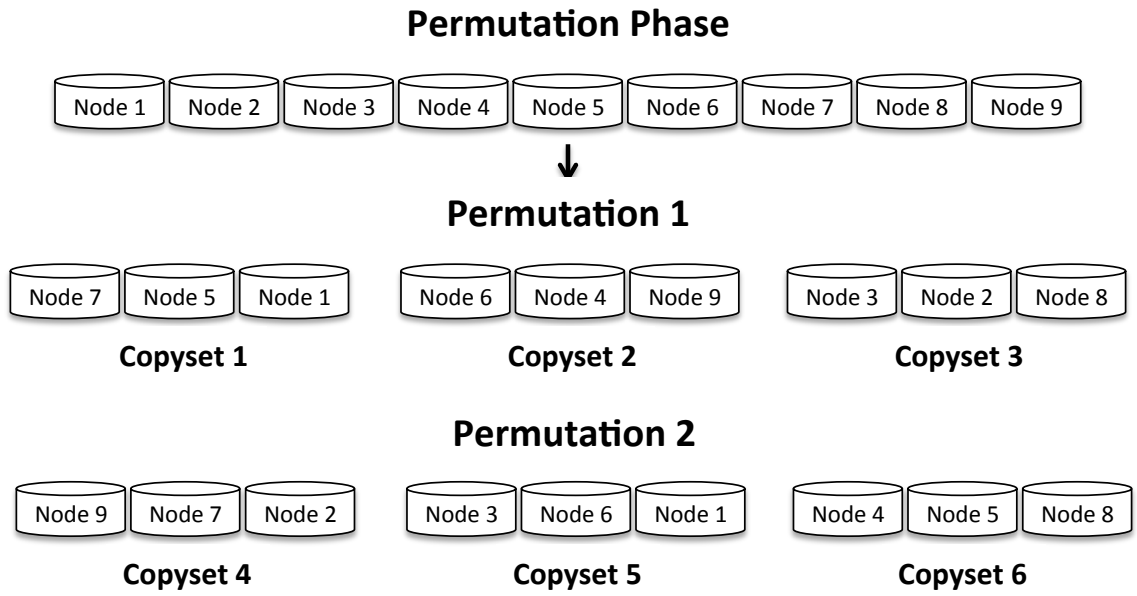

Figure 2.4: Illustration of the Copyset Replication Permutation phase.

2.4 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 [37, 42]. For a more detailed theoretical discussion see Section 2.7.1.

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.

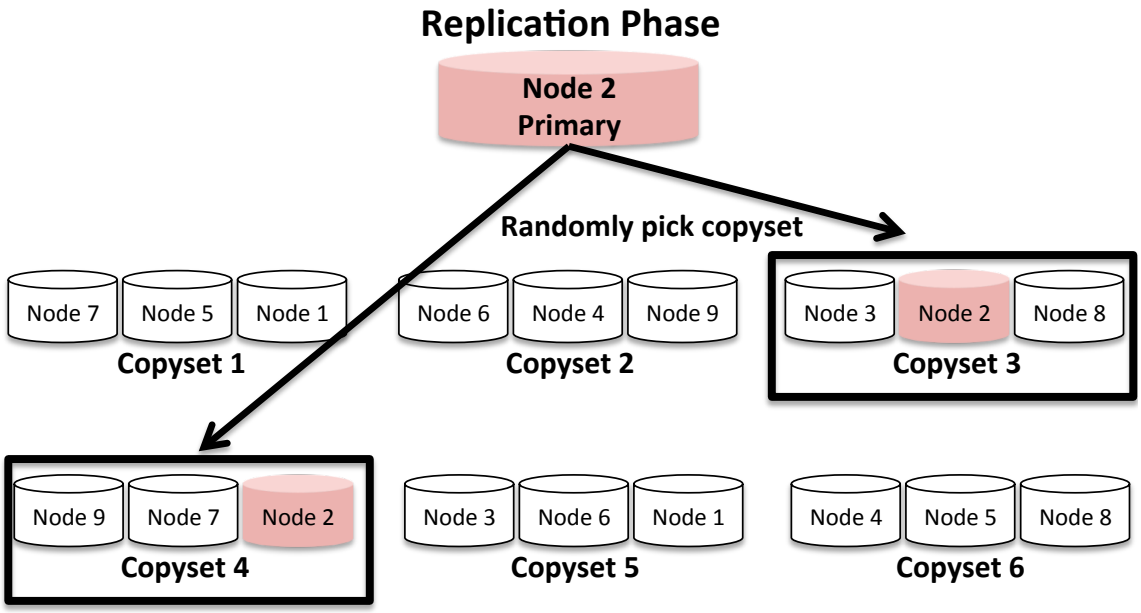

Figure 2.5: Illustration of the Copyset Replication Replication phase.

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 2.4 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 2.5) 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

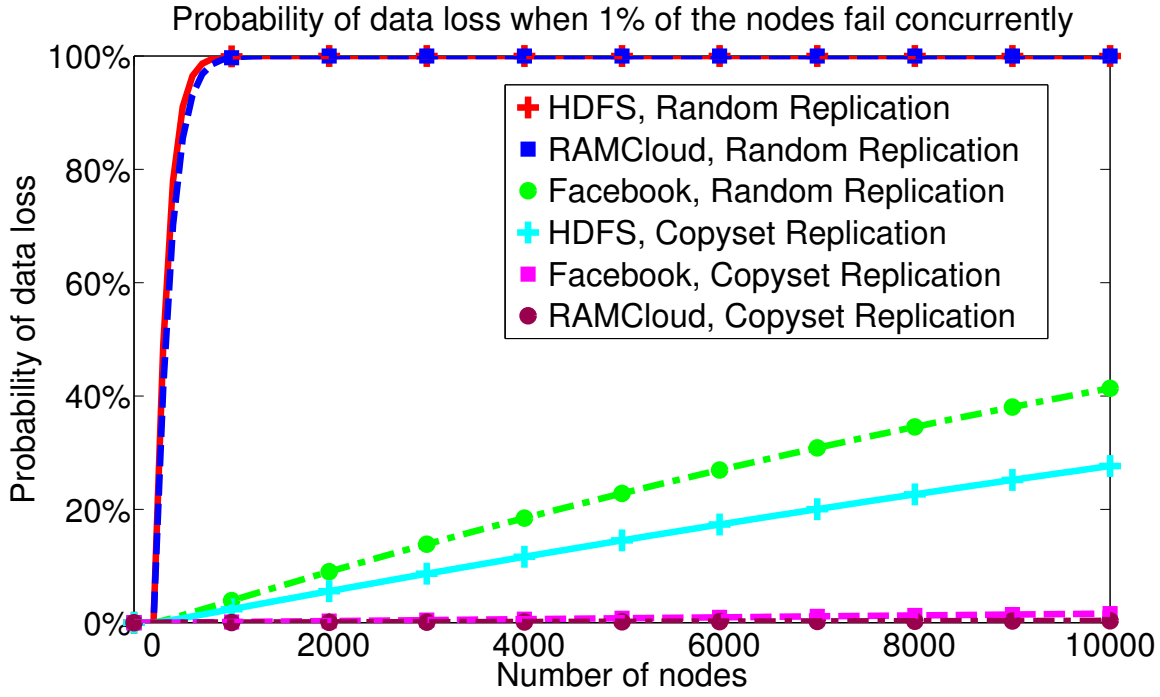

Figure 2.6: Data loss probability of random replication and Copyset Replication with $R = 3$, using the parameters from Table 2.1. HDFS has higher data loss probabilities because it uses a larger scatter width ($S = 200$).

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 [56]. 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.

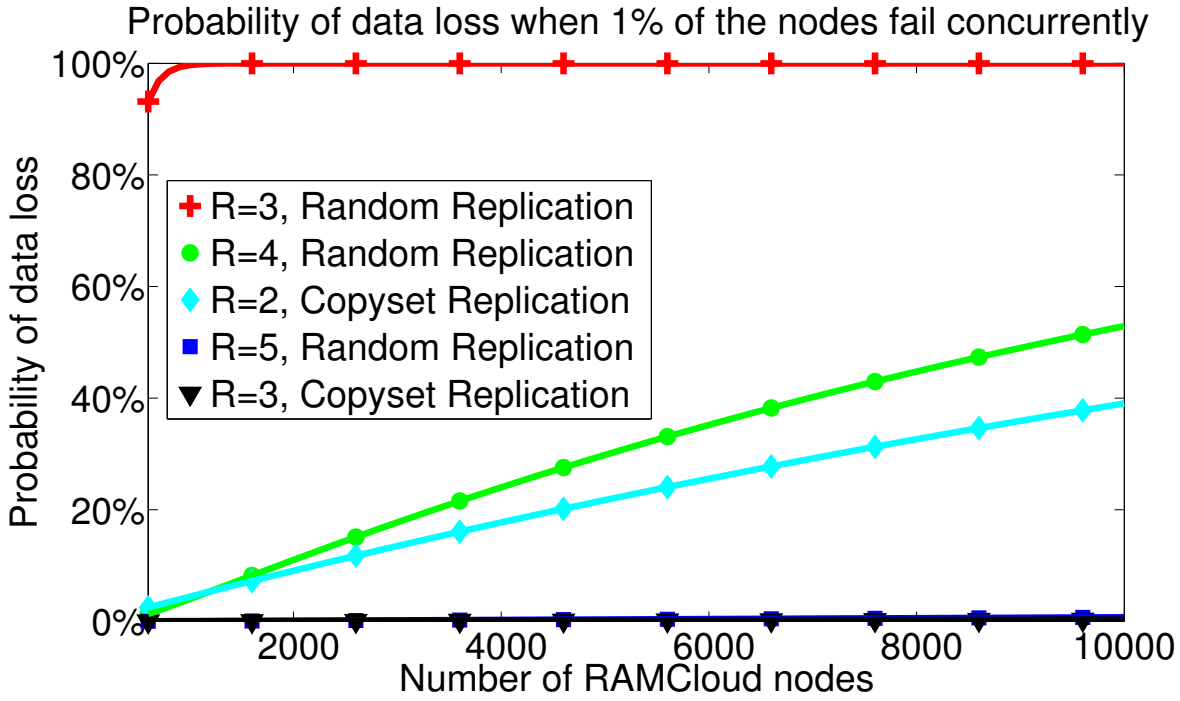

Figure 2.7: Data loss probability of random replication and Copyset Replication in RAMCloud.

