Understanding and Coping with Failures in Large-Scale Storage Systems

Technical Report UCSC-SSRC-07-06
May 2007

Qin Xin
qxin@cs.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
http://www.ssrc.ucsc.edu/

NOTE: This Ph. D. thesis was originally filed in December, 2005.

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

UNDERSTANDING AND COPING WITH FAILURES IN LARGE-SCALE STORAGE SYSTEMS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Qin Xin

December 2005

The Dissertation of Qin Xin
is approved:

Professor Ethan L. Miller, Chair

Professor Darrell D. E. Long

Professor Scott A. Brandt

Professor Thomas J .E. Schwarz, S. J.

Professor Brett D. Fleisch

Lisa C. Sloan
Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

Abstract

Understanding and Coping with Failures in Large-Scale Storage Systems

by

Qin Xin

Reliability for very large-scale storage systems has become more and more important as the need for storage has grown dramatically. New phenomena related to system reliability appear as systems scale up. In such a system, failures are a normality. In order to ensure high reliability for petabyte-scale storage systems in scientific applications, characterization of failures and techniques of coping with them are studied in this thesis.

The thesis first describes the architecture of a petabyte-scale storage system and characterizes the challenges of achieving high reliability in such a system. The long disk recovery time and the large number of system components are identified as the main obstacles against high system reliability.

The thesis then presents a fast recovery mechanism, FARM, which greatly reduces data loss in the occurrence of multiple disk failures. Reliability of a petabyte-scale system with and without FARM has been evaluated. Accordingly, various aspects of system reliability, such as failure detection latency, bandwidth utilization for recovery, disk space utilization, and system scale, have been examined by simulations.

The overall system reliability is modeled and estimated by quantitative analysis based on Markov models and event-driven simulations. It is found that disk failure models which take infant mortality into consideration result in more precise reliability estimation than

the traditional model which assumes a constant failure rate, since infant mortality has a pronounced impact on petabyte-scale systems. To safeguard data against failures from young disk drives, an adaptive data redundancy scheme is presented and evaluated.

A petabyte-scale storage system is typically built up by thousands of components in a complicated interconnect structure. The impact of various failures on the interconnection networks is gauged and the performance and robustness under degraded modes are evaluated in a simulated petabyte-scale storage system with different configurations of network topology.

This thesis is directed towards understanding and coping with failures in petabyte-scale storage systems. It addresses several emerging reliability challenges posed by the increasing scale of storage systems and study the methods to improving system reliability. The research is targeted to help system architects in the designs of reliable storage systems at petabyte-scale and beyond.

Acknowledgements

During my graduate education many people have given me support, guidance, and friendship. Foremost among them is my advisor Professor Ethan Miller. This work would not have been possible without him. His direction and support have been invaluable. His encouragement throughout these years has helped me tremendously.

I would like to thank Professor Thomas Schwarz for being involved in this research all the time. His inspirations, suggestions and discussions have had a lot of influence on this work.

I would also thank Professor Darrell Long and Professor Scott Brandt for greatly supporting me in my graduate study and Professor Brett Fleisch who has given me valuable suggestions for this work.

Thanks also to Dr. Spencer Ng, Dr. Andy Hospodor, Professor Witold Litwin for sharing their insights on file and storage systems with me, and to Professor David Helmbold and Professor Herbie Lee for the inputs on statistics and machine learning.

I always feel lucky to have great colleagues working along with in this project – Feng Wang, Sage Weil, Prof. Carlos Maltzahn, Joel Wu, R.J. Honicky, Christopher Olsen, Kristal Pollack and Lan Xue.

I also thank other present and former members of Storage Systems Research Center including Amer Ahemd, Sasha Ames, Ismail Ari, Scott Banachowski, Deepavali Bhagwat, Timothy Bisson, Nikhil Bobb, Karl Brandt, Nate Edel, Bo Hong, Caixue Lin, Ying Li, Alisa Neeman, Rosie Wacha, Tsozen (Frank) Yeh, and Lawrence You for many discussions and

help over these years. Special thanks to Sage Weil and Deepavali Bhagwat for proofreading my thesis. I appreciate SSRC for providing an excellent study and research environment for me. The staff in SSRC and computer science department, Ma Xiong, Haifang W. Telc, and Carol Mullane, have given me a lot of help.

My friends made these four years special. Particular thanks to them: Guozheng Ge, Jun Hu, Xiaojin Shi, Yu Wang, Rui Li, Sung Kim, Yeon Gwack, Kai Pan, Ming Shen, Anru Wang, Wei Wei, Rui Li, Ping Chen, Mozhen Liu, Shu Zhang and Esteban Valles.

My family is the greatest wealth in my life. They are the ones who encourage me, share my feelings and make me brave. My grandma and parents provide a home full of love, support and discipline, and I owe everything to my husband, Yong Gao, whose love and care have carried me through these years. They are my true heroes. Without them, I have no way to be who I am today. This thesis is dedicated to them.

This work has been funded by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratory under contract B520714. I also thank other sponsors of Storage Systems Research Center, including Engenio, Hewlett-Packard Laboratories, Hitachi Global Storage Technologies, IBM Research, Intel, Microsoft Research, Network Appliance, and Veritas, for their interest, guidance, and support.

To Grandma, Mom, Dad, and Yong,

for your gifts of encouragement and love.

Chapter 1

Introduction

> "The world has arrived at an age of cheap complex devices of great reliability; and something is bound to come of it."
>
> —- Vannevar Bush (1890-1974)

We are living in an era of digital information explosion. In 2002, five exabytes (5×2^{60} bytes) of new information were generated, and new data is growing annually at the rate of 30% [74]. System architects are building ever-larger data storage systems to satisfy the ever-increasing demands of bandwidth and capacity of data-driven applications. In light of rapid advances in storage technologies, highly parallel storage systems are built up with a large number of commodity components, with superior performance and low cost. However, fault tolerance, high availability and reliability[1] have not been emphasized in most designs: people hardly realize the importance of high system reliability until fault or failure occurs. Unfortunately, the consequence of failure is always bitter: for business data servers, unavail-

[1] Availability: The degree to which a system, subsystem, or equipment is operable and in a committable state at the start of a mission, when the mission is called for at an unknown, i.e., a random, time. Reliability: The ability of an item to perform a required function under stated conditions for a specified period of time (from Wikipedia http://en.wikipedia.org/wiki/Wikipedia).

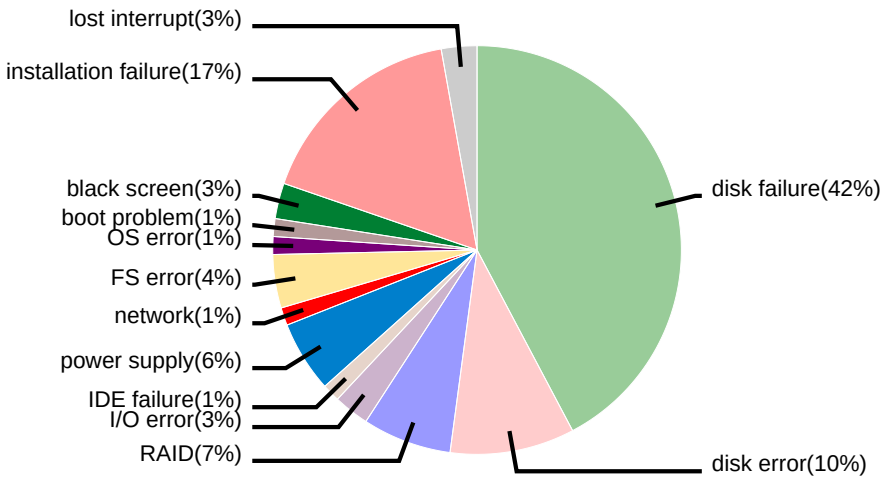


Figure 1.1: Hardware Failures Distribution at Internet Archive during 30 days

ability means loss of millions of dollars; for supercomputing clusters, failure means loss of precious, or even non-reproducible experimental data.

Modern systems are not sufficiently robust to allay concerns about reliability. Recent investigations show that modern business servers could only achieve about 99% level of reliability, which means that downtime is about 3.5 days per year [64, 92]. A failure analysis in 2003 indicated that the Internet Archive [61], a digital library that contains over 100 terabytes of compressed data and over 600 disk drives, suffered thirty disk failures within one month. Figure 1.1 plots the distribution of hardware problems collected at the Internet Archive. As shown in the chart, disk-related problems, including disk failure, disk error and disk subsystem problems, consist of 60% of all the recorded hardware problems. In a modern storage system as large as Google file system [40] and IceCube [126], failures are no longer treated as exceptions.

As Siewiorek and Swarz [106] pointed out, harsher environments, novice users, increasing repair costs, and larger systems have caused an increased concern towards fault tolerance and reliability. Let us get a glimpse of a modern large-scale storage system. A storage system containing two petabytes[2] of data is typically composed of 5,000 disk drives with 400 gigabytes each, based on the latest technology products. The challenges of high system reliability in such a large storage system can be summarized in the following four aspects:

- First, disk drive reliability has only improved slowly although advances in storage technology have reduced cost and improved performance and capacity. The gap of the progression paces has only been widened for the last decades due to the constrains of mechanical properties and increasing areal density on disk drives [49].

- Second, the increasing number of components makes infrequent failures likely in *some* device. The probability of a failure on any disk drive in a system composed of ten thousand disks is 1% within one hour, assuming the Mean-Time-To-Failure (MTTF) of disk is one million hours, as claimed by the disk manufactures. The one-million-hour MTTF number, however, is overestimated [36]. The likelihood that a drive would fail within only one hour period of time is as high as 1%.

- Third, the capacity of a single disk drive is fast increasing. Seagate Technology just released their 400 gigabyte serial-ATA drive in November, 2004. It takes four and half hours to rebuild such a disk at the bandwidth of 25 megabyte per second. With the advances in disk technology, disk capacity is still growing fast, but the increase in disk

[2] a gigabyte (GB) is 10^9 bytes; a terabyte (TB) is 10^{12} bytes; a petabyte (PB) is 10^{15} bytes; and an exabyte (EB) is 10^{18} bytes.

bandwidth has lagged behind and is outpaced by the increasing speed of disk capacity [53]. As a result, the large data capacity of each storage device lengthens data repair time and widens the window of vulnerability to data loss.

- Fourth, the complexity in interconnection architecture makes it difficult to achieve robust network connection. Besides storage devices, a petabyte system contains thousands of connection nodes such as routers and switches, and tens of thousands of network links. As the I/O requests arrive intensively, outages of these network connectors and links will result in packet loss or bandwidth congestion.

This thesis addresses these reliability challenges and discusses a family of reliability mechanisms to deal with failures in large-scale storage systems. The underlying foundation of this work is a good understanding of the characteristics of disk failures and the properties of system dependability. System reliability is analyzed quantitatively by Markov models and the important factors in designing robust petabyte-scale storage systems are identified based on the analysis and simulation results. The core of this dissertation presents the impact of failures in system aspects and evaluates data recovery approaches to ensuring high dependability with analytical analysis and simulations.

The rest of this dissertation is organized in the following manner: Chapter 2 presents the relevant research which provides the context of this work; Chapter 3 comprises an overview of large-scale storage systems. Data redundancy schemes and distributed recovery mechanisms are described and analyzed in Chapter 4. The schemes of data layout and their impact on system reliability are discussed in Chapter 5. Chapter 6 proposes the models of system

reliability based on the understanding of disk failures and presents the quantitative analysis for estimating reliability of large storage systems under various redundancy and recovery schemes. Chapter 7 demonstrates the impact of failures on network interconnections and compares several storage network topologies. Finally, Chapter 8 concludes and discusses the challenges of future work.

Chapter 2

Related Work

There has been active research in the areas of reliable storage systems and large distributed systems, including the investigation of system failures and their impacts, the modeling and analysis of system reliability, fault tolerance and failure recovery schemes. This chapter will discuss the essential ideas of the previous work, their relationship with this thesis work, and the uniqueness of this thesis research.

2.1 System Failures and Their Impacts

The Recovery Oriented Computing (ROC) project [14] points out that system failures are the problems that need to be coped with. Based on two surveys of causes for system downtime, it was found that human errors account for the primary cause of system failures. ROC improves system availability and reduces the total cost of ownership by decreasing Mean-Time-To-Repair (MTTR). The goal of this thesis work is similar to that of the ROC project,

but it focuses on failures of disk drives rather than all the components in the entire computer system.

In order to study the impact of failures on system reliability, a deep understanding of system failures is imperative. It is believed that there are various kinds of failures in large storage systems. There has been some research on system failure characterizations, however, failure behavior in petabyte-scale storage systems has not yet been studied since people are just starting building systems at this scale. This work pays particular attention to disk failures.

Patterson *et al.* [87] studied the causes of failures and found ”human errors” account for over 50% of the total failures, hardware failures are about 15% to 20%, software failures are about 8% to 34%, and overload failures are about 0% to 11%. When the scale of systems grows at a high rate, it is expected that human mistakes will take less impacts and problems caused by machine-generated errors will be significant.

A recent empirical study of operating system errors [22] examines various versions of the kernel sources and shows that the rate of device drive errors is much higher than the other kernel errors, many errors are not independent and they cluster at “a factor of two more tightly than from a random distribution.” Based on the data collected during seven years, interestingly, this study has found that an average lifetime of a bug is about 1.8 years. OS errors and failures are out of the scope of our research, although it is another source of system failures.

Characterizations of disk failures are difficult due to the lack of the long-term trace on a large system. The report of a 3.2 TB storage system [111] and the recently-collected data from the Internet Archive [61] have shown that failures coming from disk drives are not trivial

in the presence of thousands of storage nodes.

2.2 Reliability Modeling and Analysis

The study of modeling and analysis of reliability has been an active research field during the last thirty years due to its wide applications, such as industrial engineering, material engineering, and computer systems. It has become an important method to analyze large-scale storage systems with modeling and simulations since it is usually expensive to build a real large storage system in order to measure its reliability and examine many related system aspects. For a storage system composed of a large number of disk drives, there are two indispensable and related aspects that have to be considered: one is reliability modeling of an individual disk drive, and the other one is modeling of an entire storage system composed of multiple disks.