2.4.1 Durability of Copyset Replication

Figure 2.6 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

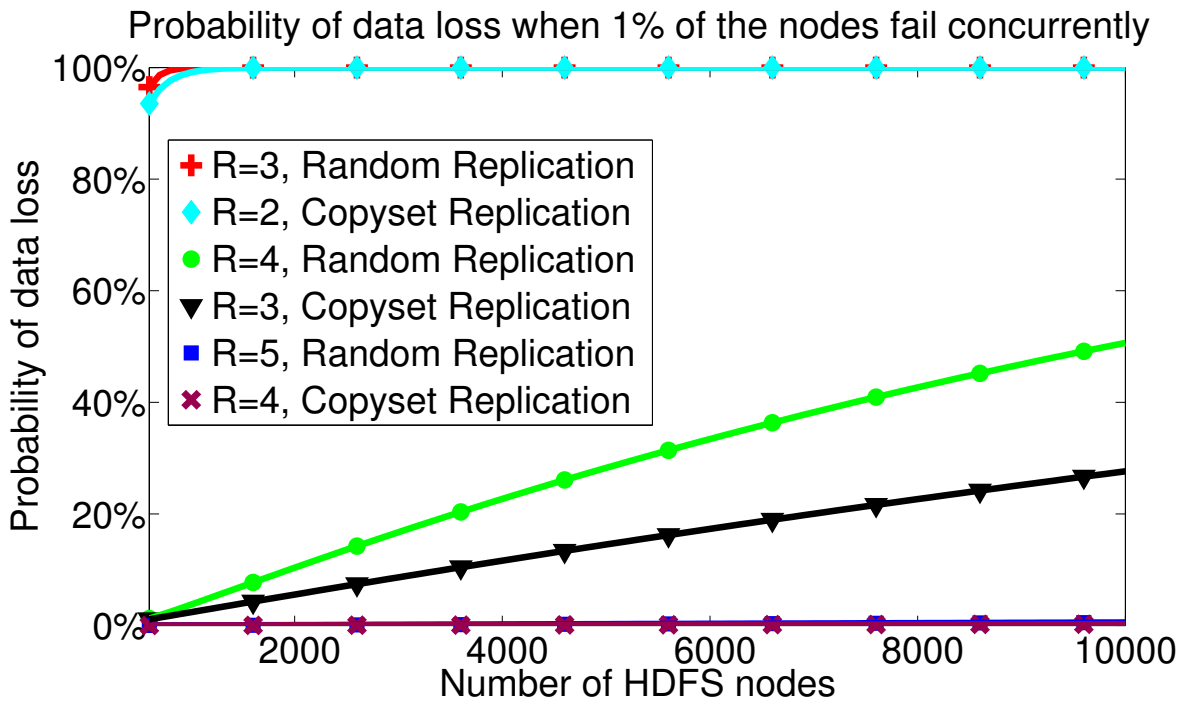

Figure 2.8: Data loss probability of random replication and Copyset Replication in HDFS.

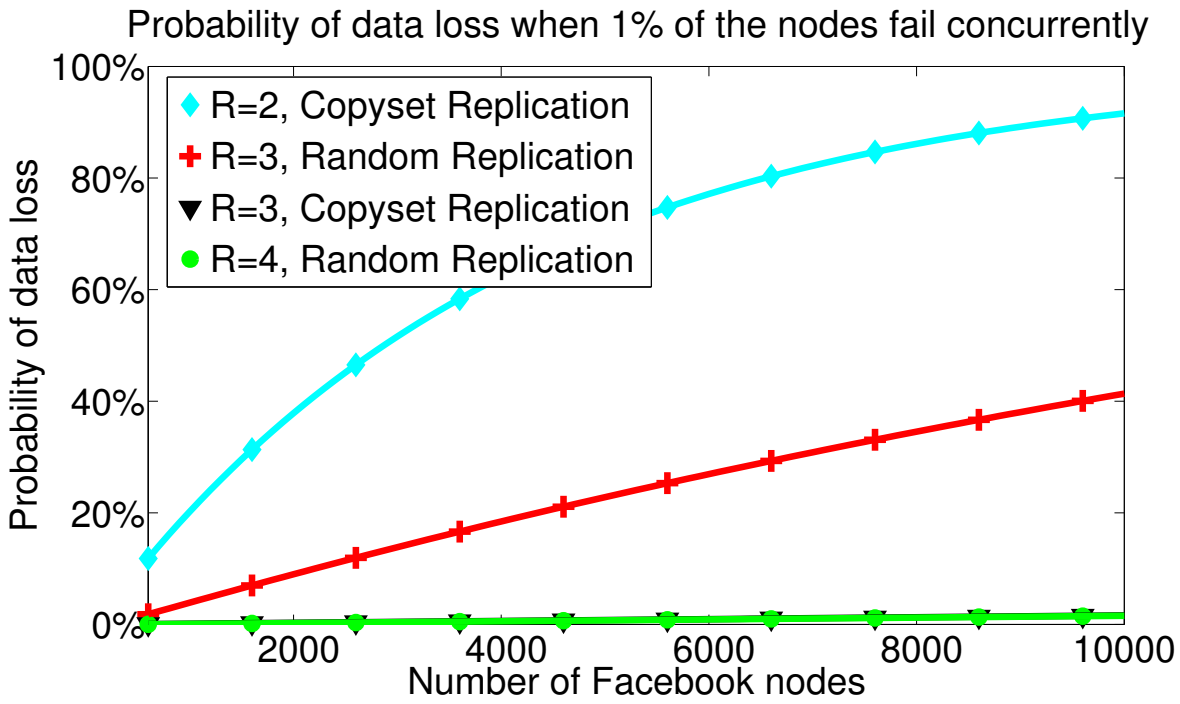

Figure 2.9: Data loss probability of random replication and Copyset Replication in Facebook's implementation of HDFS.

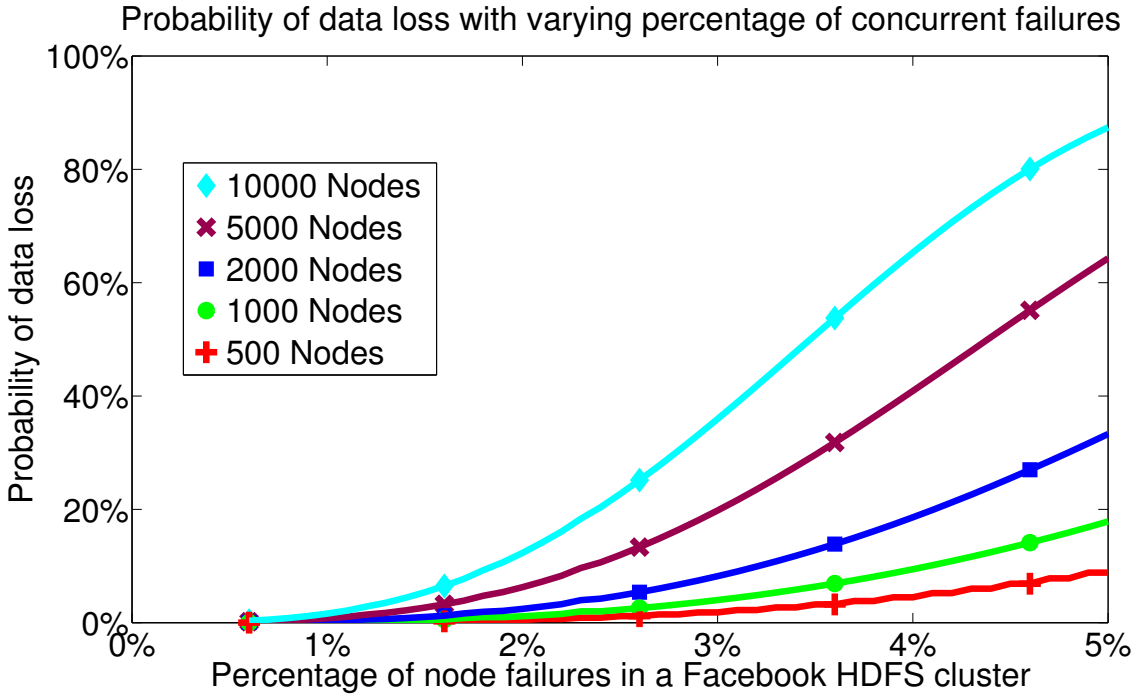

Figure 2.10: Data loss probability on Facebook's HDFS cluster, with a varying percentage of the nodes failing simultaneously.

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.

Figures 2.7, 2.8 and 2.9 depict 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 [75]. Figure 2.10 depicts the probability of data loss

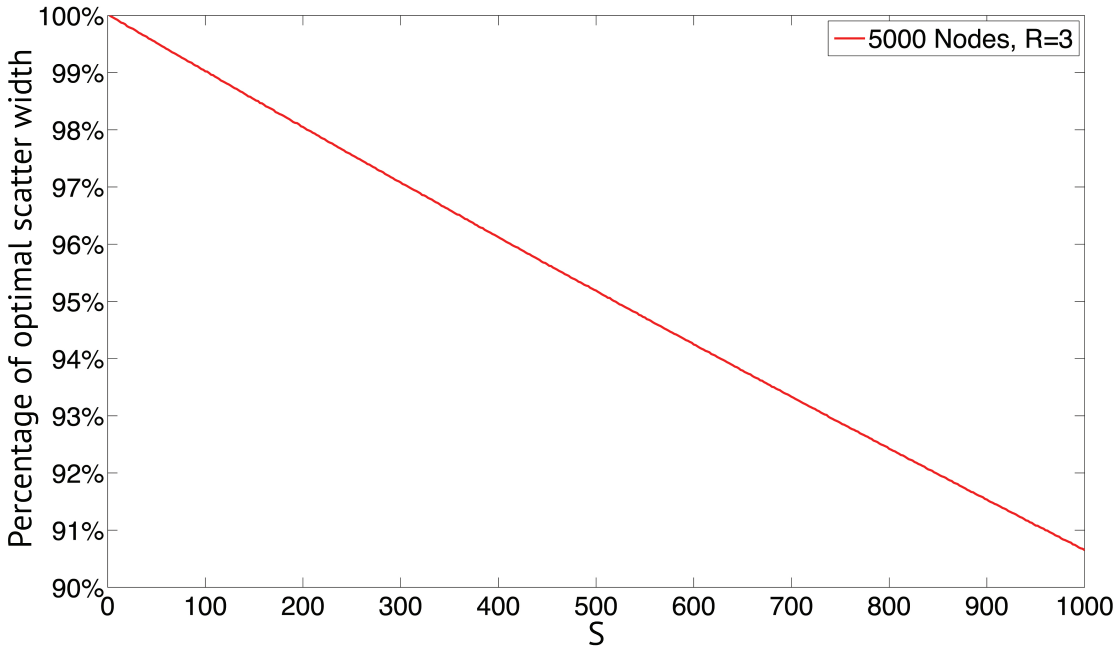

Figure 2.11: Comparison of the average scatter width ofCopyset Replication to the optimal scatter width in a 5000-node cluster.

in Facebook'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 2.1). For these cluster sizes Copyset Replication prevents data loss with a high probability, even in the scenario where 2% of the nodes fail simultaneously.

2.4.2 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 2.11 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.

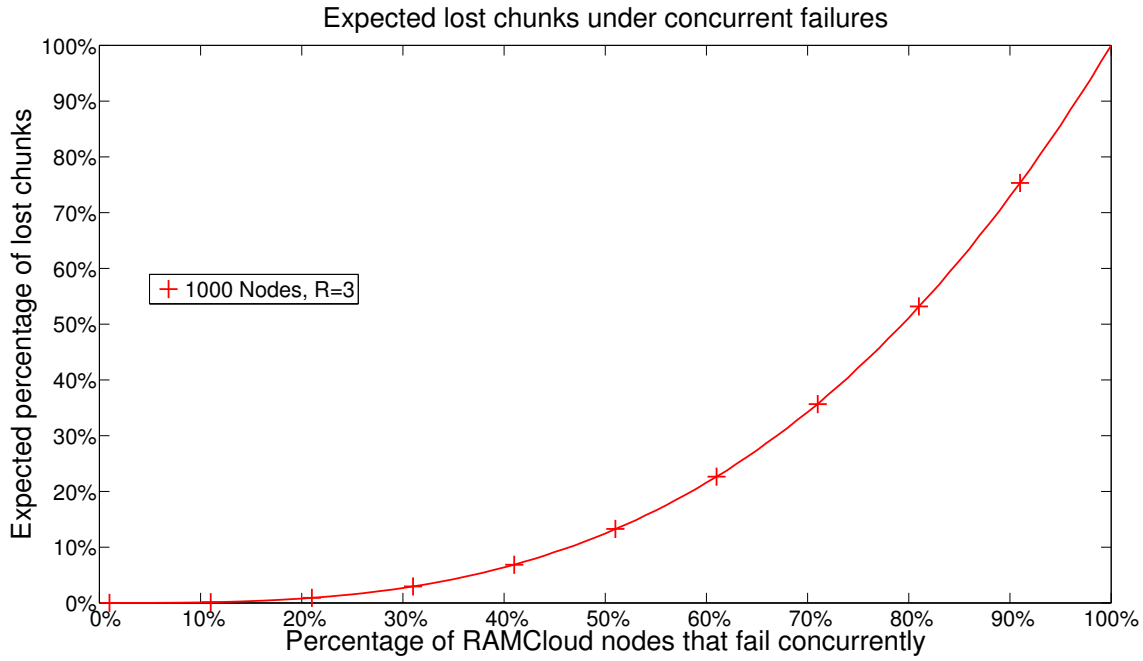

Figure 2.12: Expected amount of data lost as a percentage of the data in the cluster.

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 $S = 192$.

2.4.3 Expected Amount of Data Lost

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 2.12 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.

2.5 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.

2.5.1 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.

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.

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

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.

2.5.2 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 2.2 compares the recovery time of a single node using Copyset Replication and random replication. We ran each recovery five times.

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.

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 2.3: 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.

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).

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 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 2.3 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.

2.5.3 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.

2.5.4 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.

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.

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

Table 2.4: Comparison of backup recovery performance on RAMCloud with Copyset Replication. Recovery time is measured after the moment of failure detection.

We ran this test 100 times, both with Copyset Replication and random replication. As 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.

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 2.4 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.

2.6 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.

2.6.1 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, 25] 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.

2.6.2 Graceful Power Downs

Data center operators periodically need to gracefully power down parts of a cluster [7, 19, 28]. 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 [48, 34, 83].

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.

2.7 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.

2.7.1 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 [27]. 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 [78].

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 [43]. The Experimental Design Handbook has an extensive selection of design examples [15].

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 [37, 42].

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.

2.7.2 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. [12] 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. [90] 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 [33] 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.

2.7.3 Data Center Storage Systems

Facebook's proprietary HDFS implementation constrains the placement of replicas to smaller groups, to protect against concurrent failures [8, 3]. Similarly, Sierra randomly places chunks within constrained groups in order to support flexible node power downs and data center power proportionality [83]. 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 [28] 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 [54, 11].

Chapter 3

The Peculiar Case of the Last Replica

3.1 Introduction

Popular cloud storage systems like HDFS [75], GFS [29] and Azure [9] typically replicate their data three times to guard against data loss. The common architecture of cloud storage systems is to split each node's storage into data chunks and replicate each chunk on three randomly selected nodes.

The conventional wisdom is that replicating chunks three times is essential for preventing data loss due to node failures. In prior literature, node failure events are broadly categorized into two types: independent node failures and correlated node failures [8, 10, 28, 6, 58, 87]. Independent node failures are defined as events where nodes fail individually and independently in time (e.g., individual disk failure, kernel crash). Correlated failures are defined as failures where several nodes fail simultaneously due to a common root cause (e.g., network failure, power outage, software upgrade). In this dissertation, we are primarily concerned with events that affect data durability rather than data availability, and are therefore concerned with node failures that cause permanent data loss, such as hardware and disk failures, in contrast to transient data availability events, such as software upgrades.

This dissertation questions the assumption that traditional three-way replication is effective to guard against all data loss events. We show that, while a replication factor of three or more is essential for protecting against data loss under correlated failures, a replication factor of two is sufficient to protect against independent node failures.

We note that, in many storage systems, the third or n-th replica was introduced mainly for durability and not for read performance [10, 76, 12, 23]. Therefore, we can leverage the last replica to address correlated failures[1], which are the main cause of data loss for cloud storage systems [28, 58].

We demonstrate that in a storage system where the third replica is only read when the first two are unavailable (i.e., the third replica is not required for operational data reads), the third replica would be read almost exclusively during correlated failure events. In such a system, the third replica's workload is write-dominated,

[1]Without loss of generality, this dissertation assumes an architecture where some replicas are used for performance, and others are used for preventing data loss.

since it would be written to during every system write, but very infrequently read from (almost exclusively in the case of a correlated failure).

This property can be leveraged by storage systems to increase durability and reduce storage costs. Storage systems can split their clusters into two tiers: the *primary tier* would contain the first and second copy of each replica, while the *backup tier* would contain the backup third replicas. The backup tier would only be used when data is not available in the primary tier. Since the backup tier's replicas will be read infrequently they do not require high performance for read operations. The relaxed read requirements for the third replica enable system designers to further increase storage durability, by storing the backup tier on a remote site (e.g., Amazon S3), which significantly reduces the correlation in failures between nodes in the primary tier and the backup tier. In addition, the backup tier may also be compressed, deduplicated or stored on a low-cost storage medium (e.g., tape) to reduce storage capacity costs.

Existing replication schemes cannot effectively separate the cluster into tiers while maintaining cluster durability. Random replication, the scheme widely used by popular cloud storage systems, scatters data uniformly across the cluster and has been shown to be very susceptible to frequent data loss due to correlated failures [14, 10, 3]. Non-random replication schemes, like Copyset Replication [14], have a significantly lower probability of data loss under correlated failures. However, Copyset Replication is not designed to effectively distribute the replicas into storage tiers, does not support nodes joining and leaving the cluster and does not allow the storage system designer to add additional placement constraints, such as support chain replication or requiring replicas to be placed on different network partitions and racks.

We present Tiered Replication, a simple dynamic replication scheme that leverages the asymmetric workload of the third replica, and can be applied to any cloud storage system. Tiered Replication allows system designers to divide the cluster into primary and backup tiers, and its incremental operation supports dynamic cluster changes (e.g., nodes joining and leaving). In addition, unlike random replication, Tiered Replication enables system designers to limit the frequency of data loss under correlated failures. Moreover, Tiered Replication can support any data layout constraint,

including support for chain replication [84] and topology-aware data placement.

Tiered Replication is an optimization-based data placement algorithm that places chunks into the best available replication groups. The insight behind its operation is to select replication groups that both minimize the probability of data loss under correlated failures by reducing the overlap between replication groups, and satisfy data layout constraints defined by the storage system designer. The storage system with Tiered Replication achieves an MTTF that is 10^5 greater than random replication, and more than 10^2 greater than Copyset Replication.

We implemented Tiered Replication on HyperDex, an open-source key-value cloud storage system [23]. Our implementation of Tiered Replication versatile enough to satisfy constraints on replica assignment and load balancing, including HyperDexs data layout requirements for chain replication [84]. We analyze the performance of Tiered Replication on a HyperDex installation on Amazon, where the backup tier, containing the third replicas, is stored on a separate Amazon availability zone. We show that Tiered Replication incurs a small performance overhead for normal operations and preserves the performance of node recovery.

3.2 Motivation

The common architecture of cloud storage systems like HDFS [10, 75], GFS [29] and Azure [9] is to split each node's storage into data chunks and replicate each chunk on three different randomly selected servers. The widely held view [8, 10] is that three-way replication protects clusters from two types of failures: independent and correlated node failures. Independent node failures are failures that affect individual computers, while correlated node failures are failures related to the hosting infrastructure of the data center (e.g., network, ventilation, power) [19, 10].

In this section, we challenge this commonly held view. First, we demonstrate that it is superfluous to use a replication factor of three to provide data durability against independent failures, and that two replicas provide sufficient redundancy for this type of failure. Second, building on previous work [14, 10], we show that random three-way replication falls short in protecting against correlated failures. These findings provide motivation for a replication scheme that more efficiently handles independent node failures and provides stronger durability in the face of correlated failures.

3.2.1 Analysis of Independent Node Failures

Consider a storage system with N nodes and a replication factor R. Independent node failures are modeled as a Poisson process with an arrival rate of λ. Typical parameters for storage systems are $N = 1,000$ to $N = 10,000$ and $R = 3$ [75, 10, 28, 8].

$\lambda = \frac{N}{MTTF}$, where $MTTF$ is the mean time to permanent failure of a standard server and its components. We borrow the working assumption used by Yahoo and LinkedIn, where about 1% of the nodes in a typical cluster fail independently each month [10, 75], which equates to an annual MTTF of about 8-10 years. In our model we use an MTTF of 10 years for a single node. We also assume that the number of nodes in the system remains constant and that there is always a ready idle server to replace a failed node.

When a node fails, the cluster re-replicates its data by reading it from one or more servers that store replicas of the node's chunks and writing the data into another set of nodes. The node's recovery time depends on the number of servers that can be

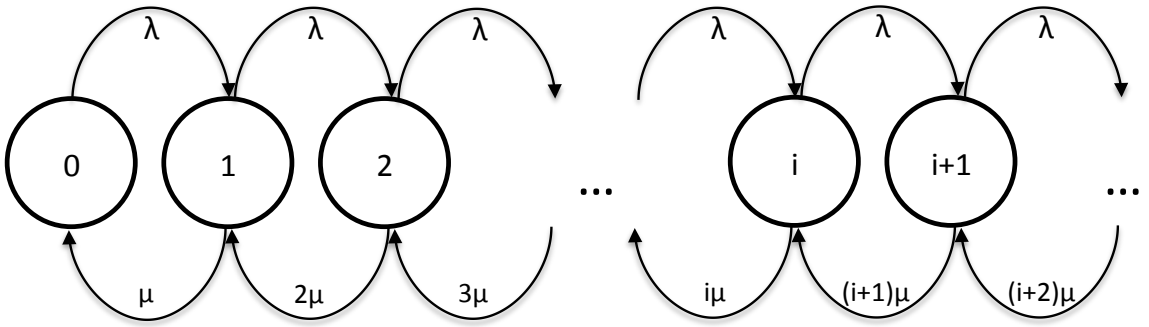

Figure 3.1: Markov chain of data loss due to independent node failures. Each state represents the number of nodes that are down simultaneously.

read from in parallel to recover its data. Using previously defined terminology [14], we term *scatter width* or S as the average number of servers that participate in a single node's recovery. For example, if a node's data has been replicated uniformly on 10 other nodes, when this node fails, the storage system can re-replicate its data by reading it from 10 nodes in parallel. Therefore, its scatter width will be equal to 10.