2.2.1 Modeling of A Single Disk Drive

The reliability of disk drives plays an essential role in overall system reliability; however, it is also "one of the trickiest drive characteristics to measure" [4] because of complex factors including duty hours, temperature, altitude, and spindle starts/stops [36]. Due to their mechanical nature, disk drives are one of the least reliable components of a computer system with MTTF varying between 80,000 to 800,000 hours [45]. Moreover, the server-type disk drives distinguish their reliability properties from the ATA drives. For an example, the typical nonrecoverable error rate of server drives is 1 in 10^{15}, while it is 1 in 10^{14} for ATA drives. Also, disk drives for personal storage (usually ATA drives) and enterprise storage (usually

SCSI drives) have different power-on hours. An analysis of a 3.2 TB storage system [111] has shown that failure rate of 24 IDE disks is 25%, whereas only 1.9% of 368 SCSI disks failed during the 18 month period of their prototype operations.

It is well-known that the failure rate of a disk drive has been described as a "bathtub curve," which reflects initially high failure rates and subsequent lower constant failure rates over the course of a disk's useful lifespan. However, an accurate disk reliability model does not yet exist because the exact distribution of disk drive failure rates is not known. Often, for the lack of a precise model, researchers assume a constant drive failure rate [8, 21, 57, 86, 102] and use the Mean Time To Failure (MTTF) as a metric of disk drive reliability. The discrepancy from actual disk behavior with the "infant mortality" is noted in several early RAID papers [15, 41, 101], but has been largely ignored since then. Disk failures are not easily disernable in small-scale storage systems due to the limited number of disks, so that "infant mortality" has little impact on small systems.

There are some effort towards more accurate models of disk failures. The Weibull distribution has been proposed as the qualitative estimation of the form of a bathtub-curve [109]. Elerath [36] claimed that Mean-Time-Between-Failures is a misleading metric because the failure distribution for disk drives is *not* exponential particularly in the first year. Instead, he built a more practical model, in which the drive hazard rate is in the form of a stair-step Weibull approximation.

In our work towards reliability modeling of very large storage systems, several statistical distribution assumptions on disk failure rate distributions are explored, from exponential distribution to bathtub-curve like distribution, at varying parameters. In order to evaluate the

effect of different disk failure models on system reliability, system behaviors are compared under different configurations. Our work also quantifies the effect of infant mortality on data reliability in storage systems with different scales.

2.2.2 Reliability Modeling of Storage Systems

As the number of disk drives scales up in a storage system, the distribution of system failure can not be easily calculated based on a single-disk model through simple operations, i.e. summation or multiplication. Far from a cumulative process, the quantitative increase of single elements in the entire storage system demands different models to determine failure behaviors, as well as the deployment of various data placement strategies and redundancy schemes in practice which make the description of system reliability even more complicated. Thus, a more elaborate model is derived.

There is a great deal of study on reliability modeling [46, 89, 107] using techniques such as combinational models, Markov models, safety modeling, availability modeling, and maintainability modeling based on various mathematical and statistical methods, including stochastic processes, Markov chains, Bayesian estimations, and statistical distributions.

Markov models are commonly used for modeling and analysis of large computer systems in order to investigate their reliability as well as performance. To study the transient and cumulative behavior of a computer system, Trivedi *et al.* [115, 116] have presented a series of techniques, including full-symbolic, semi-symbolic, and numerical methods. The two biggest problems with Markov models for complex computer systems are *largeness* of Markov state spaces, and *stiffness* which results from the simultaneous presence of time constants

of different orders of magnitude in a Markov model. For example, failure events in highly dependable are rare, while repair events are fast. Thus, there is a large gap between these two transition rates, which affects the computational efficiency greatly. Muppala *et al.* [84, 117] addressed that Stochastic Petri nets can solve the problems by automated generations of state spaces. Trivedi *et al.* have also developed "a proactive fault management technique to prevent the occurrence of more severe crash failures in the future" [114]. The method, called as "software rejuvenation," was aimed at cleaning up the system internal state and has been implemented in the IBM x-series servers.

Recently, Brown [13] argued that modern computer systems are too complex to be modeled for the variety of computer components. Clearly, it is very expensive to build a real system and conduct many testings for various schemes in order to make the best design decision. However, modeling of disk failures is feasible, because only considering disks failures rather than the entire system will make modeling more practical. Although there is a gap between system models and "real" systems, modeling is still an effective technique to examine the properties of a system.

In this thesis work, Markov models are built to investigate system reliability properties. Several statistical methods are also utilized in the analysis. Chapter 6 will discuss the details of these models.

2.3 RAID and RAID-like Systems

There has been a great deal of research on highly reliable file and storage systems. Among them RAID is one of the most widely used methods.

RAID (Redundant Arrays of Independent Disks) [20, 21] is a classic method in the field of reliable storage systems. Aimed at eliminating I/O bottleneck, RAID exploits inexpensive disk drives and various data placement policies including striping, mirroring, and parity among disks which can achieve high data availability and high I/O throughput, thereby providing customers with improved fault tolerance and data loss protection. RAID technology has been further studied in the areas of data placement policies and load balancing mechanisms [39].

Striping is widely used for improved performance, superior scalability, and high reliability. Stonebraker and Schloss [108] first proposed the idea of distributed RAID system which supports redundant data copies across a computer network to increase availability in the presence of site failures and disk failures in a storage system. However, they did not examine various data reconstruction algorithms and only used identical-sized data groups. In this thesis research, several data group configurations and reconstruction schemes will be studied.

To support high data transfer rates in general distributed systems, Swift system [16] is built on the principle of "striping data over multiple storage agents and driving them in parallel." The prototype and simulation of the Swift architecture have validated its high scalability and improved I/O performance. Later, a serverless distributed network file system, xFS [6, 121] was proposed. By eliminating central servers and distributing completely the file

system, xFS scales up the aggregate throughput of the file system with the number of nodes in the system, which resulting in a tremendous performance improvement. Petal [66] was presented later. It is a distributed block-level storage system which provides a layer of abstraction between the clients and the actual storage system known as a "virtual disk." It supports two kinds of data placement policies: one is simple data striping without redundancy, and the other is the chained-declustering redundancy scheme, which is similar to Zebra [51]. The extension of Petal, Frangipani [113] is a scalable distributed file system for storing data on a Petal virtual disk. Aimed at the same goals as xFS– high scalability, good availability, and improved performance, Frangipani has addressed file system recovery and reconfiguration which is an open question in xFS.

To provide an easy-to-use and high performance file system for highly parallel scientific computing applications, Miller and Katz proposed RAMA (Rapid Access to Massive Archive) [78, 79]. This parallel file system allows massively parallel processors to "use all the nodes for both computation and file service" by placing multiple disks at every processor node. In addition, RAMA exploits hashing to distribute data pseudo-randomly, thus avoiding performance degradation caused by poor striping configurations. It was also pointed out in the design of RAMA that optimal data striping is difficult to achieve.

Data allocation in Slice [5] is also made by distributed storage servers, thus removing bottlenecks of a central server. Some other storage systems, such as GFS [90], are less scalable since data distribution is managed by clients, although files are striped across the system.

Similar to these previous striped distributed file systems, data in our system is distributed across thousands of storage nodes and the data allocation is done in a decentralized

fashion so that there is no central bottleneck when a single file is accessed by thousands of clients at the same time. There are two unique characteristics of our system which are different from the previous systems addressed above: first, data is broken to objects and striped across the Object-based Storage Devices (OSDs) [3, 123], which differs from disks in that they can deal with low-level storage management intelligently; second, our system composes of thousands of disk drives of large capacity, resulting in a petabyte-scale storage system.

2.3.1 Mirroring and Parity

Mirroring and replication have been used more frequently as the price of disk drives keeps decreasing. Compared with parity, mirroring is always easier to implement in practice and can avoid problems which come along with parity such as small writes, complex recovery, and poor performance from disk rebuilding. EW Disk Array (Eager Writing Disk Array) [128] combines the techniques of eager writing and mirroring in order to achieve scalable system performance. Long [72] has presented an efficient algorithm for managing the consistency of mirrored disks. This low-cost algorithm uses "cohort sets" to run after the last participated disks in a write operation and track the current data.

On the other hand, parity provides a good method for system reliability with low storage overhead. However, parity-based storage systems, for example, RAID 5 system, have their own limitations, such as the small-write problem. Hwang *et al.* [59] brought up distributed RAID architecture for I/O intensive application environments. Based on a novel orthogonal striping and mirroring configuration, this architecture avoids the small write problem associated with RAID 5 and reduces I/O latency, but writing clustered mirrored blocks would

be slow for large data blocks, which is common for large-scale storage systems. In order to break the limitation that two or more sectors in a group (composed of data and parity blocks) cannot be updated at the same time in a RAID 5 system, Yeung and Yum [127] developed Dynamic Multiple Parity (DMP) Disk Array for transaction processing to achieve the speed requirement of I/O devices in database systems. RAID 5DP (RAID 5 Double Parity) [24], different from RAID 5, provides higher data protection by using two independent parity calculations. Since each stripe is composed of data sectors and two check sections, RAID 5DP can tolerate two disk failures while maintaining low overall storage cost.

In addition to RAID 5 parity scheme, erasure correcting codes such as Even-Odd codes and Reed-Solomon block codes [2, 88, 102] are studied and used to provide redundancy, thus protecting data against disk failures. Row-Diagonal Parity [25] is an algorithm that protects double disk failures, in which each data block belongs to one row parity set and to one diagonal parity set. It is simple to implement and optimal in computational complexity. Myriad [19] protects data by forming cross-site redundancy groups which are composed of data and checksum blocks from different nodes based on the technique of erasure codes. Litwin *et al.* [67, 68, 69, 70, 71] presented a family of linear hashing methods for distributing files. The proposed algorithms reduce the number of messages and scale to the growth or shrink of files efficiently. Their scalable distributed data structures have the properties of high data availability and storage efficiency by using mirroring, striping, and parities such as Reed Solomon codes and other standard techniques.

Based on shared-disk architecture [81], IBM's GPFS [100] is a highly scalable parallel file system used for cluster computers composed of a large number of nodes and disks. In

order to ensure that the entire system work correctly in the presence of failures, GPFS handles node failures by recovery logs on shared disks and uses the distributed locking protocol to ensure metadata consistency. GPFS stripes metadata and data across all disks and uses replication and RAID configurations to protect data from loss due to disk failures. However, the replication schemes or RAID configurations have not been discussed in detail, and the long disk rebuild time has not been considered in the GPFS system.

The growing capacity of disk drives increases the width of "window of vulnerability": it takes longer to rebuild a bigger disk drive because the ratio of bandwidth to capacity is continuously decreasing. Simply using RAID cannot guarantee enough system reliability for a petabyte-scale storage system. This observation motivates our research.

2.3.2 Rebuild in RAID Systems

The performance of disk arrays under failures is very important. Back to the early 90's, Muntz and Lui [83] have pointed this out: "This is of importance since single disk failures are expected to be relatively frequent in a system with a large number of disks". Menon and Mattson [75] proposed distributed sparing, where the spare disk is broken up and distributed through the array. In such an array, the reads for a data rebuild are distributed and so are the writes (to the spare space). Their analysis has shown that distributed sparing is able to improve performance by exploiting unused spare disks. Later, Alvarez *et al.* [2] presented DATUM, a method that can tolerate multiple failures by spreading reconstruction accesses uniformly over disks based on information dispersal as a coding technique. DATUM accommodates distributed sparing and uses a suite of functions that map each client address to the set of disks

and sectors. In order to shorten the rebuild time of the failed disk, Muntz and Lui [83] proposed to declustering a disk array of n disks by grouping the blocks in the disk array into reliability sets of size g. The principle of this idea comes close to the spirit of fast recovery schemes, which will be discussed in Chapter 4. Our fast recovery mechanisms, however, are not only used for avoiding performance degradation under disk failures, but also, and more importantly, for improving reliability. The other distinct aspect of our research is that we are investigating this problem in a very large-scale storage system composed of thousands of disks: recovery in such a large-scale system has not well been studied.

2.4 Highly Reliable File and Storage Systems

Research towards reliability issues in large storage systems has been very active recently as systems are scaling up to satisfy the increasing storage needs.

Google file system [40] treats failures as common events. In the Google file system, a single master makes decisions on data chunk placement and replications. A data chuck is re-replicated when the number of its replicas falls below a limit specified by users. Grid DataFarm [110] stores files as replicated fragments in distributed storage resources. Our work considers more general redundancy schemes like erasure coding, while much less aggressive erasure coding settings are used in our system than the OceanStore system [65].

OceanStore [65, 93, 94] is an infrastructure that provides high availability, locality, and security by supporting nomadic data. It uses erasure coding to provide redundancy without the overhead of strict replication and is designed to have a very long Mean-Time-To-Data-

Loss (MTTDL) [122], but at the cost of dramatically increasing the number of disk requests for writing a block. They use very aggressive erasure coding settings such as 32/64[1] which only fits well in a read-only system like OceanStore, but increases the complexity during updates and reduces performance under degraded modes for a storage system where writes are equivalently frequent as reads. It was also assumed in OceanStore that the availabilities of data blocks are independent, however, this is not true in our system. Another difference is that data objects in our system have large size so that it is not practical for us to make use of promiscuous caching.

Microsoft's FARSITE project [1, 30, 31, 32, 33] studies a serverless distributed file system that provides improved file availability and reliability, high system security and superior scalability. Based on the data collected from a large number of personal computers, they studied the file system properties and found that disk usage can be greatly reduced by eliminating duplicate file contents. They found that the average machine lifetime is about 300 days. As one of the important parts and the most relevant to our research, replica management is discussed in the design of FARSITE system, including replica placement policies, replication factors, the identity of encrypted replicas, and the reclamation of used space from duplicated files. Their large-scale simulations indicate that the random replica placement is better than the replacements that consider availability at initialization, for reasons of the distribution of free space and evenness of machine reliability. They proposed three hill- climbing algorithms (MIN-MAX, MIN-RAND, and RAND-RAND) used to improve the placement by monitoring machine availability and relocating replicas by swapping. They found out that MIN-RAND

[1] 32/64 is an erasure coding scheme which guarantees no data loss occurrence as long as 32 or more out of total 64 pieces in this data group are available.

method provides a reasonable trade-off between efficiency and efficacy. The replication factor affects the system availability and reliability, which are exponentially sensitive to the number of distributed replicas. It has been shown that failures of machines are independent if the replication factor is smaller than four. A self-arranging, lossy, associate database is used to identify the replicas in a decentralized, scalable manner.

There are three major differences between our storage system and FARSITE. First, it was assumed that FARSITE be used in an untrusted environment, so that replicas were encrypted in order to achieve higher security. Our system, instead, is considered to be trustable so that the security of replicas has not been taken into account. Second, they focused on system availability but we emphasize reliability. Third, our system is designed to provide high performance for large-scale scientific computations, while FARSITE was designed to support typical desktop workloads in academic and corporate environments.