A single node's recovery time is modeled as an exponential random variable, with a parameter of μ. For simplicity's sake, we assume that recovery time is a linear function of the *scatter width*, or a linear function of the number of nodes that recover in parallel. $\mu = \frac{S}{\tau}$, where τ is the time to recover a full disk over the network. Typical values for τ are between 1-30 minutes [10, 75, 28]. Throughout the paper we use a conservative recovery time for a single node of $\tau = 30\ minutes$. For a scatter width of $S = 10$, which is the value used by Facebook [3], the recovery time will take on average 3 minutes. Note that there is a practical lower bound to recovery time. Most systems first make sure the node has permanently failed before they start recovering the data. For simplicity's sake, we do not consider scatter widths that cause the recovery time to drop below 1 minute.

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

how to compute these probabilities, and in the final subsection we show how to derive the overall rate of failure due to independent node failures.

Probability of i Nodes Failing Simultaneously

We first express $Pr(i\ failed)$ using a Continuous-time Markov chain, depicted in Figure 3.1. Each state in the Markov chain represents the number of failed nodes in a cluster at a given point in time.

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

The probability of each state in a Markov chain with N states can always be derived from a set of N linear equations. However, since N is on the order of magnitude of 1,000 or more, and the number of simultaneous failures due to independent node failures in practical settings is very small compared to the number of nodes, we derived an approximate closed-form solution that assumes an infinite sized cluster. This solution is very simple to compute, and we provide the analysis for it in Appendix 3.6.

The probability of i nodes failing simultaneously at a given point in time is:

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

Where $\rho = \dfrac{\lambda}{\mu}$. We compute the probabilities for different cluster sizes in Table 3.1. The results show that the probability of two or more simultaneous failures due to independent node failures is very low.

Now that we have estimated $Pr(i\ failed)$, we need to estimate $Pr(loss|i\ failed)$.

Number of Nodes	Pr(2 Failures)	Pr(3 Failures)	Pr(4 Failures)
1,000	1.8096×10^{-8}	1.1476×10^{-12}	5.4586×10^{-17}
5,000	4.5205×10^{-7}	1.4334×10^{-10}	3.4091×10^{-14}
10,000	1.8065×10^{-6}	1.1457×10^{-9}	5.4493×10^{-13}
50,000	4.4820×10^{-5}	1.4212×10^{-7}	3.3800×10^{-10}
100,000	1.7758×10^{-4}	1.1262×10^{-6}	5.3568×10^{-9}

Table 3.1: Probability of simultaneous node failures due to independent node failures under different cluster sizes. The model uses $S = 10$, $R = 3$, an average node MTTF of 10 years and a node recovery time of 3 minutes.

Probability of Data Loss Given i Nodes Failed Simultaneously

Previous work has shown how to compute this probability for different types of replication techniques using simple combinatorics [14]. Replication algorithms map each chunk to a set of R nodes. A *copyset* is a set that stores all of the copies of a chunk. For example, if a chunk is replicated on nodes $\{7, 12, 15\}$, then these nodes form a copyset.

Random replication selects copysets randomly from the entire cluster. Facebook has implemented its own random replication technique, where the R nodes are selected from a pre-designated window of nodes. For example, if the first replica is placed on node 10, the remaining two replicas will randomly be placed on two nodes out of a window of 10 subsequent nodes (i.e., they will be randomly selected from nodes $\{11, ..., 20\}$) [14, 3].

Unlike these random schemes, Copyset Replication minimizes the number of copysets each node is a member of [14]. To understand the difference between Copyset Replication and Facebook's scheme, consider the following example.

Assume our storage system has the following parameters: $R = 3$, $N = 9$ and $S = 4$. If we use Facebook's scheme, each chunk will be replicated on another node chosen randomly from a group of S nodes following the first node. E.g., if the primary replica is placed on node 1, the secondary replica will be randomly placed either on node 2, 3, 4 or 5.

Therefore, if our system has a large number of chunks, it will create 54 distinct

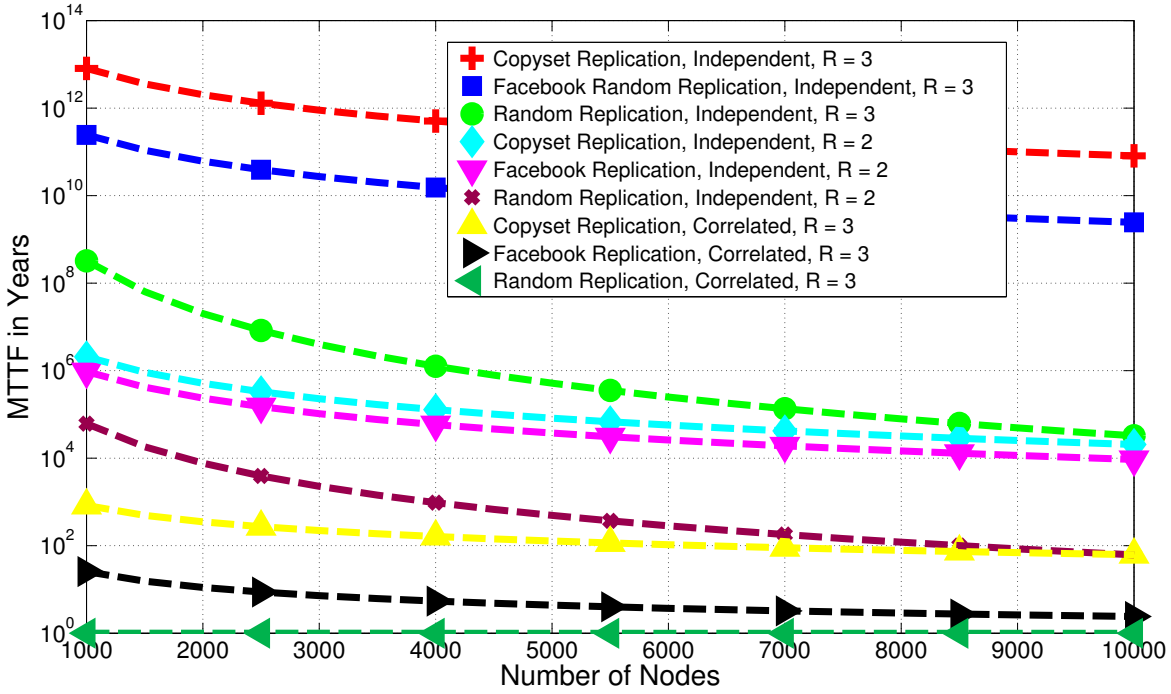

Figure 3.2: MTTF due to independent and correlated node failures of a cluster with a scatter width of 10.

copysets.

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

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

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

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

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

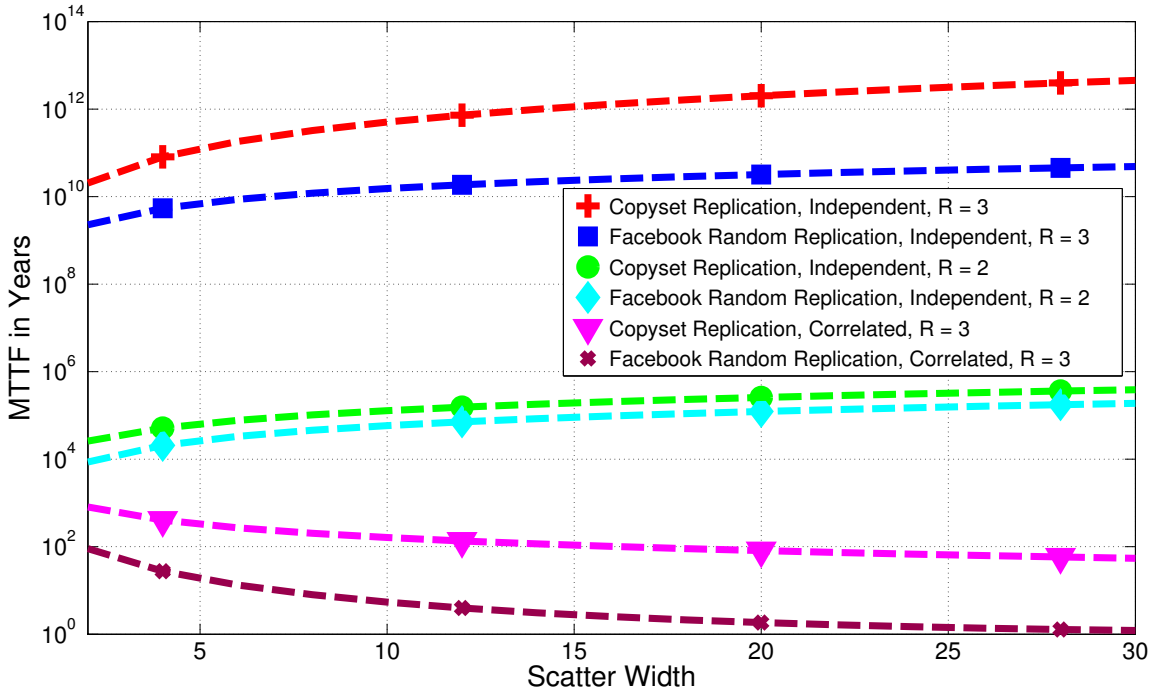

Figure 3.3: MTTF due to independent and correlated node failures of a cluster with 4000 nodes.

only 6 copysets. Now, if three nodes fail, the probability of data loss is:

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

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

Note however, that we decrease the number of copysets, the frequency of data loss under correlated failures will decrease, but each correlated failure event will incur a higher number of lost chunks. This is a desirable trade-off for many storage system designers, where each data loss event incurs a fixed cost [14].

Another design choice that affects the number of copysets is the scatter width. As we increase the scatter width, or the number of nodes from which a node's data can be recovered after its failure, the minimal number of copysets that must be used

increases.

MTTF Due to Independent Node Failures

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

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

The equation accounts for all events in which the Markov chain switches from state $i-1$, in which no loss has occurred, to state i, in which data loss occurs. λ is the transition rate between state $i-1$ and i, $Pr(i-1\ failed)$ is the probability of being in state $i-1$, $(1 - Pr(loss|i\ failed))$ is the probability that there was no data loss when $i-1$ nodes failed and finally $Pr(loss|i\ failed)$ is the probability of data loss when i nodes failed.