2.5 Dependability in Wide-Area Distributed Systems

There has been research beyond RAID in reliability and recovery for large-scale systems which has focused on the use of storage in wide-area systems. Pangaea [98, 99] is a peer-to-peer wide-area file system that provides fault tolerance by pervasive replication, and allows individual servers to continue serving most of their data even when disconnected. It has two types of replicas: *gold* and *bronze* which differ in that *gold* replicas play an additional role in maintaining the hierarchical name space. However, this approach is designed for wide-area systems, while it is not practical for our system to have up to thirty replicas. In addition,

Pangaea does not permit files to be striped across dozens of servers, but striping is an effective method for enhancing bandwidth in a very large-scale storage system.

Other peer-to-peer file systems such as PAST [96], CFS [27], FreeNet [23], Gnutella [95], and Glacier [50], are designed with high replication for untrusted environments. In contrast, our system is assumed to be in a safe environment. Also, large replication factor, for example, more than four, would not practical in consideration of storage efficiency.

In our system, replication is used mainly for the purposes of high reliability and also for performance concerns. The protocols of replication will not be discussed in this work.

2.6 Replication and Recovery Protocols

Castro and Liskov [17, 18] have proposed a practical Byzantine fault tolerance system that uses a new proactive recovery scheme for Byzantine-fault replicas and symmetric cryptography for security. The asynchronous state-machine algorithm in the system also provides detection of denial-of-service attacks, thus narrowing the "window of vulnerability." The goal of the fast recovery schemes in this thesis work is aimed at reducing "window of vulnerability" to deal with disk failures, but not for Byzantine faults caused by malicious attacks and software errors.

To provide multi-level fault tolerant ability and make the system more reliable, Vaidya [119] presented a "two-level" recovery scheme in distributed systems that makes use of both 1-checkpoint and N-checkpoints to tolerate more probable failures with low overhead.

Goodson [48] discussed a set of protocols appropriate for constructing block-based

storage and metadata services for fault-tolerant storage systems. These protocols focus on strong consistency which is enabled by versions maintained by storage nodes.

Replication is one of the interesting topics in distributed systems as well as in database system and other areas. Recently, Wiesmann *et al.* [125] have presented a functional model to compare existed replication protocols in database and distributed systems. They pointed out that replication in distributed systems is used "mainly for fault tolerance purpose" and "mainly for performance reasons" in databases. (We believe that replication in databases is used for fault tolerance as well.) The major differences between distributed systems and databases are: (1) distributed systems distinguish synchronous between asynchronous system model but databases do not; (2) Abnormal behaviors in distributed systems usually do not need operator interventions but database replication protocols may admit operators to deal with the failure of a server. They also pointed out that there are several similarities of distributed systems and databases, which implies that the integration of these two communities can help to get a better replication protocol. Ji *et al.* [62] recently present a remote mirroring protocol–Seneca.

2.7 Interconnection Network for Storage Systems

Designs for parallel file systems, such as Vesta [26] and RAMA [79], were aimed at high performance computing, but did not consider the large scale of today's systems and the impact of failures which comes along with such a scale.

Industry solutions, such as IBM TotalStorage Infrastructure [60], EMC SAN arrays

using fibre channel switches [63], and Network Appliance Fibre Channel SAN storage solutions [85], can support up to tens of terabytes data capacity. Our work is aimed for the design of petabyte-scale storage systems that can provide non-stop data delivery in the presence of failures.

Hospodor and Miller [56] explored potential interconnection architectures for petabyte-scale storage systems. The main concern of their work is efficient network interconnection between storage nodes, routers, switches and servers; however, they do not consider the consequences of failures on the network.

One of the main approaches to achieve fault-tolerant interconnection networks is adding redundant nodes and links in arrays and meshes. Blake and Trivedi [9] analyzed the reliability of multi-stage interconnection networks and showed that adding intra-stage links is more beneficial than adding an extra stage. Zhang [129] designed fault-tolerant graphs with small degree for good scalability and small number of spare nodes for low cost.

Resilient routing is crucial for interconnection network reliability in the face of link outages. Vaidya *et al.* [118] studied fault-tolerant routing algorithms in a multiprocessor system. Their focus was Duato's routing algorithm—Double East-Last West Last (DELWL) [34] in a 2D mesh topology. The system they examined was on the order of thirty to forty nodes; while our study is for much larger scale storage networks—on the order of thousands of nodes. In addition, we have switches and routers which differ in functionality from storage nodes.

The MIT Resilient Overlay Networks (RON) architecture [80] provides reactive routing to path failure detection and recovery on large-scale, Internet-based distributed systems. It monitors the quality of paths by frequent probing between nodes, and stores probe

results in a performance database. In principle, there is much less control on the overall architecture of an Internet network such as RON than our storage network. Also, we consider both node failure and link failure while RON only focuses on path failure.

2.8 Summary

This chapter introduces the background and the relevant research in the areas of reliability modelings of disk drives and storage systems, highly reliable RAID system and other distributed file and storage systems, system failures and their impacts, and the interconnection networks for storage systems. In contrast to the previous work, this dissertation focuses on the mechanisms to ensuring high reliability of large-scale storage systems based on a better understanding of disk failure characteristics and failure impacts. A family of redundancy schemes are studied and analyzed, and the fast recovery mechanism is evaluated in large-scale storage systems, by using more accurate disk failure models. Additionally, the failure impact on the network interconnections is examined in a petabyte-scale storage system built with several different architectures . The *large system scale* makes this work challenging and meaningful. The thesis will present our work on understanding and improving reliability of petabyte-scale storage systems in the following chapters.

Chapter 3

System Overview

This thesis research targets the reliability of a storage system containing multiple petabytes of data, which is primarily used in supercomputing environments for large-scale scientific applications. In such a system, some data is difficult— perhaps even impossible— to regenerate and may not be reproducible, such that a high level of reliability is required. Besides the supercomputing environment, this work is also applicable to general large-scale storage systems which require high data reliability, especially systems for data-driven business servers and online repositories of critical client data.

As part of the project “Scalable File Systems for High Performance Computing” sponsored by the US Department of Energy, this thesis work examines reliability of a system based on object-based storage devices. This chapter first introduces the background of object-based storage systems, and then presents the overall picture of our system.

3.1 Background: Object-based Storage Systems

Object-based storage devices (OSDs) [76] are as the underlying storage nodes in our system for better scalability; however, our analysis and models are also applicable to traditional block-based storage systems.

The development of storage systems has come to a point that the interface of storage devices should be altered. Inspired by the idea of Network Attached Secure Disks (NASD) [42, 43], the concept of object-based storage devices (OSDs) [3] was proposed. Rather than relying on the traditional "block based" interface, an OSD-based storage system uses objects, which combine the advantages of files and blocks, to store the entire data structures. Objects provide direct, file-like access to storage devices, which manage those variable-sized objects intelligently with the support of a set of additional commands, moving low-level storage management into the storage devices themselves. Slice [5], Lustre [11], and Swift [73] have discussed the use of OSDs to store data. Such OSD-based storage systems enable efficient large-scale data storage since low-level storage management (such as request scheduling and data layout) can be handled by storage devices themselves, making the systems more scalable. As for security concerns, object storage offers secure direct access without consulting central servers and separates the flexible security policies set by applications from the OSD's mechanism for enforcing security. Semi-intelligent OSDs also allow security policies to be enforced at the disk instead of relying on a central point of access.

3.2 System Architecture

Figure 3.1 shows the architecture of our system. The main hardware component of our file system is an object-based storage device: one or multiple disks drives managed by a single CPU and seen by the file system as a single device. The data in the system is distributed across many OSDs, with high bandwidth coming from large numbers of concurrently operating OSDs. In this system, the high-level information such as translation of names of file identifiers and data striping patterns across OSDs is managed separately by specialized metadata servers. A cluster of tens of metadata servers provides scalable access to the file system hierarchy by the dynamic metadata management [124], addressing scenarios such as a large number of clients accessing the same directory. Compared to the traditional architectures in which metadata and data are managed in a monolithic fashion, the separation of data from metadata management can facilitate greater system scalability and efficiency.

Our system has the following principle design goals:

1. **Very Large Scale** Data storage capacity in the system is at the order of magnitude of petabytes (2^{50} bytes), with additional storage capacity for redundancy information. There are thousands of OSD nodes in the system, and tens of metadata servers. The storage nodes will use disks with individual disk capacity in the hundreds of gigabytes.

2. **High Performance** Files are broken up into objects and striped across the entire OSD cluster. Read and write operations are done in a distributed fashion. Accordingly, the aggregate I/O bandwidth is greatly increased. The separation of data and metadata storage and management further provides high access bandwidth to the large-scale distributed

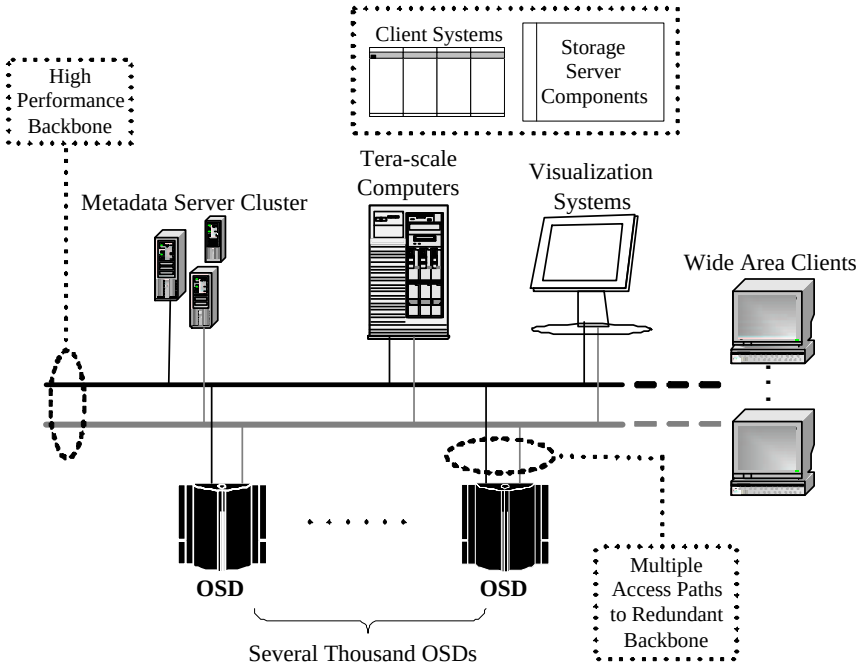


Figure 3.1: The overall architecture of a petabyte-scale storage system for supercomputing environment.

storage system.

3. **High Scalability** Metadata is managed separately by one or more specialized metadata servers, which is critical to system scalability. The OBSD project team has designed an efficient lazy hybrid metadata management scheme [12] for metadata server clusters and a fast online data reorganization algorithm [55] to deal with the data placement problem in the presence of adding or removing disks.

4. **High Availability** Fast network and good management protocols make the nodes in the system well coordinated. Superior data distribution policies will eliminate data access ”hot spots” and increase availability of the system.

5. **High Reliability** Efficient failure recovery and smart redundancy mechanisms will reduce the impact of system failures and ensure high system reliability. This is the theme of this dissertation. A brief overview of the reliability challenges in large-scale storage systems is provided below.

3.2.1 Workload Characteristics

A set of file system traces[1] were collected on a large Linux cluster with more than 800 dual processor nodes at the Lawrence Livermore National Laboratory (LLNL). Thirty-two file servers were used for the Linux cluster during a science run with 1.4 terabytes storage capacity on each server. On average, there were 350,250 files and 1.04 terabytes of data on each server, which took more than 70% of their capacity. Except for five file servers, the number and capacity of files are similar on most servers. The file servers were dedicated to a small number of large-scale scientific applications, which provides a good model of data storage patterns for the types of applications we are interested in.

Figure 3.2 presents file size distributions by number and file capacity. The ranges of file sizes were sampled from 0–1 byte to 1–2 gigabytes. Some of the partitions were merged due to space limitations. The size distribution in this large-scale file system is generally in accordance with Gaussian distribution with an average file size of about 3 megabytes, while most disk space is consumed by files larger than 1 megabytes. It is observed that over 80% of the files were between 512 kilobytes and 16 megabytes in size and these files accounted for over 80% of the total capacity. Among various file size ranges, the most noticeable one is

[1]Thanks to Tyce T. McLarty at Lawrence Livermore National Laboratory for providing these traces.

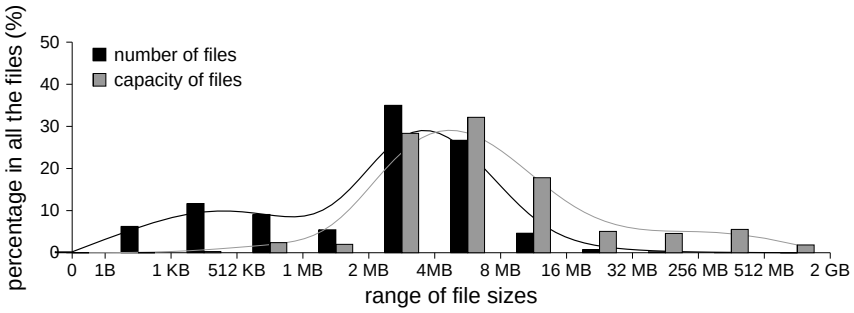
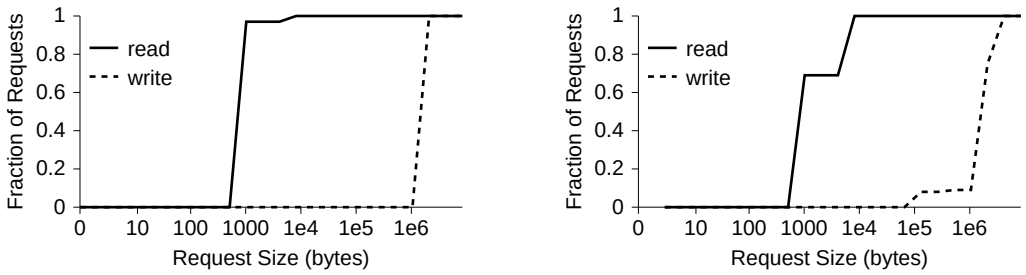


Figure 3.2: Distribution of File Sizes

from 2 megabytes to 8 megabytes: about 61.7% of all files and 60.5% of all bytes are within this range.

To profile I/O workload characteristics of such kind of storage systems which support large-scale scientific applications, I/O activities of two typical parallel scientific applications were traced during July, 2003. The first application, referred as *f1*, is a physics simulation run on 343 processes. In this application, a single node gathers a large amount of data in small pieces from the others nodes, then a small set of nodes write these data to a shared file. The second application, as called *m1*, is another physics simulation which runs on 1620 nodes. This application uses individual output file for each node and dumps the results to the underlying storage pool.

The traces record each I/O request in *f1* (last about 90 minutes) and *m1* applications (last about 4 minutes). Figure 3.3 shows the cumulative distribution functions (CDF) of the I/O requests in the two applications respectively. Most write requests are larger than one megabyte, and most read requests are larger than one kilobyte in the *f1* application. Similarly,



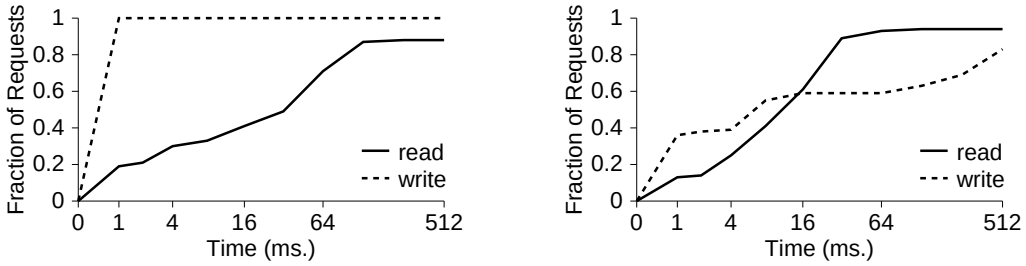
(a) f1 application

(b) m1 application

Figure 3.3: Cumulative Distribution Functions (CDF) of the size of I/O requests over the request size in *f1* and *m1* applications (X axis in log-scale).

more than 84% of the read requests are larger than sixty four kilobytes and more than 91% of the write requests are larger than one megabyte in the *m1* application. The distribution of I/O request sizes shows that large I/O requests are dominant in typical large-scale scientific applications.

The intensity of I/O requests was also studied for the *f1* and *m1* applications and the cumulative distribution functions (CDF) of I/O request inter-arrival times are plotted in Figure 3.4. Write activities are very intensive in the *f1* application: nearly all the write requests have inter-arrival time within 1 millisecond. Read requests are generally less intensive than write requests because reads are synchronous. As indicated in Figure 3.3(a), half of the read requests in the *f1* application arrive at the rate of 1 request in 32 millisecond. In the *m1* application, write requests are more intensive than read: about 36% of the write requests and 13% of the read requests have the inter-arrival time of less than 1 millisecond, while 59% of the writes and 89% of the reads arrive at the rate of 1 in 32 milliseconds. In summary, the I/O



(a) f1 application

(b) m1 application

Figure 3.4: Cumulative Distribution Functions (CDF) of Inter-arrival Time of I/O Requests in *f1* and *m1* applications (X axis in log-scale).

requests are intensive in these large-scale scientific applications.

3.2.2 Reliability Concerns in Large-Scale Storage Systems

Whereas a storage system with two petabytes of data poses unique opportunities for system performance, it also provides unique challenges on reliability. With thousands of nodes in the system, it is unlikely that all components will function properly at all times; even infrequent failures are likely in *some* part of the system. Reliability mechanisms including redundancy schemes, failure detection, and failure recovery schemes must be used to safeguard the system in the presence of various failures such as nodes failures, communication failures, software crashes, power outages, and other hardware failures. Among them, disk failures should receive particular attention since they lead to data loss, which cannot be tolerated in storage systems for scientific applications. In addition to the very large scale, disk rebuild times are becoming longer as increases in disk capacity outpace increases in bandwidth [53].

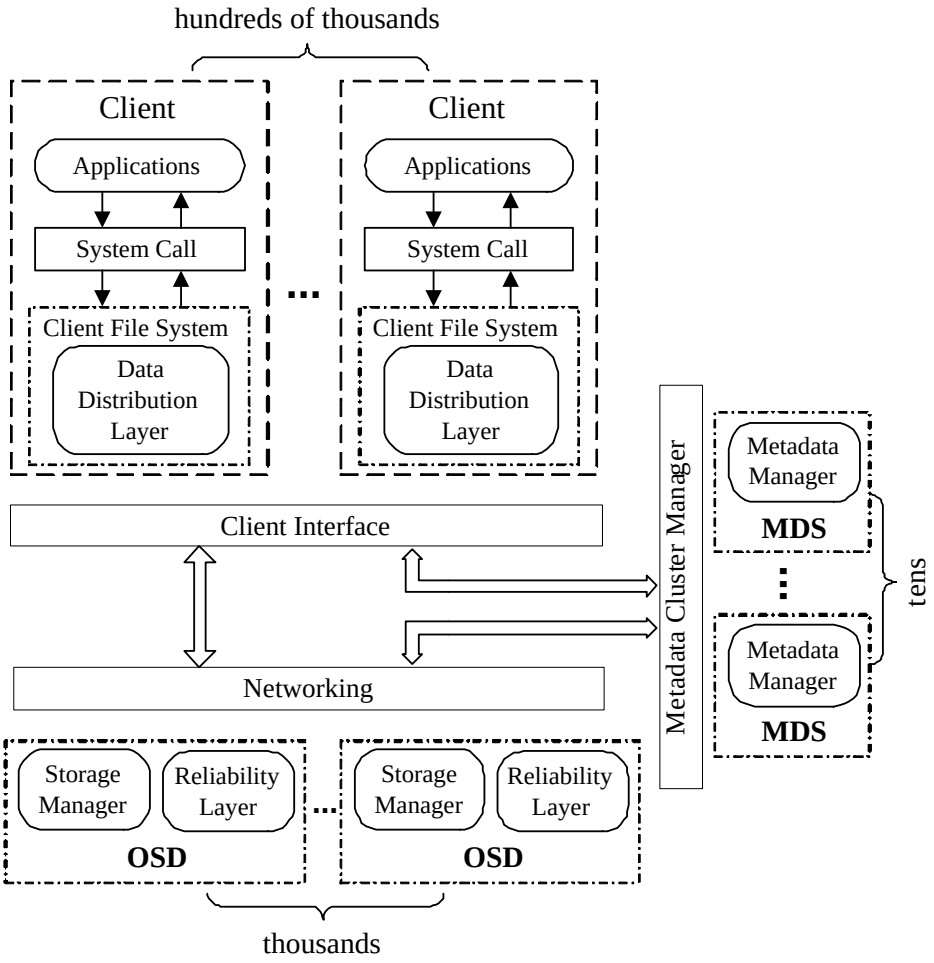


Figure 3.5: OSD Model

The “window of vulnerability” is lengthened and the chances of data loss are increased.

For storage systems that are built to support large-scale scientific simulations and experiments, it may take a long time to obtain the desired results, and even worse, the results usually consist of a large amount of data that has to be stored across many storage nodes. If one node fails, the scientists might have to re-run the whole simulation or experiment only because one part of the results is lost due to disk failure. This will greatly increase the computational cost. To make the matters worse, some experiments are difficult to repeat and data may not be reproducible.

These characteristics make the reliability problems of our system different from those of previous systems in that normal RAID architectures alone cannot meet the requirements of system reliability. Our research, thereby, is aimed to explore novel reliability schemes in this kind of very-large scale object-based distributed storage systems.

The object-based storage management in the system leaves the responsibility of low-level storage allocation to individual OSD nodes, while letting the higher-level file system determine data layout and redundancy. Accordingly, we develop various redundancy mechanisms in order to reduce the risk of data loss in the system. The OSD system model is shown in Figure 3.5, where the data distribution layer and reliability layer are directly related to our reliability concerns.

3.2.3 Data Distribution

In order to achieve high reliability in large systems, user data is stored redundantly using either replication or some form of erasure correcting code such as storing the parity of a

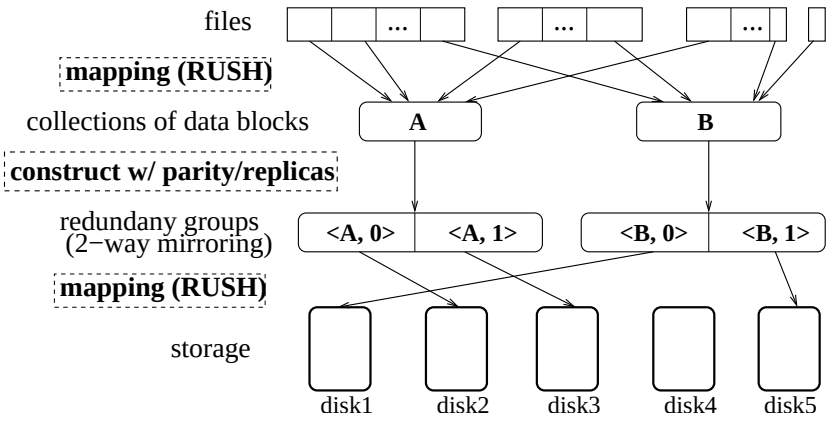


Figure 3.6: Redundancy group construction.

group of blocks. A group of data blocks composed of user data and their associated replicas or parity / erasure code blocks are referred as a *redundancy group*. The size of a redundancy group is the total user data in the group, not including the replicas or parity blocks. RUSH [55] algorithm is used to allocate data and its redundancy across the nodes among the system. RUSH is a pseudo-random mapping function that supports data replication, system growth, and disk obsolescence.

In our system, files are broken up into fixed-size *blocks*; the default size of a block is 1 MB. A number of blocks are gathered in a *collection* with the mapping of block to collection done by RUSH [55]. A collection is then assigned to a *redundancy group* by a redundancy scheme, such as replication or adding parity blocks. Collections of blocks enable efficient data management and redundancy groups provide enhanced reliability. RUSH is then used again to allocate redundancy groups to storage devices. Figure 3.6 illustrates the construction process of redundancy group A and B under a two-way mirroring configuration. Each data block is marked as $\langle grp\_id, rep\_id \rangle$, where *grp_id* is the identifier of the group to which it belongs and

rep_id is its identifier in the group. Data blocks that reside in the same redundancy group are called "buddies;" they share one *grp_id* with various values of *rep_id*. The options of the configuration of redundancy groups will be elaborated in Chapter 4.

3.3 Summary

Our system is a petabyte-scale data storage cluster with thousands of storage components which can either be object-based storage devices or traditional block-based disk drives. The most significant feature of this system is its huge scale. This chapter has described the system architecture, demonstrated the reliability demand of the system, and described data distribution and redundancy groups over the storage devices. Next chapter will present the fast recovery mechanisms and examine various factors of system reliability.

Chapter 4

Data Recovery Schemes

Storage clusters consisting of thousands of disk drives are now being used both for their large capacity and high throughput. However, their reliability is far worse than that of smaller storage systems due to the increased number of storage nodes. RAID technology is no longer sufficient to guarantee the necessary high data reliability for such systems, because disk rebuild time lengthens as disk capacity grows. This chapter presents FAst Recovery Mechanism (FARM), a distributed recovery approach that exploits excess disk capacity and reduces data recovery time. FARM works in concert with replication and erasure-coding redundancy schemes to dramatically lower the probability of data loss in large-scale storage systems. The essential factors that influence system reliability, performance, and costs have been examined, such as failure detection, disk bandwidth usage for recovery, disk space utilization, disk drive replacement, and system scale, by simulating system behavior under disk failures. Our results show the reliability improvement from FARM and demonstrate the impacts of various factors on system reliability. Using our techniques, system designers will be better able to build multi-

petabyte storage systems with much higher reliability at lower cost than previously possible.

FARM (FAst Recovery Mechanism) makes use of declustering [2, 75, 83] to reduce the time required to deal with a disk failure. RAID designers have long recognized the benefits of declustering for system performance. Declustering distributes the mirrored copies or redundancy groups across the disk array, so that, after a disk failure, the data needed during the reconstruction process is distributed over a number of drives in the disk array. In this way, bandwidth of multiple disks can be well utilized and the time period of the reconstruction can be greatly shortened. Declustering thus leads to good performance for storage systems in degraded mode. FARM uses declustering not only to improve performance during data recovery but also — more importantly — to improve reliability. FARM deals with the consequences of failures much faster and thus limits the chance that additional failures will lead to data loss during the window of vulnerability. The distributed recovery scheme is evaluated in a petabyte-scale storage system. Our simulated results show that FARM improves reliability across a wide range of system configurations.

To the best of our knowledge, no research yet has been directed towards the architecture of such a high-performance, large-scale system with such high reliability demands. OceanStore [93] aims for a high availability and high security world-wide peer-to-peer system, but does not provide high bandwidth. FARSITE [1] stores data in free disk space on workstations in a corporate network. While both systems have concerns similar to ours, they have less control over the individual storage devices and they provide primarily read-only access at relatively low bandwidths (megabytes per second). In principle, the fast recovery mechanisms are closer to the classical storage solutions such as RAID [21]. However, the size of our system is

so much larger that simply using traditional RAID techniques alone will not provide sufficient reliability.

FARM is studied in a petabyte-scale storage system with thousands of storage devices. Storage systems built from Object-based Storage Devices (OSDs), which are capable of handling low-level storage allocation and management, have shown great promise for superior scalability, strong security, and high performance [42]. Our mechanisms not only provide high reliability for object-based storage systems, but also are applicable to general large-scale data storage systems built from block devices. The term "disk drives" refers to both OSDs and traditional block devices.

Our simulation results show that FARM is successful across most data redundancy techniques. Other factors that influence system reliability are investigated, including failure detection latency, data recovery bandwidth, disk space utilization, disk drive replacement, and overall system scales. With our reliability schemes and analysis for related factors, system designers can choose the techniques necessary to ensure that their storage system is sufficiently reliable for their users.