Note that no data loss can occur when $i < R$. Therefore, the sum can be computed from $i = R$.

In addition, Table 3.1 shows that under practical system parameters, the probability of i simultaneous node failures due to independent node failures drops dramatically as i increases. Therefore:

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

Using this equation, Figure 3.2 depicts the MTTF of data loss under independent failures for $R = 2$ and $R = 3$ with three replication schemes, Random Replication, Facebook's Random Replication and Copyset Replication, as a function of the cluster's size.

It is evident from the figure that Facebook's Random Replication and Copyset Replication have much higher MTTF values than Random Replication. The reason is that they use a much smaller number of copysets than Random Replication, and therefore their $Pr(loss|i\ failed)$ is smaller.

3.2.2 Analysis of Correlated Node Failures

Correlated failures occur when an infrastructure malfunction causes multiple nodes to be unavailable for a long period of time. Such failures include power outages that may affect an entire cluster, network switch malfunctions and rack power failures [10, 19]. Storage system designers can largely avoid data loss related to some of the common correlated failure scenarios, by placing replicas on different racks or network segments [28, 8, 10]. However, these techniques only go so far to mitigate data loss, and storage systems still face unexpected simultaneous failures of nodes that share replicas. Such data loss events have been documented by multiple data center operators, such as Yahoo [75], LinkedIn [10] and Facebook [8, 3].

In order to analyze the affect of correlated failures on MTTF, we use the observation made by LinkedIn and Yahoo, where on average, once a year, 1% of the nodes do not recover after a cluster-wide power outage. This has been documented as the most severe cause of data loss due to correlated failures [10, 75]. We compute the probability of data loss for this event using the same technique used by previous literature [14].

Figure 3.2 also presents the MTTF of data loss under correlated failures. It is evident from the graph that the MTTF due to correlated failures for $R = 3$ is three orders of magnitude lower than independent failures with $R = 2$ and six orders of magnitude lower than independent failures with $R = 3$, for any replication scheme.

Therefore, our conclusion is that $R = 2$ is sufficient to protect against independent node failures, and that system designers should only focus on further increasing the MTTF under correlated failures, which is by far the main contributing factor to data loss. This has been corroborated in studies conducted by Google [28] and LinkedIn [10].

This also provides further evidence that random replication is much more susceptible to data loss under correlated and independent failures than other replication schemes. Therefore, in the rest of the paper we compare Tiered Replication only against Facebook's Random Replication and Copyset Replication.

Figure 3.3 plots the MTTF for correlated and independent node failures using the same model as before, as a function of the scatter width. This graph demonstrates

that Copyset Replication provides a much higher MTTF than Facebook's Random Replication scheme.

The figure also shows that increasing the scatter width has an opposite effect on MTTF for independent and correlated node failures. The MTTF due to independent node failures increases as a function of the scatter width, since a higher scatter width provides faster node recovery times. In contrast, the MTTF due to correlated node failures decreases as a function of the scatter width, since higher scatter width produces more copysets.

However, since the MTTF of the system is determined primarily by correlated failures, we can also conclude that if system designers wish to reduce the probability of overall data loss events, they should use a small scatter width.

3.2.3 The Peculiar Case of the Nth Replica

This analysis prompted us to investigate whether we can further increase the MTTF under correlated failures. We assume that the third replica was introduced in most cases to provide increased durability and not for increased read throughput [10, 76, 12]. This is true especially in storage systems that utilize chain replication, where the reads will only occur from one end of the chain (from the head or from the tail) [23, 84, 82].

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

In the independent failure model depicted by Figures 3.2 and 3.3, the third replica is required in very rare circumstances for Facebook Random Replication and Copyset Replication, on the order of magnitude of every 10^5 years. However, this third replica is essential for protecting against correlated failures.

In order to leverage this property, we can split our storage system into two tiers.

The *primary tier* would contain the first and second replicas of each chunk, while the *backup tier* would contain the third replica of each chunk. If possible, failures in the primary tier will always be recovered using nodes from the primary tier. We only recover from the backup tier if both the first and second replicas fail simultaneously. In case the storage system requires more than two nodes for read availability, the primary tier will contain the number of replicas required for availability, while the backup tier will contain an additional replica.

Therefore, the backup tier will be mainly read during large scale correlated failures, which are fairly infrequent (e.g., on the order of once or twice a year), as reported by various data center operators [10, 75, 8]. Consequently, the backup tier can be viewed as write dominated storage, since it is written to every time a chunk is changed (e.g., thousands of times a second), but only read from a few times a year.

Splitting the cluster into tiers provides multiple advantages. The storage system designer can significantly reduce the correlation between failures in the primary tier and the backup tier. This can be achieved by storing the backup tier in a geographically remote location, or by other means of physical separation such as using different network and power infrastructure. It has been shown by Google that storing data in a physical remote location significantly reduces the correlation between failures across the two sites [28].

Another possible advantage is that the backup tier can be stored more cost-effectively than the primary tier. For example, the backup tier can be stored on a cheaper storage medium (e.g., tape, or disk in the case of an SSD based cluster), its data may be compressed [69, 38, 50, 64, 44], deduplicated [63, 92, 22] or may be configured in other ways to be optimized for a write dominated workload.

The idea of using geo-replication to reduce the correlation between replicas has been explored extensively using full cluster geo-replication. However, existing geo-replication techniques replicate all replicas from the main cluster to a second cluster, which more than doubles the cost of storage [28, 52].

In this paper, we propose a replication technique, Tiered Replication, that supports tiered clusters and does not duplicate the entire cluster. Previous random replication techniques are inadequate, since as we presented in Figure 3.2 they are highly

susceptible to correlated node failures. Previous non-random techniques like Copyset Replication do not readily support data topology constraints such as tiered replicas and fall short in supporting dynamic data center settings when nodes frequently join and leave the cluster [14].

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

Table 3.2: Tiered Replication algorithm's variables and helper functions.

3.3 Design

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

Algorithm 1 describes Tiered Replication, while Table 3.2 contains the definitions used in the algorithm. Tiered Replication continuously creates new copysets until all nodes are replicated with sufficient scatter width. Each copyset is formed by iteratively picking candidate nodes with a minimal scatter width that meet the constraints of the nodes that are already in the copyset. Algorithm 2 describes the part of the

Algorithm 1 Tiered Replication

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

Algorithm 2 Check Constraints Function

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

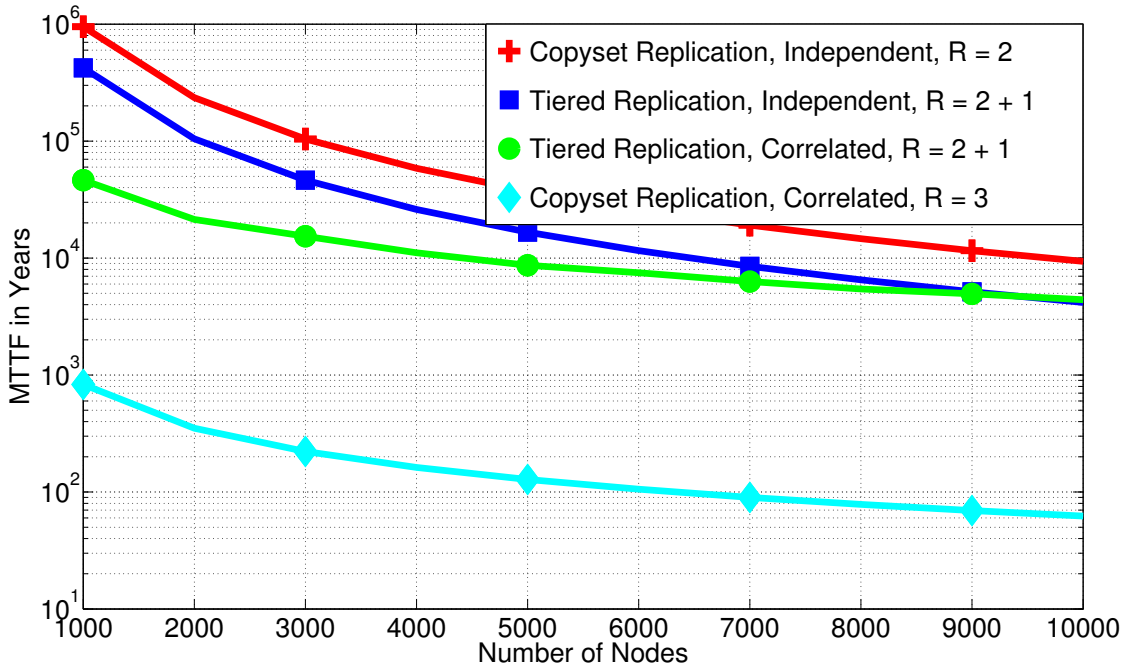

Figure 3.4: MTTF due to independent and correlated node failures of a cluster with a scatter width of 10.

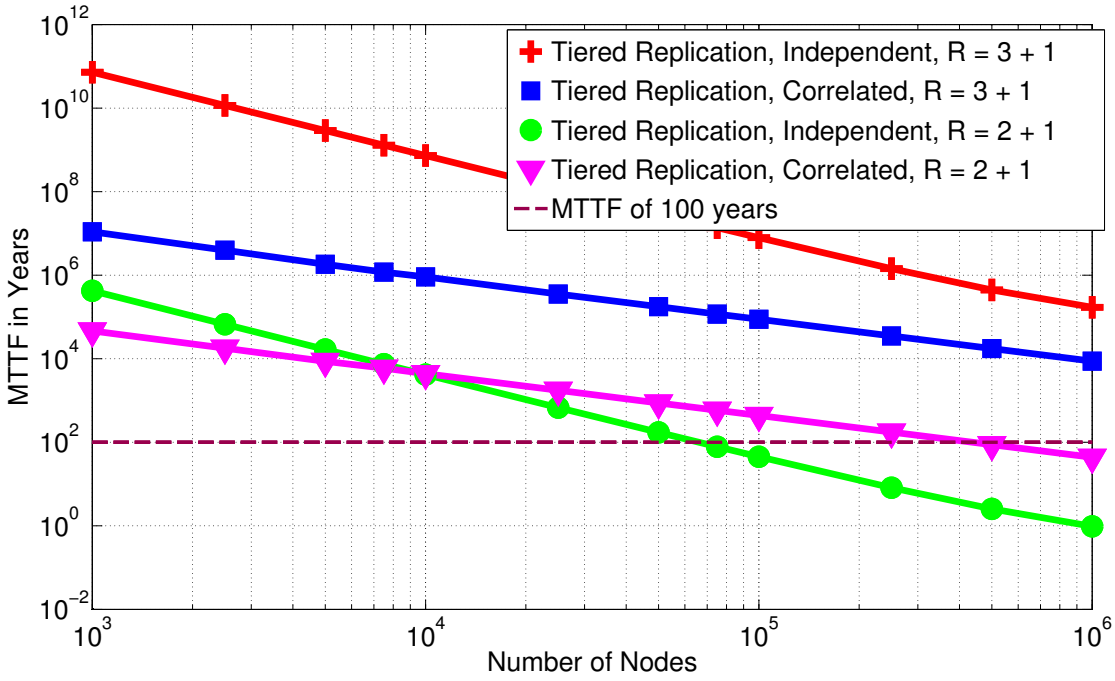

Figure 3.5: MTTF due to independent and correlated node failures of a cluster with a scatter width of 10.

algorithm that checks whether the copyset has met the constraints. The first constraint satisfies the tier requirements, i.e., having exactly one node in each copyset that belongs to the backup tier. The second constraint enforces the minimization of the number of copysets, by requiring that the nodes in the new copyset do not appear with each other in previous copysets. This constraint minimizes the number of copysets, because each new copyset contributes the maximum increase of scatter width. For more information see [14].