4.1 FAst Recovery Mechanism

As the increase in capacity outpaces that of bandwidth, disk rebuild time is increasing. During the rebuild process, additional disk failures can lead to data loss. Traditional recovery schemes such as RAID that rebuild the data on a failed drive onto a single replacement drive cannot guarantee sufficient reliability in large storage systems, as we will see in Sec-

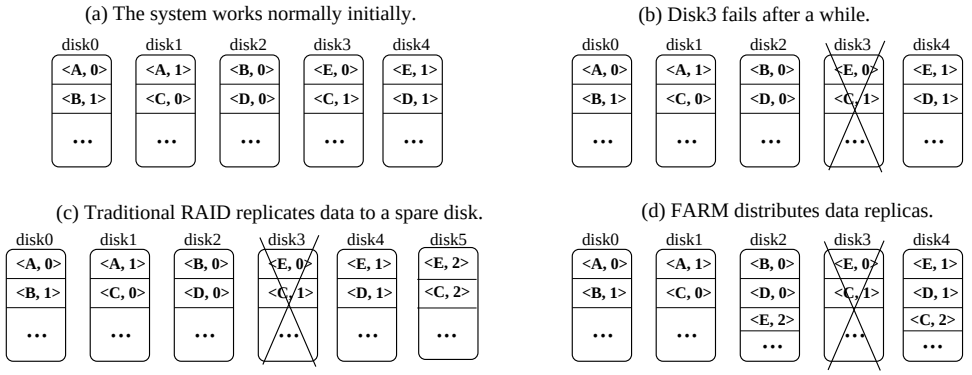


Figure 4.1: Upon disk failure, different behaviors with and w/o FARM.

tion 4.2. We propose FAst Recovery Mechanism (FARM) that declusters redundancy groups to speed up recovery from disk failure and thus achieves high reliability for very large-scale storage systems. In FARM, failure and recovery are transparent to users.

4.1.1 Configuration of Redundancy Groups

As discussed in section 3.2.3, data blocks are collected into *redundancy groups*.

No redundancy scheme is simpler than replication. An *n*-way mirroring scheme maintains *n* copies of each user data block, each residing on a different disk. Alternatively, a parity scheme adds a block containing the (XOR) parity of the user data blocks. For higher failure tolerance, we can use an erasure correcting code (ECC) to generate more than a single parity block. These *generalized parity blocks* are made up of the check symbols of the code.

There are many good candidates for an ECC. Since disk access times are comparatively long, time to compute an ECC is relatively unimportant. A good ECC will create *k* parity blocks of the same size as the *m* user data blocks. It will be able to reconstruct the contents

of any block out of m parity or data blocks. Examples of such ECCs are generalized Reed-Solomon schemes [70, 88] and EvenOdd [10]. These are generically called m/n schemes, where $n = m + k$. An m/n scheme gives us m-availability, but must update all k parity blocks whenever a data block changes. If only a single block changes, this can often be done as in RAID 5 by calculating the difference between the new and the old user data. The difference is then propagated to all parity blocks, which are updated by processing the difference and the old parity values.

The blocks in a redundancy group reside on different disks, but each disk contains data that belongs to different redundancy groups. We allocate redundancy groups to disk drives in a fully distributed fashion using an algorithm [55] that gives each disk statistically its fair share of user data and parity data, so that read and write loads are well balanced. A good data placement algorithm limits the amount of data any disks contributes to the data recovery process; the one we are using has this property.

The trade-offs between the different redundancy mechanisms are ease of implementation, the complexity of parity management and recovery operations, the bandwidth consumed by recovery [122], and the storage efficiency, *i. e.*, the ratio between the amount of user data and the total amount of storage used. Two-way mirroring is easy to implement and simple to run, but only has a storage efficiency of 1/2. An m/n scheme is more complex and write performance is worse, but it has better storage efficiency of m/n. There are some schemes that put a user data block into more than one redundancy group [52] and mixed schemes that structure a redundancy group by data blocks and an (XOR-)parity block, and a mirror of the data blocks with parity.

The consequences for reliability of each scheme depend greatly on the system in which they are deployed. For example, the FARSITE implementers noticed that failure of storage sites could no longer be considered independent for a replication factor $m > 4$ [32]. In a large storage system, placement and support services to the disk introduce common failure causes such as a localized failure in the cooling system.

4.1.2 Design Principles

The traditional recovery approach in RAID architectures replicates data on a failed disk to one dedicated spare disk upon disk failure. Such a scheme works properly in a small system consisting of up to one hundred disk drives, but it fails to provide sufficient reliability for systems with thousands of disks.

Menon and Mattson [75] proposed the distribution of a dedicated spare disk in a RAID 5 system among all the disks in the disk array. Each disk then stores blocks for user data, blocks for parity data, and blocks reserved for data from the next disk to fail. Some time after a failure, a replacement disk is rebuilt from the data distributed to the storage array from the failed disk. *Distributed sparing* results in better performance under normal conditions because the load is divided over one more disk, but has the same performance in "degraded mode" (after a disk failure). In addition, reliability benefits greatly from reduced data reconstruction time after a disk failure [103], because of a smaller window of vulnerability to further drive losses.

The large storage systems that we envision do not differ only in scale from disk arrays; they are also dynamic: batches of disk drives will be deployed to replace failed or

old disks and to provide additional capacity. Our proposal for systems of this sort reduces recovery time by parallelizing the data rebuild process and increases redundancy adaptively when system load and capacity utilization allow. FARM can be characterized as follows: A RAID is *declustered* if parity data and spare space are evenly distributed throughout the disk array. Declustering is motivated by performance. FARM is declustering at a much higher scale, with a primary focus on reliability, though performance also benefits.

Figure 4.1 illustrates the principles of FARM in a small storage system, compared with a traditional RAID. For simplicity, two-way mirroring is used in this example. When disk 3 fails, FARM creates a new replica for each redundancy group that had a replica on disk 3, blocks C and E in Figure 4.1(b). Rather than creating all of the replicas on a spare disk, say disk 5 shown in Figure 4.1(c), FARM distributes all new replicas to different disks, as shown in Figure 4.1(d). In a storage system with thousands of disks, replication can proceed in parallel, reducing the window of vulnerability from the time needed to rebuild an entire disk to the time needed to create one or two replicas of a redundancy group, greatly reducing the probability of data loss.

Our data placement algorithm, RUSH [55] provides a list of locations where replicated data blocks can go. After a failure, we select the disk on which the new replica is going to reside from these locations. The selected disk is referred as the *recovery target*, and the locations of the buddies that help to rebuild the *recovery sources*. The recovery target chosen from the candidate list (a) must be alive, (b) should not contain already a buddy from the same group, and (c) must have sufficient space. Additionally, it should currently have sufficient bandwidth, though if there is no better alternative, it will still be used as the recovery target.

If S.M.A.R.T. (Self Monitoring and Reporting Technology) [58] or a similar system is used to monitor the health of disks, unreliable disks can be avoided. Unlike FARSITE [32], replicas are not moved once placed.

Even with S.M.A.R.T., the possibility that a recovery target fails during the data rebuild process remains. In this case, an alternative target will be chosen. If a recovery source fails, and there is no alternative, a data loss occurs. Otherwise, the failed source is replaced with an alternative one. The occurrence of this problem, which is called as *recovery redirection*, is rare. We found that, at worst, it happened to fewer than 8.0% of our systems even once during six simulated years.

4.1.3 Reliability Factors

The reliability of a large storage system employing FARM depends on the size and structure of redundancy groups, the size of the blocks, the latency of disk failure detection, the bandwidth utilized for recovery, the amount of data stored on disks, the number of disks in the system, and the way disk drives get replaced.

Two inherent factors affect the probability of data loss: the total number of the redundancy groups across the system and the size of a single redundancy group. Our reliability modeling (the details are discussed in Section 6.3) shows that these two factors balance each other out in the case of two-way mirroring, and that the data loss probability is independent of the redundancy group size under the idealizing assumption of zero failure detection time and independent redundancy groups.

Strategies for efficient failure detection are beyond the scope of this thesis; we

merely measure the impact of failure detection latencies, which add to the rebuild times. The speed of a rebuild depends also on the data transfer rate for the rebuild. This recovery bandwidth is not fixed in a large storage system. It fluctuates with the intensity of user requests, especially if system idle time [47] can be exploited and recovery process can be adaptive according to the workload.

Storage overhead in a large system is costly. At $1/GB, the difference between two- and three-way mirroring amounts to millions of dollars in a petabyte-scale storage system. For this reason, disk space utilization is investigated.

FARM is a general approach to combat disk failures in large storage systems. As storage systems grow, relatively rare failure types become statistically significant. FARM might not make a difference for small systems, but it is needed for large-scale systems.

Finally, the batch size—the number of new disks introduced into the system at any one time—has an effect on reliability. When new disk drives are introduced to the system, data should be migrated from old disks to the new ones. The number of the replacement processes determines the frequency of data reorganization and also affects system reliability, because of the possible failures in a batch.

4.2 Experimental Results

Since building a real petabyte-scale system is expensive, and running multi-year reliability tests is impractical at best, we ran our experiments using discrete event-driven simulations built with the PARSEC simulation tool [77].

Table 4.1: Disk failure rate per 1000 hours [35].

Period (month)	**0-3**	**3-6**	**6-12**	**12-72**
failure rate	0.5%	0.35%	0.25%	0.2%

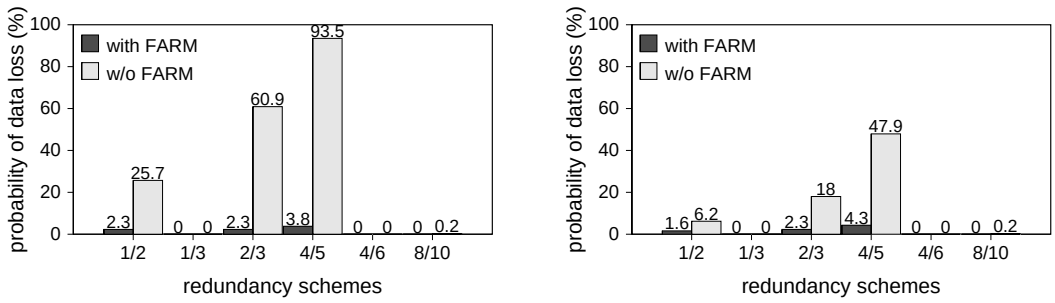
Table 4.2: Parameters for a petabyte-scale storage system.

Parameter	**Base Value**	**Examined Value**
total data in the system	2 PB	0.1–5 PB
size of a redundancy group	10 GB, 50 GB	1–100 GB
group configuration	two-way mirroring	varied
latency to failure detection	300 sec.	0–3600 sec.
disk bandwidth for recovery	16 MB/sec	8–40 MB/sec

4.2.1 System Assumptions

Our experiments explored the behavior of a two petabyte (PB) storage system, except as noted. Depending on the redundancy scheme employed, the system contains up to 15,000 disk drives, each with an extrapolated capacity of 1 TB and an extrapolated sustainable bandwidth of 80 MB/sec (based on the 56 MB/sec of the current IBM Deskstar [29]). It is assumed that recovery can use at most 20% of the available disk bandwidth, and that each device reserves no more than 40% of its total capacity at system initialization for recovered data. The size of a redundancy group is defined as the amount user data stored in it, and vary this amount from 1 GB to 100 GB.

It is well-known that disks do not fail at a constant rate; the failure rates are initially high, then decrease gradually until disks reach their End Of Design Life (EODL). The industry has proposed a new standard for disk failure distribution [35, 112]. We assume our disk drives have a typical EODL of 6 years and follow Elerath [35] for the failure rates enumerated in



(a) redundancy group size = 10 GB.

(b) redundancy group size = 50 GB.

Figure 4.2: Reliability comparisons of systems with and without FARM data protection, assuming that latency to failure detection is zero (1000 runs for each). An m/n scheme indicates that a redundancy configuration by which a redundancy group can be reconstructed from any m parity or data blocks out of the total n data and parity blocks, as discussed in Section 4.1.1.

Table 4.1.

Table 4.2 lists the default values of the system parameters together with the range of values that are used to quantify the tradeoffs in the design of our system.

4.2.2 Reliability Improvement

We first measured the improvement in reliability gained by using FARM in a large-scale storage system. We constructed redundancy groups with six types of configurations: two-way mirroring ($1/2$), three-way mirroring ($1/3$), two RAID 5 schemes ($2/3$ and $4/5$), and two ECC configurations ($4/6$ and $8/10$). We then computed the probability of data loss in a two petabyte system configured with the base parameters listed in Table 4.2, both with and without FARM data protection, assuming the latency to failure detection is zero. We further varied the size (usable capacity) of the redundancy group between 10 GB and 50 GB. The

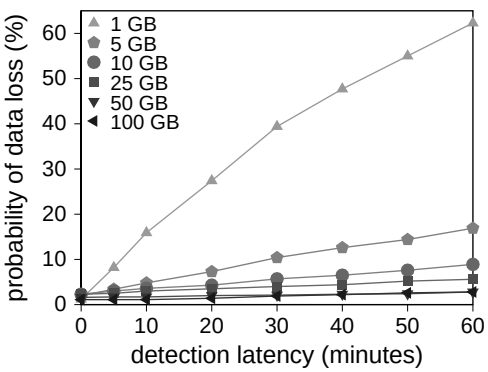
system is simulated for six years. At the end of six years, the remaining disks would be near the end of their lives and be ready to be replaced.

Our results, shown in Figure 4.2, demonstrate that FARM always increases reliability. RAID 5-like parity without FARM fails to provide sufficient reliability. With two-way mirroring, FARM reduces probability of data loss down to 1–3%, as compared to 6–25% without FARM. 3-way mirroring limits the probability of data loss to less than 0.1% during the first six years, similar to the probability of $4/6$ and $8/10$ with FARM.

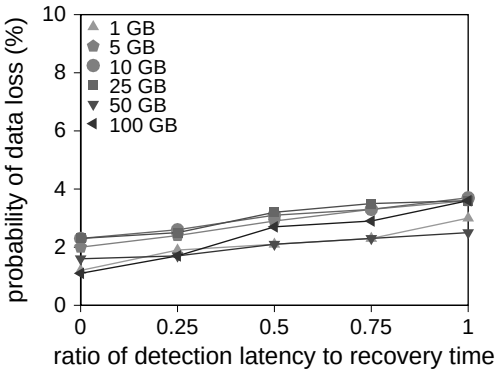
Figures 4.2(a) and 4.2(b) show that the size of redundancy groups has little impact on systems using FARM, but *does* matter for systems without FARM. According to our theoretical analysis based on modeling (in Section 6.3), data loss probability is independent of group size under two-way mirroring with FARM, if the latency of failure detection is zero. Without FARM, reconstruction requests queue up at the single recovery target. Data loss occurs if any of the recovery sources or their alternatives fail before the block is reconstructed. With smaller redundancy groups, there are more recovery sources that can fail during that time, so that the probability of data loss increases.

4.2.3 Latency of Failure Detection

After a disk fails, the failed disk has to be identified first and then the data on the failed disk can get reconstructed. The window of vulnerability consists of the time to detect a failure and the time to rebuild the data. Discovering failure in such a large system is not trivial and the latency to failure detection cannot be neglected. As the size of redundancy groups varies from 1 GB to 100 GB under two-way mirroring plus FARM, a system with smaller



(a) The effect of detection latency on the probability of data loss.