Note that there may be cases when the algorithm does not succeed to find a copyset that satisfies all the constraints. In this case, the algorithm is run again, with a relaxed set of constraints (e.g., we can relax the constraint of minimizing the number of copysets, and allow more overlap between copysets).

3.3.1 Analysis of Tiered Replication

We evaluate the durability of Tiered Replication under independent and correlated node failures. To measure the MTTF under independent node failures, we use the same Continuous-time Markov model that we presented in Section 3.2. The results are presented in Figures 3.4 and 3.5. Note that $R = 2 + 1$ means we use Tiered Replication with two replicas in the primary tier and one replica in the backup tier.

Note that in Tiered Replication when a replica fails in the primary tier, if possible, it is only recovered from other nodes in the primary tier. Therefore, fewer nodes will participate in recovery, because the backup tier nodes will not be recovered from. In order to compensate for this effect, system designers that use Tiered Replication may choose to increase the scatter width. For our analysis we compute the MTTF using the same scatter width for Tiered Replication and other replication schemes. Figure 3.4 shows that for $S = 10$, the MTTF under independent node failures is higher for Copyset Replication compared to Tiered Replication, because fewer nodes participate in the recovery of primary replicas and its single-node recovery time is therefore higher.

Also, note that in Figures 3.4 and 3.5, we assume that for $R = 2 + 1$, the third replica is never used to recover from independent node failures. In reality, the backup

tier is used for any failure of two nodes from the primary tier, and therefore will be used in the rare case of an independent node failure that simultaneously affects two nodes in the primary tier that are in the same copyset. Hence, the MTTF under independent node failures for Tiered Replication is even higher than depicted by the graphs.

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

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

To compute the MTTF of data loss due to this scenario, we need to compute the probability that a node failure will occur in the backup cluster while a correlated failure event is occurring in the primary cluster. To be on the conservative side, we assume that it takes 12 hours to fully recover the data after the correlated failure in the primary tier (LinkedIn data center operators report that unavailability events typically take 1-3 hours to recover from [10]). We compute the probability of data loss in this scenario, using the same combinatorial methods that we used to compute the MTTF under correlated failures before.

Figure 3.4 shows that the MTTF of Tiered Replication is more than two orders of magnitude greater than Copyset Replication. This is due to the fact that it is much less likely to lose data under correlated failures when one of the replicas is stored on an independent cluster. Recall that Copyset Replication's MTTF was already three orders of magnitude greater than random replication.

In Figure 3.5 we explore the following question: what is the turning point, where a storage system needs to use $R = 4$ instead of $R = 3$? We plot the MTTF of Tiered Replication and extend it $N = 1,000,000$, which is a much larger number of nodes than is used in today's clusters. Assuming that storage designers are targeting an MTTF of at least 100 years, our results show that at around 100,000 nodes storage systems should switch to a default of $R = 4$. Note that Figure 3.4 shows that Copyset Replication needs to switch to $R = 4$ much sooner, at about 5,000 nodes. Other replication schemes, like Random Replication or Facebook's scheme, fail to achieve an MTTF of 100 years with $R = 3$, even for very small clusters.

3.3.2 Dynamic Cluster Changes

Since running Tiered Replication is fast to execute (on the order of milliseconds, see Section 3.4) and the algorithm is structured to create new copysets incrementally, the storage system can run it every time the cluster changes its configuration.

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

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

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

One of the main disadvantages of Copyset Replication is that it is a static scheme

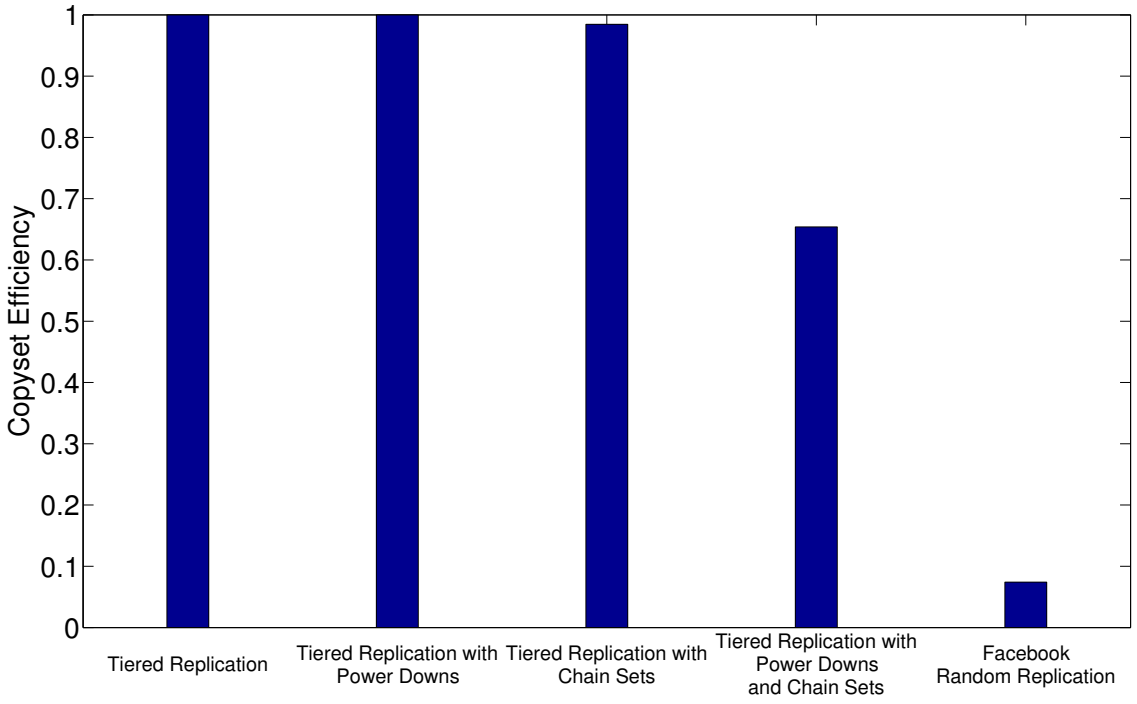

Figure 3.6: Adding constraints on Tiered Replication increases the number of copysets, on a cluster with $N = 4000$, $R - 3$ and $S = 10$.

where the copysets are generated by randomly permuting all the nodes in the cluster, and then dividing them to copysets. Addition and deletion of nodes are handled locally in a non-optimal way, and they create excessive copysets. This means that at some point the system designers would need to completely rearrange the data placement in the entire cluster. Unlike Copyset Replication, Tiered Replication is inherently an online algorithm, which incrementally adds copysets as needed, and is always optimized for reducing the overall number of copysets.

3.3.3 Additional Constraints

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

Controlled Power Down: Some storage designers would like to allow parts of the

cluster to be temporarily switched off to reduce power consumption (e.g., according to diurnal patterns). For example, Sierra [83], allows a cluster with three replicas to power down two thirds of its cluster and still allow access to data. This feature can easily be added as a constraint to Tiered Replication by forcing each copyset to contain a node that belongs to a different partition. This feature is also important for supporting controlled software or hardware upgrades, where parts of the cluster may be powered down without affecting the cluster availability.

Chain Replication: In chain replication, each replica it is assigned a position in the chain (e.g., head, middle, tail) [84]. Chain replication can provide improved performance and consistency. A desirable property of chain replication is that each node will have an equal number of replicas in each position. It is straightforward to incorporate this requirement into Tiered Replication. In order to ensure that nodes have an even distribution of chain positions for their replicas, when the algorithm assigns nodes to copysets and chain positions, it tries to balance the number of times the node will appear in each chain position. For example: if a node has been assigned to the head position twice, middle position twice and tail position once, the algorithm will enforce that it will be assigned to a tail position in the next copyset the node will belong to.

To demonstrate the ability to incorporate additional constraints to Tiered Replication, we implemented it on HyperDex [23], a storage system that uses chain replication. Note that Copyset Replication and Random Replication are inefficient for supporting balanced chain sets. Copyset Replication is not designed for incorporating such constraints because it randomly permutes the entire set of nodes. Random Replication is not effective for this requirement because its random placement of nodes frequently creates imbalanced chain positions.

3.3.4 Analysis of Additional Constraints

Figure 3.6 demonstrates the effect of adding constraints on the number of copysets. In the figure, *copyset efficiency* is equal to the ratio between the number of copysets generated by an optimal replication scheme that minimizes the number of copysets,

and the number of copysets generated by the replication scheme with the constraint.

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

3.4 Implementation

We have implemented Tiered Replication on HyperDex, an open-source key-value storage system [23]. In Hyperdex, the coordinator is responsible for replicating chunks across the nodes. It utilizes a replication technique called chain replication [84]. In chain replication, each node has a position in the chain. The first node in the chain is the head, there are middle nodes, and the last node in the chain is the tail. Heads service client read requests and receive write requests. When writes are serviced they are passed sequentially through the nodes in the chain, and if successful the tail acknowledges the write.

Each node serves as a head, tail or middle node for different chunk replicas. In order to spread the workload evenly across the nodes in the cluster, HyperDex requires that each node will have a balanced number of positions in the chain.

Since Tiered Replication only takes several milliseconds to run, and is only called when nodes join or leave the cluster, it is implemented in the main loop of the configuration manager of HyperDex. We added the chain replication requirement as an additional constraint to the Tiered Replication algorithm. In addition to tracking which copysets each node has appeared in, we also track the chain positions that each node has already appeared in. When a node is assigned to a new copyset, among the nodes that satisfy the scatter width requirement (i.e., that haven't appeared with the other nodes in a copyset before), we select the node that has a minimal number of appearances in a chain position that is still open in the copyset. For simplicity's sake, we chose to replicate to the backup tier synchronously; that is, the storage system does not acknowledge a write until it has been written to the backup tier.

In the next subsections, we evaluate the performance of our Tiered Replication implementation. The goal of this evaluation is to demonstrate the performance impact of using Tiered Replication in a practical setting. Note that our evaluation does not evaluate the frequency of data loss, because the typical time scale (decades) and cluster sizes (thousands of nodes) required for such experiments are impractical.