(b) The effect of the ratio of detection latency to recovery time on the probability of data loss.

Figure 4.3: The effect of latency to detect disk failures on overall system reliability under various redundancy group sizes.

group sizes is more sensitive (Figure 4.3(a)) to the detection latency. It takes less time to reconstruct smaller-sized groups, so a constant failure detection latency makes up a much larger relative portion of the window of vulnerability. For example, it takes 64 seconds to reconstruct a 1 GB group, assuming reconstruction runs at a bandwidth of 16 MB/sec, while it takes 6400 seconds for a 100 GB group. If it takes 10 minutes to detect a failure, detection latency represents 90.4% of the window of vulnerability for the former, and only 0.86% for the latter. We hypothesized that the *ratio* of failure detection latency to actual data recovery time determines the probability of data loss. Our results, summarized in Figure 4.3(b), show that this is the case.

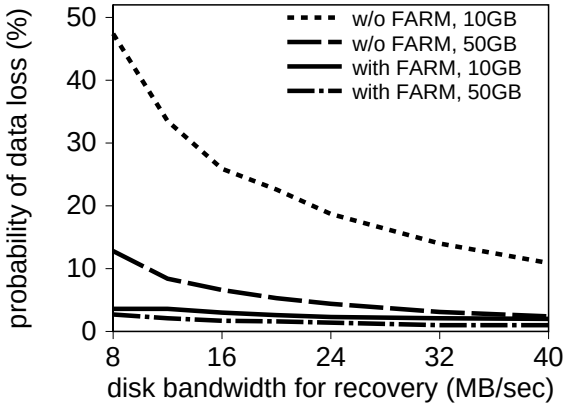


Figure 4.4: System reliability at various levels of recovery bandwidth with size of redundancy groups varies as 10 GB and 50 GB, under FARM and traditional recovery scheme, respectively.

4.2.4 Disk Bandwidth Usage for Recovery

Data rebuild time can be shortened by allocating a higher portion of device bandwidth to recovery. To gauge the impact of recovery on usable bandwidth, various disk bandwidths contributed to data recovery are examined in a 2 PB storage system with two-way mirroring under the assumption that failure detection latency is 300 seconds.

As expected, the probability of data loss decreases as the recovery bandwidth increases (Figure 4.4). In all cases, we observed that the chance of data loss is higher for a smaller group size, due to the impact of failure detection latency. High recovery bandwidth improves system reliability dramatically for the systems without FARM, but does not have a pronounced effect when FARM is used. The advantage of higher recovery bandwidth is to reduce the data rebuild time, but FARM has already cut recovery time dramatically, so that further reductions from higher bandwidth utilization achieve little improvement. Without FARM, recovery time is quite long due to the single recovery target, so high recovery bandwidth can

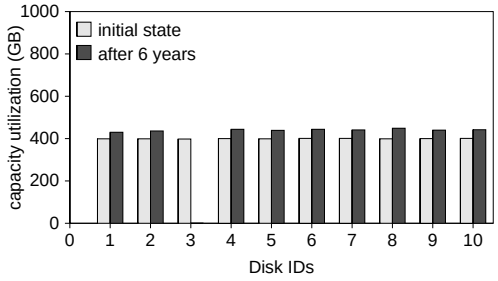
statistical values	1 GB		10 GB		50 GB	
	initial state	six years later	initial state	six years later	initial state	six years later
mean	400 GB	442.33 GB	400 GB	442.33 GB	400 GB	442.33 GB
standard deviation	1.41 GB	6.44 GB	18.03 GB	26.41 GB	81.52 GB	92.53 GB

Table 4.3: Mean and standard deviation of disk utilization in the system initial state and after the end of disk lifetime (six years). Redundancy groups are configured as 1 GB, 10 GB and 50 GB, respectively.

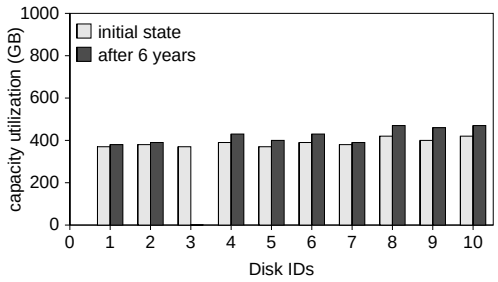
greatly improve reliability. In systems where disks are much less reliable, or storage capacity exceeds petabytes, high recovery bandwidth can be effective even when FARM is used.

4.2.5 Disk Space Utilization

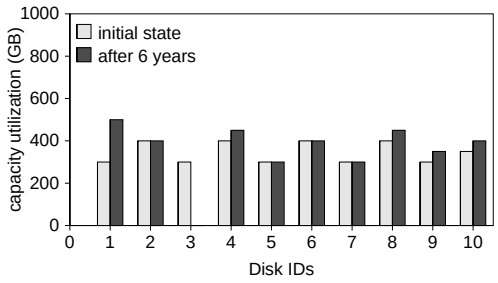
FARM distributes both data and redundancy information across the whole system. As disks fail, data stored on them is redistributed to the rest of the system; it is never recollected to a single disk. This approach has the potential to introduce imbalances in the actual amount of data stored on each individual drive. However, our technique does not suffer from this problem, as demonstrated by an experiment summarized in Figure 4.5. Data is distributed using the placement algorithm on 10,000 1 TB disks with an average utilization of 40%, including both primary and mirror copies of data. Disk failures and data reconstruction are then simulated for six years. The mean and standard deviation of capacity utilization are listed in Table 4.3. Smaller-sized redundancy groups result in a lower standard deviation on capacity, although the mean values stay the same. Figure 4.5 shows the load for ten randomly-chosen disk drives both before and after the six years of service. Disk 3 failed during the service period so it does not carry any load. The other nine disk drives have increased their disk space usage due to the distributed redundancy created by FARM. The unevenness in data distribution



(a) redundancy group size = 1 GB.

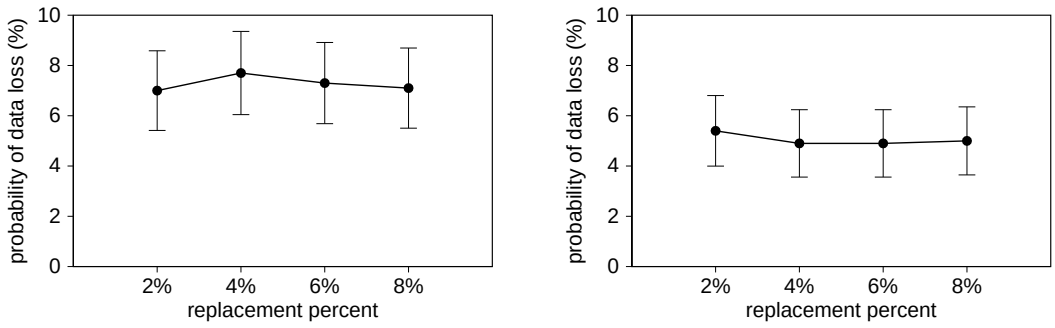


(b) redundancy group size = 10 GB.



(c) redundancy group size = 50 GB.

Figure 4.5: Disk utilization for ten randomly selected disks. Utilization was measured at the start of the simulation period and at the end of the six year period. The size of redundancy groups is varied as 1 GB, 10 GB and 50 GB.



(a) redundancy group size = 10 GB

(b) redundancy group size = 50 GB

Figure 4.6: Effect of disk drive replacement timing on system reliability, with 95% confidence intervals. New disks are added in the system after losing 2%, 4%, 6%, and 8% of the total disks.

is caused by the relatively large ratio of redundancy group size to disk size. Reducing redundancy group size to 1 GB would alleviate this problem and balance disk utilization better, but at the cost of lower reliability, as described in Section 4.2.2.

4.2.6 Disk Drive Replacement

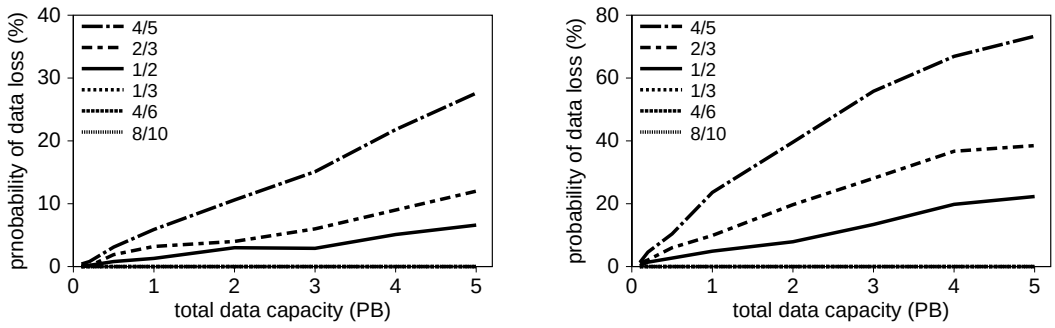
Large-scale storage systems are always dynamic: old disk drives are removed when they fail or retire, and new disk drives are added to satisfy the demands of disk replacement and data capacity growth. It is typically infeasible to add disk drives one by one into large storage systems because doing so would require daily (if not more frequent) drive replacement. Instead, a cluster of disk drives, called a *batch*, is added. The choice of batch size determines the replacement frequency and the amount of data that needs to migrate onto new drives in order to keep the system in balance.

The newly-added disks come from different disk vintages and have various storage capacities. The reorganization of data should be based on the *weight* of disks, determined by disk vintage, reliability properties, and capacity. The determination of these weights is not within the scope of this thesis; currently, the weight of each disk is set to that of the existing drives for simplicity.

Large batch sizes can have a negative impact on system reliability because they introduce a large number of new, and hence more failure-prone, disks into the system. This effect is called as the *cohort effect*. It is experimented by replacing failed disk drives once the system has lost 2%, 4%, 6%, and 8% of the total number of disk drives. As Figure 4.2.6 reports, the cohort effect is not visible using a redundancy group size of 10 GB, in large part because only about 10% of the disks fail during the first six years. As a result, disk replacement happens about five times at the batch size of 2% and about once at 8%, assuming that total data capacity remains unchanged. The number of disks in each replacement batch is 200 for 2% and 800 for 8%, so that only 2% and 8% of the data objects migrate to newly-added disks. This number is too small for the cohort effect, so batch size and replacement frequency does not significantly affect system reliability. Thus, there is little benefit beyond delaying some cost to just-in-time replacement.

4.2.7 System Scale

Our findings apply not only to a two-petabyte system, but also to any large-scale storage system. As expected, the probability of data loss, shown in Figure 4.7(a), tends to increase approximately at a linear rate as system scales from 0.1 PB to 5 PB. For a 5 PB storage



(a) Disks with the failure rate listed in Table 4.1.

(b) Disks with a failure rate twice that listed in Table 4.1.

Figure 4.7: The probability of data loss in a storage system under FARM is approximately linear in the size of the storage system. The size of redundancy groups is 10 GB.

system, FARM plus two-way mirroring achieves a data loss probability as low as 6.6%. However, RAID 5-like parity cannot provide enough reliability even with FARM. Using a 3-way mirroring, 6 out of 8, or 8 out of 10 scheme with FARM, the probability of data loss is less than 0.1%. This result is not unexpected—it is well-known that a system with twice as many disks is approximately twice as likely to fail given otherwise identical parameters.

Disk vintage [35] is an important aspect in system reliability. Disk drives with various vintages differ in failure rates and even failure modes. Disk drives are set up with failure rates as twice high as the disk vintage listed in Table 4.1 and vary the system scale. A similar trend in increase of data loss probability is observed, as shown in Figure 4.7(b). As the failure rate of individual drives doubled and the rest of the configuration stayed the same, the probability of data loss more than doubled. This is due to several factors, including an increased likelihood of failure when one or both disks are new. Keeping disk failure rates low is a critical

factor in ensuring overall system reliability because system reliability decreases at a rate faster than individual disk reliability decreases.

4.3 Summary

Traditional redundancy schemes can not guarantee sufficient reliability for petabyte-scale storage systems. Simply increasing the number of replicas is cumbersome and costly for read/write systems, and using *m* out of *n* schemes, while cheaper, is also more complex. In response, FARM is developed, a fast recovery scheme that distributes redundancy on demand effectively to improve reliability. Data objects are clustered into redundancy groups that may use various replication and erasure-coding schemes. FARM provides much higher reliability for large-scale storage systems. Our results show that failure detection latency affects reliability greatly for small redundancy groups, and that the ratio of recovery time to failure detection latency is crucial. Our results show that allocating a higher portion of bandwidth to recovery does not have a pronounced influence for FARM because data rebuild times have already been greatly reduced by FARM. FARM has also been validated for general large-scale storage systems of different sizes.

Our work offers a more effective redundancy mechanism for designers of large-scale storage systems, especially those that must provide high data reliability. This chapter discusses a family of redundancy techniques for large-scale storage systems and presents the quantifying analysis of different system parameters on the reliability of such systems. Our work provides designers of petabyte-scale systems the techniques they need to build the high-performance,

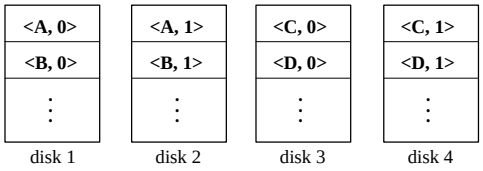
high-capacity systems demanded by scientific computation and “computing utilities.” By applying these techniques, designers can ensure that their systems feature high reliability as well as high performance.

Chapter 5

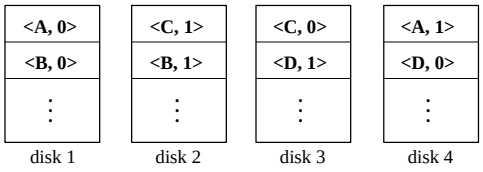
Data Layout Schemes

System reliability not only depends on recovery mechanisms, but also is directly related to data layout schemes. This chapter examines the effect of different schemes of data layout on reliability of large-scale storage systems and discusses the merits and limitations of each scheme.

Data layout is subjected to change over time in a system because of the additions of new drives, removal of failed or old drives, and data migrations. There are two critical states of data layout: one is the initial data placement, and the other one is data placement during data recovery. These two are related with each other in that the initial data layout can have an effect on where data will be placed after a disk failure. In order to decide how to place data initially in a system, one needs to consider several system factors, including system scale, I/O workload, performance, and reliability requirement. After data recovery, data placement is largely dependent on recovery schemes, which has been analyzed in details in the previous chapter. In this chapter, three initial data placement schemes are compared, including (1) one



(a) placement without data declustering



(b) placement with data declustering

Figure 5.1: Data layout comparison: without declustering and with declustering.

layer data placement with declustering, (2) one layer data placement without declustering, and (3) hierarchical data placement, and the impact of each data layout on overall system reliability is discussed.