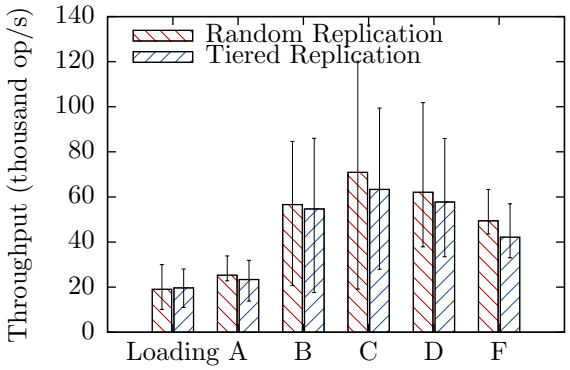

Figure 3.7: Tiered Replication throughput under YCSB benchmark. Each bar represents the throughput under a different YCSB workload.

3.4.1 Performance Benchmarks

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

We compare Tiered Replication to HyperDex's default replication scheme, random replication. We ran 6 nodes in one availability zone (us-east-1a) and the three remaining nodes in a second availability zone (us-east-1b). In Amazon EC2, each availability zone runs on its own physically distinct, independent infrastructure. Common points of failures like generators and cooling equipment are not shared across availability zones. Additionally, they are physically separate, such that even extremely uncommon disasters such as fires, tornados or flooding would only affect a single availability zone [2].

Figure 3.7 presents the results of running Tiered Replication and random replication under two availability zones, using YCSB (Yahoo! Cloud Serving Benchmark) [17]. We used 16 client threads per host on each of the 9 hosts, with one million 1KiB objects. We plotted the throughput we achieved using both replication schemes. We attribute the difference in throughput between Tiered Replication and Random Replication to random performance variabilities in the Amazon virtualized environment. The error bars in the graph depicts the high variation of the throughput

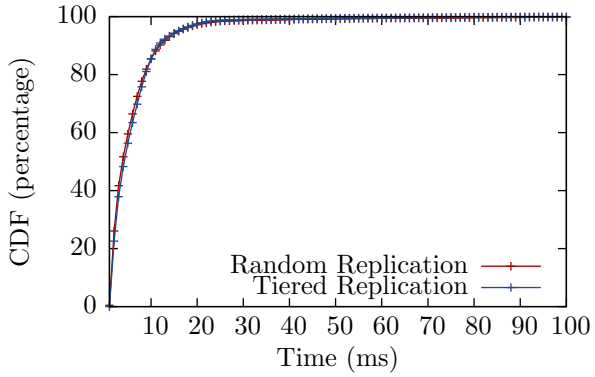

Figure 3.8: Tiered Replication write latency under YCSB benchmark.

among different runs of the experiment.

3.4.2 Write Latency

We evaluate the impact of Tiered Replication on latency. Figure 3.8 compares the write latency of Tiered Replication and random replication under the YCSB benchmark. The figure shows that there is no measurable difference in latency.

Note that we are comparing two replication schemes that split nodes across two clusters. Clearly, we expect the write latency of Tiered Replication if it is compared to the write latency of a storage system that is deployed entirely in a single availability zone.

3.4.3 Recovery Evaluation

We measured the time it takes to recover a node in the primary tier with HyperDex using Tiered Replication while running the YCSB benchmarks. HyperDex took approximately 98 seconds to recover all of the node's data, which is consistent with the performance of HyperDex using random replication.

3.5 Related Work

Several researchers have made observations that the MTTF under independent failures is much higher than from correlated failures [58, 87]. A LinkedIn field study reported no record of data loss due to independent node failures [10]. Google researchers have shown that the MTTF with three replicas under correlated failures is almost three orders of magnitude lower than the MTTF under independent node failures [28].

Google researchers developed an analysis based on a Markov model that computes the MTTF for a single stripe under independent and correlated failures. However, they did not provide an analysis for the MTTF of the entire cluster [28]. Nath et al. modeled the affect of correlated node failures and demonstrated that replication techniques that prevent data loss under independent node failures are not always effective for preventing correlated node failures [58]. In addition, several researchers have modeled the MTTF for individual device components, and in particular for disks [70, 39, 5, 72, 62].

Several replication schemes addressed the high probability of data loss under correlated failures. Facebook's HDFS implementation [8, 3] limits the scatter width of Random Replication, in order to reduce the probability of data loss under correlated failures. Copyset Replication [14] improved Facebook's scheme, by restricting the replication to a minimal number of copysets for a given scatter width. Tiered Replication is the first replication technique that not only minimizes the probability of data loss under correlated failures, but also leverages the much higher MTTF under independent failures to further increase the MTTF under correlated failures. In addition, unlike Copyset Replication, Tiered Replication can gracefully tolerate dynamic cluster changes, such as nodes joining and leaving the cluster and planned cluster power downs. It also supports chain replication and the ability to distribute replicas to different racks and failure domains, which is a desirable requirement of replication schemes [53, 8].

The traditional way to increase durability under correlated failures is to use geo-replication [28, 52, 49, 77, 91]. Geo-replication duplicates the entire cluster to a remote site. Therefore, if the cluster was using three replicas, once it is geo-replicated, the storage provider will effectively use six replicas. Similarly, Glacier [33] and Oceanstore [46, 66] design an archival storage layer that provides extra protection against correlated failures by adding multiple new replicas to the storage system. While the idea of using archival replicas is not new, Tiered Replication is more cost-efficient, since does not require any additional storage for the backup: it migrates one replica from the original cluster to a backup tier. In addition, previous replication techniques utilize random placement schemes and do not minimize the number of copysets, which leaves them susceptible to correlated failures.

Storage coding is a technique for reducing the storage overhead of replication [69, 38, 50, 64, 44]. De-duplication is also commonly used to reduce the overhead of redundant copies of data [63, 92, 22]. Tiered Replication is fully compatible with any coding or de-duplication schemes for further reduction of storage costs. Moreover, Tiered Replication enables storage systems to further reduce costs by storing the third replicas of their data on a cheap storage medium such as tape, or hard disks in the case of an solid-state based storage cluster.

3.6 Appendix

This section contains the closed-form solution for the Markov chain described in Section 3.2 and Figure 3.1 with an infinite number of nodes. The state transitions for state i are:

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

Therefore:

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

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

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

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

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

If we apply the recursive formula:

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

Using the equality $\sum_{i=0}^{\infty} \frac{\rho^i}{i!} = e^{\rho}$, we get: $Pr(0) = e^{-\rho}$.

Therefore, we now have a simple closed-form formula for all of the Markov state probabilities:

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

Chapter 4

Conclusions

4.1 Conclusions

This works challenged current best practices of data durability in cloud storage systems. It demonstrated that existing replication techniques are far from being optimal for guarding against different types of node failure events. This dissertation formed new paradigms for analyzing node failures, and designing non-random data placement schemes that provide significantly higher durability than existing techniques, at the same storage cost and cluster performance.

The first part of the dissertation shows that randomly replicating across an entire cluster incurs almost guaranteed data loss when a small number of nodes fail at the same time, due to a correlated failure. Given that many storage system designers prefer to reduce the occurance of data loss events, since they incur a high fixed cost, the work presents a novel framework that trades off between the frequency of data loss events and the data loss magnitude in each event. It presents a design of a novel replication scheme, Copyset Replication, which provides a near optimal reduction in the probability of data loss events, as a function of the scatter width, or node recovery time. Copyset Replication was implemented on two open-source cloud storage systems, RAMCloud and Hadoop File System, and the work demonstrated that when 1% of the nodes fail simultaneously, Copyset Replication reduces the probability the probability of data loss from 99.99% to 0.15%. In the Facebook Hadoop File System setup, Copyset Replication would reduce the probability of data loss from 22.8% to 0.78%.

In addition to the practical aspects of cloud replication, the work also demonstrates the connection between cloud data placement, to the field of Balanced Incomplete Block Designs (BIBD). Existing BIBD results only address the particular case where the scatter width is equal to the number of nodes in the system. Therefore, the copyset problem introduces a new thoretical framework for developing block designs where the scatter width is smaller than the number of nodes in the cluster.

The second part of this dissertation questioned the common wisdom of cloud storage systems that rely on three-way replication within a cluster to protect against independent node failures, and on full geo-replication of an entire cluster to protect

against correlated failures. It provided an analytical framework for computing the probability of data loss under independent and correlated node failures, and demonstrated that the standard replication architecture used by cloud storage systems is inefficient. Three-way replication is excessive for protecting against independent node failures, and clearly falls short of protecting storage systems from correlated node failures. The key insight of this chapter is that since the third replica is rarely needed for recovery from independent node failures, it can be placed on a geographically separated cluster, without causing a significant impact to the recovery time from independent node failures, which occur frequently in large clusters.

The work presented Tiered Replication, a replication technique that automatically places the n-th replica on a separate cluster, while minimizing the probability of data loss under correlated failures, by minimizing the number of copysets. Tiered Replication improves the cluster-wide MTTF by a factor of 100,000 compared to random replication, without increasing the storage capacity. In addition, unlike geo-replication, Tiered Replication does not duplicate an entire cluster's data. Tiered Replication supports additional data placement constraints required by the storage designer, such as rack awareness and chain replication assignments, and can dynamically adapt when nodes join and leave the cluster. An implementation of Tiered Replication on HyperDex, a key-value storage system, demonstrates that it incurs a small performance overhead.

A potential future extension of this result is designing and implementing Tiered Replication with asynchronous replication. Asynchronous replication would allow Tiered Replication to acknowledge write requests before the last replica is written to the backup tier. This would decrease the write latency of Tiered Replication in cases where writing to the backup tier causes a long delay.

Bibliography

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

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

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

[4] Sérgio Almeida. Geo-replication in large scale cloud computing applications. *Master's thesis, Univ. Técnica de Lisboa*, 2007.

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

[6] Mehmet Bakkaloglu, Jay J Wylie, Chenxi Wang, and Gregory R Ganger. On correlated failures in survivable storage systems. Technical report, DTIC Document, 2002.

[7] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, December 2007.

[8] Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, Rodrigo Schmidt, and Amitanand 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] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas 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] Robert J. Chansler. Data Availability and Durability with the Hadoop Distributed File System. *;login: The USENIX Magazine*, 37(1), February 2012.

[11] Robert J Chansler. Personal Communication, 2013.

[12] Byung-Gon Chun, Frank Dabek, Andreas Haeberlen, Emil Sit, Hakim Weatherspoon, M Frans Kaashoek, John Kubiatowicz, and Robert Morris. Efficient replica maintenance for distributed storage systems. *NSDI*, 6:4–4, 2006.

[13] Asaf Cidon, Robert Escriva, Sachin Katti, Mendel Rosenblum, and Emin Gun Sirer. Tiered replication: A cost-effective alternative to full cluster geo-replication. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 31–43, Santa Clara, CA, July 2015. USENIX Association.