5.1 Data Declustering

In the OSD system, tens of thousands of clients access files from the petabyte-scale system consisting of thousands of object-based disks. I/O requests arrive at intensive rates and the request have large sizes, as shown in Section 3.2.1. The technique of declustering was developed in order to improve system performance and reduce the time required in constructing the data on a failed disk in a RAID system online.

5.1.1 The Degree of Data Declustering

Data allocation incorporates two key parameters: (1) determination of the degree of data declustering and (2) mapping of the redundant fragments (declustered data) to disks in the system.

The degree of data declustering is defined as the average number of redundancy groups on each disk. A high degree of data declustering favors intra-transaction parallelism and load balancing during data recovery, but at the cost of higher overhead of communication among disks involved during failure recovery; on the other hand, a low degree of data declustering can reduce communication overhead, but at the expense of intensive workload on selective disks and longer recovery time during data reconstruction process. The selection of an appropriate degree of data declustering remains rather empirical because of high complexity of large-scale storage systems, therefore, a compromise respect to several contradicting goals can be reached generally based on the application and customer requirements.

Data and its redundancy are placed in a highly distributed fashion to enable maximum bandwidth utilization and reduce the vulnerability of single points of failure. The RUSH [55] algorithm is used to determine the location of data and its associated redundancy. RUSH provides good random allocation of objects, accommodates replication and more advanced data redundancy configurations, and preserves load balance in the face of system growth or drive failures. The initial data placement uses declustering to spread the redundancy groups across the entire system, rather than limiting data to be placed in a small number of disks. Figure 5.1 shows a simple example of a system with four disks. Four redundancy

groups — *A*, *B*, *C*, and *D* — are configured with two-way mirroring scheme. A simple and widely used placement method is disk to disk mirroring, by which all the data on one disk is replicated to another one and the data on both disks is identical, as shown in Figure 5.1(a). As a comparison, the data layout with declustering, shown in Figure 5.1(b), spreads the replicas or parities of the redundancy groups on one disk to multiple other disks, resulting in a completely distributed layout.

FARM with data declustering shortens the time of recovery upon a disk failure, although multiple disk drives are exposed in the window of data loss vulnerability. For example, suppose we use two-way mirroring redundancy and there is 400 GB data stored in a failed disk with 50 GB data in each redundancy group, then eight (400/50) disks store the replicas of the data segments on the failed disk. During data recovery process, these eight disks are the recovery sources and another eight disks will serve as recovery targets which home the recovered contents. Altogether sixteen disks are involved in the recovery process: if any of the eight source disks fails during this process, data loss occurs; the failure of recovery targets will not result in data loss: recovery will be redirected to another disk if any of the recovery targets fail. Comparing the recovery process in a system that data is purely mirrored on two disks without declustering or FARM, once a disk fails, only two disks — its mirrored drive and a spare disk — are involved in the recovery process; thus, the recovery process is much longer than that using FARM with declustering. Two contradicting factors, the number of disks exposed in the window of vulnerability and the length of the window (the recovery time), are related to each other. FARM with declustering shortens the length of the window but increases the number of disks exposed during the window of vulnerability; pure mirroring without declustering limits

the number of disks vulnerable in the window but suffers from a long period of recovery time. These two factors balance out each other and result in an equal reliability level in these two methods: FARM with declustering and pure mirroring without declustering.

The number of redundancy groups on one disk drive is a configurable parameter. In the design of FAB — distributed enterprise disk arrays from commodity components [97], the choice of the number of "seggroups", which are similar to the redundancy groups in our system, was discussed. It was indicated that the choice of the number of seggroups per disk "reveals a tension between load balancing and reliability": increasing the number of redundancy groups reduces the system's reliability since it raises the number of combinations of disk failures that can lead to data loss; while decreasing the number of redundancy groups yields poor load balancing. The number of redundancy groups was picked as four per brick (the storage component in the FAB system). A typical value of the number of redundancy group is eight in the OSD system, and the value can be configured to meet the specific need of a system. The analysis of reliability in FAB is based on a system with about 100 bricks and the model does not consider different methods of data recovery. Comparatively, in our OSD system consisted of thousands of object-based disks, FARM is used together with data declustering to shorten the repair time, thus mitigating the effect of the increased combinations of disk failures that can lead to data loss.

5.1.2 Effect of Sparing Space

Except the degree of data declustering, there is another important factor which favors FARM on system reliability, which is distributed sparing space. FARM utilizes the spare space

among the disk drives already being used in the system, rather than bringing up a new disk as a replacement in the pure mirroring scheme. Compared with the disks already being used in the system, a newly added disk is usually less reliable due to the effect of infant mortality. One can always use burnt-in disks as new replacement but that induces extra cost and increases the complexity of storage management since the burnt-in disks can fail and need to be replaced as well. In addition, to replace a dead disk with a new one takes time. This period of time, called "replacement latency", directly affects system reliability. This latency lasts from a few hours to even several days. The longer the replacement latency, the wider the vulnerability window, and the higher likelihood of data loss.

Figure 5.2 shows the simulated results of data loss probability in the time span of six years under different disk replacement latencies in a system using pure mirroring data placement without declustering. It is assumed that failure detection latency is 300 seconds and disk failures follow the bathtub-curve model as described in Section 4.2.1. It has been observed that replacement latency has a pronounced effect on data loss probability: the probability of data loss increases linearly along with the increase of replacement latency (the equation of linear fit is $y = 0.2658x + 2.1687$ with R-square value as 0.9997, where x is the replacement latency in hours and y is the percentage of the data loss probability in six years). For example, when it takes 24 hours to replace a failed disk, data loss probability is as high as 8.45%. Fortunately, the replacement latency is a non-factor for FARM. the data loss probability in six years is only 1.6% using FARM with highly distributed spare space, as shown in Figure 4.2(b). In a system using FARM, disks are replaced in batches gradually as regular maintenance. As a result, the latency of disk replacement does not directly widen the vulnerable window of data

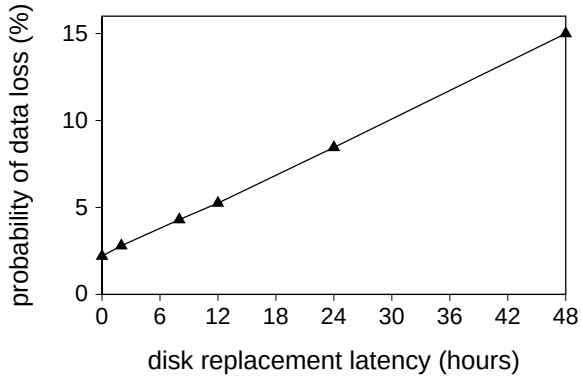


Figure 5.2: Probability of Data Loss under Varied Disk Replacement Latencies in Six Years

loss and is irrelevant to data loss.

From the point of view on performance, FARM takes the advantage of bandwidth utilization of multiple disks and results in high performance under the degraded mode over pure mirroring without declustering. Comparatively, under pure mirroring, the disks dedicated for recovery will be under the degraded mode for a relatively long time (up to days).

5.2 Multiple-level Data Redundancy

In light of advances of storage technologies, it is intuitive to consider achieving higher system reliability by using multiple layers of data redundancy.

5.2.1 Hierarchical Disk Array

Data declustering with fast recovery mechanism improves reliability by utilizing I/O bandwidth from multiple disk drives to shorten recovery time. Alternatively, we can enforce higher redundancy by placing additional level(s) of redundancy on the top of a RAID-based

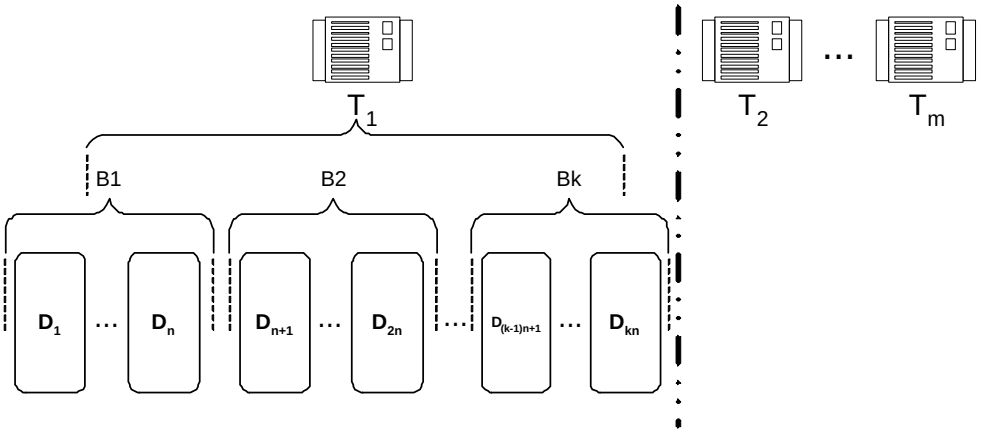


Figure 5.3: Two-level Hierarchical Disk Array.

system. Hierarchical arrays with more than two levels are too complex to be exploited in practice, so that we only consider two-level arrays. An example of a two-level hierarchical array system is depicted in Figure 5.3. We refer the two levels as the "top" and the "bottom" level respectively. At the bottom level, disks are configured in groups, with each group composed of n disks. These groups, labeled as B_1, B_2, etc. , are the components of the top-level array, from T_1 to T_m, at the top level. Note that the redundancy configurations at the bottom and top level can be different.

We simulated the two-level disk array systems and found that they can provide very high reliability. This is not unexpected: the structure of the hierarchical disk arrays provides superior redundancy at the expense of the complexity and cost.

Five kinds of configurations of two-level array systems are examined by simulations. The configurations are presented as "$\{\mathrm{A}\} \times \{\mathrm{B}\}$" where A is the configuration for the top level, and B is the configuration for the bottom level: (1) $\{$two-way mirroring$\} \times \{$two-way

Table 5.1: Comparison of Data Loss Probability of Hierarchical Array and FARM

storage efficiency	k-availability	FARM	HR-RAID
25%	3	$< 0.05\%$	$< 0.05\%$
40%	3	$< 0.05\%$	$< 0.05\%$
60%	2	$< 0.05\%$	$< 0.05\%$
80%	2	$< 0.05\%$	$< 0.05\%$
90%	2	$< 0.05\%$	0.1%

mirroring}; (2) {two-way mirroring} $\times$ {4+1 RAID-5}; (3) {4+1 RAID-5} $\times$ {4+1 RAID-5}; (4) {9+1 RAID-5}$\times$ {9+1 RAID-5}; and (5) {19+1 RAID-5} $\times$ {19+1 RAID-5}. The data loss probability over six years of simulated time in a two-petabyte storage system is evaluated and 2,000 data samples are collected for each configuration. There was no data loss occurrence among our simulated results under (1), (2), (3) and (4) configurations, and only two data loss occurrences (0.1%) under the (5) configuration. Our results indicate that data loss is very rare in two-level hierarchical array systems. It is assumed that the latency to replace a disk is 24 hours and the latency to replace a bottom-level array is 72 hours. In real systems, the latency may be even longer, especially for the process of replacing an array composed of multiple disks with complex redundancy configurations. As discussed in Section 5.1, long replacement latency hurts system reliability.

To compare reliability provided by hierarchical arrays and FARM, several one-level data redundancy configurations with FARM are simulated. A fair comparison of these two kinds of schemes is based on the same level of storage efficiency (which is calculated as the ratio of the amount of user data to the total amount of storage used) and the same number of *k* availability (*k* is the number of failures that can be tolerated at the same time). Accord-

ingly, five kinds of redundancy schemes with the use of FARM are simulated: (1) four-way mirroring; (2) 2 out of 5 erasure coding; (3) 3 out of 5 erasure coding; (4) 8 out of 10 erasure coding; and (5) 18 out of 20 erasure coding. These five kinds of redundancy schemes result in 25%, 40%, 60%, 80%, and 90% storage efficiency respectively, which is comparable to those obtained by these five configurations of hierarchical disk arrays, which are 25%, 40%, 64%, 81%, and 90.25%. Also, these five configurations result in the same number of k availability as the five configurations of two-level arrays. The data loss probabilities over six years by using these different kinds of redundancy schemes are simulated and compared. Table 5.1 lists the simulation results: both one-layer data declustering with FARM and two-level hierarchical data layout provide high reliability at the same level of data redundancy. The data loss occurrence is less than 0.05% for both schemes when storage efficiency is at the level of 25%, 40%, 60%, and 80%. Note that 2000 data samples are collected for each configuration, resulting in $\frac{1}{2000}$ accuracy. When less redundancy is used and storage efficiency is 90%, the data loss probability is 0.1% for two-level array while it is below 0.05% for FARM with 18 out of 20 erasure coding configuration. The results indicate that FARM can provide system reliability as high as the hierarchical arrays, and FARM does not suffer from disk replacement latency as hierarchical arrays do.

5.2.2 Discussions

Both one-level data redundancy with FARM (referred as “FARM”) and two-level data redundancy schemes (referred as “HRRAID”) can improve system reliability. The performance and cost issues of these two schemes are further discussed below.

For read requests, when the data is not in the cache, in a system using FARM, clients will calculate the data locations by the RUSH algorithm and access these OSDs in parallel directly. Comparatively, in a system using HRRAID, the requests are first mapped to the top level of the array and then the RAID controllers perform data address mapping and schedule I/O accesses to their underlying disks. The overhead of two-level data mapping leads to poor system performance, even though intelligent scheduling schemes by RAID controllers can improve performance.

For write requests, besides user data, redundant data also needs to be updated. For *n*-way mirroring, the two systems have similar performance in that a single write request from a client leads to *n* writes. However, when parity-based redundancy is deployed, HRRAID systems have to calculate and write the parity information for both levels, while FARM only does it for one level. The performance of small writes on HRRAID is expected to degrade more than that on system using FARM due to the dual layers of parity calculation and update. In the presence of drive failures, FARM distributes the reconstruction task to multiple disks and reduces the degraded period of each disk involved in the recovery process. FARM only utilizes a small fraction of disk bandwidth (typically 20%), thus allowing these disks to serve their normal I/O requests concurrently. HRRAID dedicates reconstruction to a single spare disk, which makes the disks involved in the recovery process remain in the degraded mode for a long period time. During that time, those disks may not be able to serve normal I/O requests when recovery takes a large portion of disk bandwidth or the disks have to serve under the degraded mode for a long time when recovery is processed at a low rate. In terms of the overhead from coordinating disks during recovery process, FARM poses higher overhead

since a larger number of disks have to be managed, while HRRAID delegates some of the management jobs to the RAID controllers at the bottom level of the hierarchical array.