[14] Asaf Cidon, Stephen M. Rumble, Ryan Stutsman, Sachin Katti, John Ousterhout, and Mendel Rosenblum. Copysets: Reducing the frequency of data loss in cloud storage. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*, USENIX ATC'13, pages 37–48, Berkeley, CA, USA, 2013. USENIX Association.

[15] W.G. Cochran and G.M. Cox. Experimental designs . 1957.

[16] WG Cochran and DJ Watson. An experiment on observer's bias in the selection of shoot-heights. *Empire Journal of Experimental Agriculture*, 4(13):69–76, 1936.

[17] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[18] Frank Dabek, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with cfs. *ACM SIGOPS Operating Systems Review*, 35(5):202–215, 2001.

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

[20] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007.

[21] Norman R Draper and Friedrich Pukelsheim. An overview of design of experiments. *Statistical Papers*, 37(1):1–32, 1996.

[22] Cezary Dubnicki, Leszek Gryz, Lukasz Heldt, Michal Kaczmarczyk, Wojciech Kilian, Przemyslaw Strzelczak, Jerzy Szczepkowski, Cristian Ungureanu, and Michal Welnicki. Hydrastor: A scalable secondary storage. In *FAST*, volume 9, pages 197–210, 2009.

[23] Robert Escriva, Bernard Wong, and Emin Gün Sirer. Hyperdex: A distributed, searchable key-value store. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 25–36, New York, NY, USA, 2012. ACM.

[24] Sérgio Esteves, Joao Silva, and Luís Veiga. Quality-of-service for consistency of data geo-replication in cloud computing. In *Euro-Par 2012 Parallel Processing*, pages 285–297. Springer, 2012.

[25] Bin Fan, Wittawat Tantisiriroj, Lin Xiao, and Garth Gibson. DiskReduce: Replication as a prelude to erasure coding in data-intensive scalable computing, 2011.

[26] Stephen E Fienberg, Burton Singer, and Judith M Tanur. Large-scale social experimentation in the united states. In *A Celebration of Statistics*, pages 287–326. Springer, 1985.

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

[28] Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean 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.

[29] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *SOSP*, pages 29–43, 2003.

[30] Garth A Gibson. *Redundant disk arrays: Reliable, parallel secondary storage*, volume 368. MIT press Cambridge, MA, 1992.

[31] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 350–361. ACM, 2011.

[32] Jim Gray and Catharine Van Ingen. Empirical measurements of disk failure rates and error rates. *arXiv preprint cs/0701166*, 2007.

[33] Andreas Haeberlen, Alan Mislove, and Peter Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *IN PROC. OF NSDI*, 2005.

[34] Danny Harnik, Dalit Naor, and Itai Segall. Low power mode in cloud storage systems.

[35] Mark Holland and Garth A Gibson. *Parity declustering for continuous operation in redundant disk arrays*, volume 27. ACM, 1992.

[36] Robert Horst and Pankaj Mehra. Method and apparatus for cluster interconnection using multi-port nodes and multiple routing fabrics, 2002.

[37] SK Houghten, LH Thiel, J. Janssen, and CWH Lam. There is no (46, 6, 1) block design*. *Journal of Combinatorial Designs*, 9(1):60–71, 2001.

[38] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. Erasure coding in windows azure storage. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[39] Weihang Jiang, Chongfeng Hu, Yuanyuan Zhou, and Arkady Kanevsky. Are disks the dominant contributor for storage failures?: A comprehensive study of storage subsystem failure characteristics. *Trans. Storage*, 4(3):7:1–7:25, November 2008.

[40] M Kalyanakrishnam, Zbigniew Kalbarczyk, and Ravishanka Iyer. Failure data analysis of a lan of windows nt based computers. In *Reliable Distributed Systems, 1999. Proceedings of the 18th IEEE Symposium on*, pages 178–187. IEEE, 1999.

[41] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

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

[43] J.C. Koo and JT 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.

[44] Ramakrishna Kotla, Lorenzo Alvisi, and Mike Dahlin. Safestore: a durable and practical storage system. In *USENIX Annual Technical Conference*, pages 129–142, 2007.

[45] Tim Kraska, Gene Pang, Michael J Franklin, Samuel Madden, and Alan Fekete. Mdcc: Multi-data center consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 113–126. ACM, 2013.

[46] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, et al. Oceanstore: An architecture for global-scale persistent storage. *ACM Sigplan Notices*, 35(11):190–201, 2000.

[47] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[48] Jacob Leverich and Christos Kozyrakis. On the energy (in)efficiency of hadoop clusters. *SIGOPS Oper. Syst. Rev.*, 44(1):61–65, March 2010.

[49] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 265–278, Berkeley, CA, USA, 2012. USENIX Association.

[50] Runhui Li, Patrick PC Lee, and Yuchong Hu. Degraded-first scheduling for mapreduce in erasure-coded storage clusters.

[51] Wyatt Lloyd, Michael J Freedman, Michael Kaminsky, and David G Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 401–416. ACM, 2011.

[52] Wyatt Lloyd, Michael J Freedman, Michael Kaminsky, and David G Andersen. Stronger semantics for low-latency geo-replicated storage. In *Symposium on Networked Systems Design and Implementation*, 2013.

[53] John MacCormick, Nicholas Murphy, Venugopalan Ramasubramanian, Udi Wieder, Junfeng Yang, and Lidong Zhou. Kinesis: A new approach to replica placement in distributed storage systems. *ACM Transactions On Storage (TOS)*, 4(4):11, 2009.

[54] Kannan Mathukkaruppan. Personal Communication, 2012.

[55] Pankaj Mehra. Network and method of configuring a network, 2003.

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

[57] Richard R Muntz and John CS Lui. *Performance analysis of disk arrays under failure*. Computer Science Department, University of California, 1990.

[58] Suman Nath, Haifeng Yu, Phillip B Gibbons, and Srinivasan Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. In *NSDI*, volume 6, pages 225–238, 2006.

[59] Daniel Nurmi, John Brevik, and Rich Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Euro-Par 2005 Parallel Processing*, pages 432–441. Springer, 2005.

[60] Diego Ongaro, Stephen M. Rumble, Ryan Stutsman, John K. Ousterhout, and Mendel Rosenblum. Fast crash recovery in RAMCloud. In *SOSP*, pages 29–41, 2011.

[61] David Oppenheimer, Archana Ganapathi, and David A Patterson. Why do internet services fail, and what can be done about it? In *USENIX Symposium on Internet Technologies and Systems*, volume 67. Seattle, WA, 2003.

[62] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz Andr Barroso. Failure trends in a large disk drive population. In *5th USENIX Conference on File and Storage Technologies (FAST 2007)*, pages 17–29, 2007.

[63] Sean Quinlan and Sean Dorward. Awarded best paper! - venti: A new approach to archival data storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[64] KV Rashmi, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster. In *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*. USENIX, 2013.

[65] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. *A scalable content-addressable network*, volume 31. ACM, 2001.

[66] Sean C Rhea, Patrick R Eaton, Dennis Geels, Hakim Weatherspoon, Ben Y Zhao, and John Kubiatowicz. Pond: The oceanstore prototype. In *FAST*, volume 3, pages 1–14, 2003.

[67] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 188–201. ACM, 2001.

[68] Ramendra K Sahoo, Mark S Squillante, Anand Sivasubramaniam, and Yanyong Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *Dependable Systems and Networks, 2004 International Conference on*, pages 772–781. IEEE, 2004.

[69] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. Xoring elephants: novel erasure codes for big data. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 325–336. VLDB Endowment, 2013.

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

[71] Bianca Schroeder and Garth A Gibson. A large-scale study of failures in high-performance computing systems. *Dependable and Secure Computing, IEEE Transactions on*, 7(4):337–350, 2010.

[72] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. Dram errors in the wild: A large-scale field study. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 193–204, New York, NY, USA, 2009. ACM.

[73] Eric J Schwabe and Ian M Sutherland. Improved parity-declustered layouts for disk arrays. In *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, pages 76–84. ACM, 1994.

[74] Thomas Schwarz, Mary Baker, Steven Bassi, Bruce Baumgart, Wayne Flagg, Catherine van Ingen, Kobus Joste, Mark Manasse, and Mehul Shah. Disk failure investigations at the internet archive. In *Work-in-Progess session, NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST2006)*, 2006.

[75] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. *Mass Storage Systems and Technologies, IEEE / NASA Goddard Conference on*, 0:1–10, 2010.

[76] Emil Sit, Andreas Haeberlen, Frank Dabek, Byung-Gon Chun, Hakim Weatherspoon, Robert Morris, M Frans Kaashoek, and John Kubiatowicz. Proactive replication for data durability. In *IPTPS*, 2006.

[77] Yair Sovran, Russell Power, Marcos K Aguilera, and Jinyang Li. Transactional storage for geo-replicated systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 385–400. ACM, 2011.

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

[79] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[80] Nisha Talagala and David Patterson. An analysis of error behavior in a large storage system. Technical Report UCB/CSD-99-1042, EECS Department, University of California, Berkeley, Feb 1999.

[81] Dong Tang, Ravishankar K Iyer, and Sujatha S Subramani. Failure analysis and modeling of a vaxcluster system. In *Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, pages 244–251. IEEE, 1990.

[82] Jeff Terrace and Michael J Freedman. Object storage on craq: High-throughput chain replication for read-mostly workloads. In *Proc. USENIX Annual Technical Conference*, page 59, 2009.

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

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

[85] Martin B Wilk and Oscar Kempthorne. Some aspects of the analysis of factorial experiments in a completely randomized design. *The Annals of Mathematical Statistics*, pages 950–985, 1956.

[86] Jun Xu, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Networked windows nt system field failure data analysis. In *Dependable Computing, 1999. Proceedings. 1999 Pacific Rim International Symposium on*, pages 178–185. IEEE, 1999.

[87] Praveen Yalagandula, Suman Nath, Haifeng Yu, Phillip B Gibbons, and Srinivasan Seshan. Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems. In *USENIX WORLDS*, 2004.

[88] Jimmy Yang and Feng-Bin Sun. A comprehensive review of hard-disk drive reliability. In *Reliability and Maintainability Symposium, 1999. Proceedings. Annual*, pages 403–409. IEEE, 1999.

[89] F Yates and I Zacopanay. The estimation of the efficiency of sampling, with special reference to sampling for yield in cereal experiments. *The Journal of Agricultural Science*, 25(04):545–577, 1935.

[90] Haifeng Yu, Phillip B. Gibbons, and Suman 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.

[91] Yang Zhang, Russell Power, Siyuan Zhou, Yair Sovran, Marcos K Aguilera, and Jinyang Li. Transaction chains: achieving serializability with low latency in geo-distributed storage systems. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 276–291. ACM, 2013.

[92] Benjamin Zhu, Kai Li, and Hugo Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 18:1–18:14, Berkeley, CA, USA, 2008. USENIX Association.