The performance in a real system depends heavily on other factors besides data layout, such as workload, scheduling policy, and the underlying hardware components and the associated software. There is no general claim that one system can outperform another in any case.

The scheme of hierarchical disk arrays can greatly improve system reliability, but at the cost of high complexity. There are two general ways to implement a hierarchical array: via software-based array management and via RAID cards. Software-based management is relatively cheap in cost as it does not need extra hardware, but yields rather poor performance due to the computational complexity related to the dual array layers. Comparatively, using disk array cards results in better performance, but it introduces extra components which can also fail and may result in more serious failures in a system.

In practice, hierarchical arrays are rarely used due to their high cost and complexity. It is especially difficult to make the storage management at different levels working gracefully together. In terms of cost, hierarchical arrays require array controllers sitting between different levels, which greatly increases the budget of hardware and management softwares for each level, without regard to the high cost of multiple-level system maintenance.

The two schemes — FARM and HRRAID can be used in a complementary manner to get the best out of these two: to ensure high reliability and maintain high performance under degraded mode and to reduce the overhead of communicating. An OSD device can be composed of five disks with 4+1 RAID-like configuration, and another layer of two-way

mirroring is added among OSDs. FARM and data declustering are employed on top of OSDs. In this way, each OSD handless management and scheduling of the underlying disk drives such that the communication overhead among disks can be reduced, while data reconstruction process is spread to multiple OSDs by FARM to achieve efficient load balancing and high performance during degraded mode.

5.3 Summary

The schemes of data layout and their impact on system reliability are discussed in this chapter. The technique of data declustering and its related issues, including the degree of data declustering, and the utilization of sparing space, have been investigated. A hierarchical data placement scheme has been studied and compared with FARM and data declustering in the aspects of reliability, performance, and complexity. It has been shown that each scheme has its own advantages and limitations, and the two schemes can be used coherently to improve system performance and reliability.

Chapter 6

Modeling for Reliability of Large-Scale Storage Systems

> "That which today calls itself science gives us more and more information, and indigestible glut of information, and less and less understanding."
>
> —- Edward Abbey (1927-1989)

Our research focuses on understanding and coping with disk failures in large-scale storage systems. Reliability of disk drives, marked as "one of the trickiest drive characteristics" [4], is related to many factors and is difficult to measure. Understanding the properties and the impacts of disk defects would provide us the basis of reliability modeling.

6.1 Disk Drive Failure Taxonomy

Storage systems must deal with defects and failures in rotating magnetic media. There are three principal failure modes that affect disks: block failures, also known as media

failures; catastrophic device failures; and data corruption or unnoticed read errors.

On a modern disk drive, each block consisting of one or more 512 Byte sectors contains error correcting code (ECC) information that is used to correct errors during a read. If there are multiple errors within a block, the ECC can either successfully correct them, flag the read as unsuccessful, or in the worst case, miscorrect them. The latter is an instance of data corruption, which fortunately is extremely rare. If the ECC is unable to correct the error, a modern disk retries the read several times. If a retry is successful, the error is considered a soft error; otherwise, the error is hard. Failures to read data may reflect physical damage to the disk or result from faulty disk operations. For example if a disk suffers a significant physical shock during a write operation, the head might swing off the track and thus destroy data in a number of sectors adjacent to the one being written.

Device failure rates are specified by disk drive manufacturers as MTTF (Mean-Time-To-Failure) values. The disk manufacturers typically claim that MTTF of a disk drive is one million hours (114 years), however, the actual observed values depend in practice heavily on operating conditions that are frequently worse than the manufacturers' implicit assumptions. Block failure rates are published as failures per number of bits read. A typical value of 1 in 10^{14} bits for a commodity ATA drive means that if we access data *under normal circumstances* at a rate of 10 TB per year, we should expect one uncorrectable error per year.

This chapter focuses on modeling of catastrophic device failures and estimation of system reliability. The block failures are not considered in the disk failure models since the rate of block failures heavily depends on the specific workload. A related work on disk scrubbing [104] has discussed the schemes to combat against block failures.

6.2 Disk Failure Modeling

Disk failures are modeled by continuous-time Markov chains [116]: state transitions between disk failures and recoveries can take place at any instant of time with certain probability distribution patterns. Particularly, infant mortality is considered in reliability models of disk drives.

6.2.1 Mean Time To Failure of Disk Drives

The Mean-Time-To-Failure (MTTF) is a widely used metric of drive reliability. It provides a general idea of the expected lifetime of a disk drive, however, it assumes that the distribution of disk failure rates be exponential which is not accurate for disk drives at their first-year lifetime [35]. Disk MTTF is still good for the measurement of Mean-Time-To-Failure or Mean-Time-To-Data-Loss of a storage system consisting of multiple disk drives, although it cannot give any details on the distribution of system failures.

6.2.2 Disk Infant Mortality

A disk is a complicated device that consists of many electronic, magnetic, and mechanical components. Many manufacturing errors are latent and only lead to failure later in the lifespan of the product. For this reason, disk manufacturers subject new disks to a testing regimen called *burn-in*. Burn-in periods for more expensive and higher-performing SCSI drives are longer, but are much more limited for commodity IDE drives. Even so, some SCSI disks are "dead on arrival." A true disk failure model would faithfully reflect the possible states of each disk component after production and calculate the lifespan of a disk by finding the first

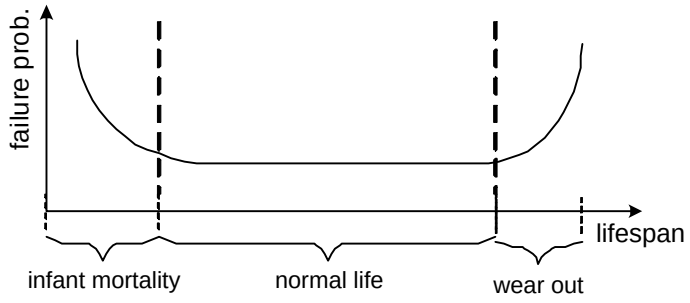


Figure 6.1: Disk failure rate is plotted as a bathtub curve: higher failure rate at the beginning (infant mortality) and the end (wear-out) than the middle (normal life) of disk lifespan.

combination of conditions that lead to death, but a disk's complexity renders this approach impractical. Instead, a phenomenological approach is taken, which is, to model the failure rates of a disk, but does not explain them.

It is well-known in the disk drive industry that disk drives do not fail at a constant rate. Rather, failure rates are initially high, a problem known as *infant mortality*, but then diminish rapidly to level out about one year into a disk's lifespan, only to rise again after the disk becomes economically obsolete at an age of 5–6 years (for current drives). This failure profile is known as a "bathtub curve." A plot of the shape of the curve is shown in Figure 6.1.

6.3 Reliability Estimation: Mean-Time-To-Data Loss Calculation of a Storage System

Mean-Time-To-Data-Loss (MTTDL) of a petabyte-scale storage system is first calculated under varied configurations and the distribution of data loss in the whole system is analyzed.

Table 6.1: Parameters for a 2 PB storage system.

Parameter	**Value**
Z (total data in the system)	2 PB
γ (recovery rate)	10^2 GB/hr
N (number of redundancy groups)	Z/S
$MTTF_{disk}$	10^5 hours
D (disks in an OSD RAID 5)	5
S (data in one redundancy group)	varies

6.3.1 MTTDL for Various Redundancy Mechanisms

Using Markov models, we compared MTTDL in a 2 PB storage system by using Mirror 2, Mirror 3, and RAID 5+1 redundancy configurations. The storage system parameters used in our comparison are listed in Table 6.1. μ is the failure rate and ν is the repair rate in figures of state transitions and the calculations. Here, $\nu \gg \mu$, meaning that mean time to repair a disk is much shorter than mean time between failures.

Figure 6.2(a) shows state transitions for one redundancy group using the Mirror 2 mechanism, where μ is the failure rate and ν is the repair rate. State 0 indicates that both OSDs that contain a copy of the objects in the redundancy group are functioning properly; if either one of them is down, then it goes to state 1. The model goes to state 2 only when the second OSD is down while the earlier failed one has not yet been rebuilt.

Assuming that there is at most only one failure or repair happening during a very short time interval Δt, we find that *MTTDL* of one redundancy group is

$$\begin{aligned} MTTDL_{RG-Mirror\,2} &= \frac{3\mu+\nu}{2\mu^2} \\ &= \frac{\nu}{2\mu^2} + \Delta_{RG-Mirror\,2} \end{aligned} \tag{6.1}$$

The relative error is

$$\frac{\Delta_{RG-Mirror\,2}}{MTTDL_{RG-Mirror\,2}} = \frac{3\mu}{3\mu+\nu} \approx \frac{3\mu}{\nu} \tag{6.2}$$

Since $\frac{\mu}{\nu}$ is very small (10^{-7} when the size of a redundancy group is 1 GB), the approximate *MTTDL* for one redundancy group under Mirror 2 is

$$MTTDL_{RG-Mirror\,2} = \frac{\nu}{2\mu^2}\,. \tag{6.3}$$

We then derived the *MTTDL* of one redundancy group when the Mirror 3 mechanism is used from the Markov model shown in Figure 6.2(b) in a similar way:

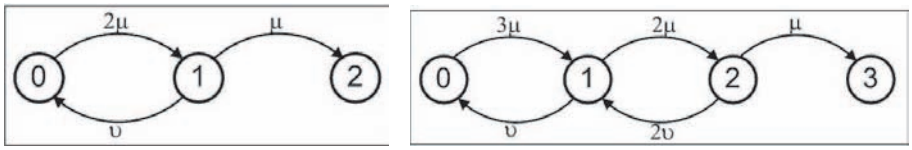
$$MTTDL_{RG-Mirror\,3} = \frac{2\nu^2+7\mu\nu+11\mu^2}{6\mu^3} = \frac{\nu^2}{3\mu^3} + \Delta_{RG-Mirror\,3} \tag{6.4}$$

$$\frac{\Delta_{RG-Mirror\,3}}{MTTDL_{RG-Mirror\,3}} = \frac{7\mu\nu+11\mu^2}{2\nu^2+7\mu\nu+11\mu^2} \approx \frac{7\mu}{2\nu}\,. \tag{6.5}$$

The approximate *MTTDL* for Mirror 3 and the relative error are shown in Equations 6.4 and 6.5. From these equations, it can be seen that the approximate *MTTDL* for a redundancy group in Mirror 3 is

$$MTTDL_{RG-Mirror\,3} = \frac{\nu^2}{3\mu^3}\,. \tag{6.6}$$

The Markov model of one redundancy group under RAID 5+1 is shown in Figure 6.3(b). In this state diagram, D is the total number of disks in one RAID 5 and $\langle x,y,z\rangle$



(a) State Transitions for Mirror 2 (b) State Transitions for Mirror 3

Figure 6.2: Markov Model for two-way and three-way Mirroring.

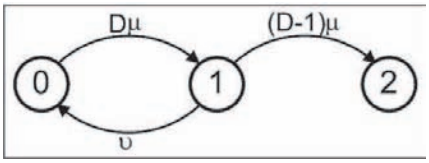
indicates that there are *x* pairs of OSDs in which both of them are in operation, *y* pairs of OSDs in which one of the them are in operation and the other one is down, and *z* pairs of OSDs in which neither of the two OSDs are working. Here, the two mirrored OSDs are referred as a *pair*.

State transitions with the RAID 5+1 mechanism are more complicated than those for Mirror 2 and Mirror 3 since the model goes to the failure state only when two OSDs in RAID 5 fail and the two corresponding mirroring OSDs in another RAID 5 fail at the same time. The Markov model is simplified by first deriving the failure rate and repair rate for a pair of OSDs. The failure rate of a mirrored pair of OSD is then substituted into the Markov model for a single RAID depicted in Figure 6.3(a). The derivation is as follows:

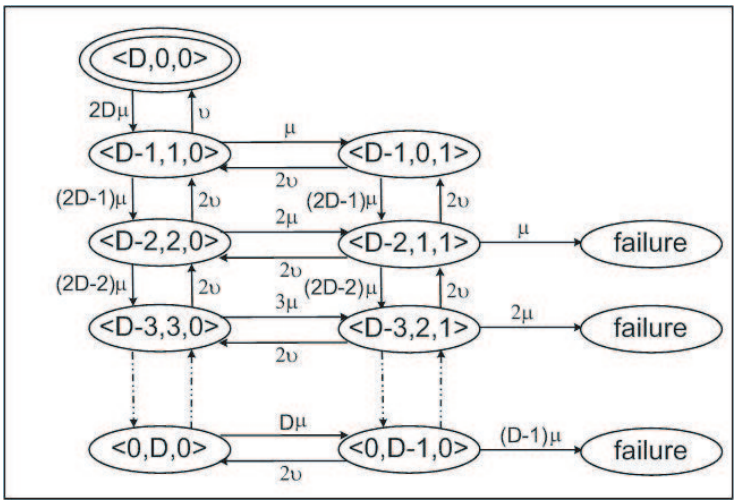
$$MTTDL_{RG-RAID5} = \frac{(2D-1)\cdot \mu_{RAID5} + \nu_{RAID5}}{D\cdot(D-1)\cdot {\mu_{RAID5}}^2}. \tag{6.7}$$

Using Equation 6.3, we have

$$\mu_{RAID5} = \frac{2\mu^2}{3\mu+\nu} \quad \text{and} \quad \nu_{RAID5} = \nu\ . \tag{6.8}$$



(a) State transitions for RAID 5



(b) State transitions for RAID 5+1

Figure 6.3: Markov models for RAID 5+1

Thus, we find the *MTTDL* of one redundancy group under RAID 5+1 is

$$MTTDL_{RG-RAID5+1} = \frac{(3\mu\nu + \nu^2 + (2D-1)\cdot 2\mu^2)\cdot(3\mu+\nu)}{D\cdot(D-1)\cdot 4\mu^4}$$
$$= \frac{\nu^3}{D\cdot(D-1)\cdot 4\mu^4} + \Delta_{RG-RAID5+1}\ . \tag{6.9}$$

The relative error is

$$\frac{\Delta_{RG-RAID5+1}}{MTTDL_{RG-RAID5+1}} \approx \frac{6\mu}{\nu}\ . \tag{6.10}$$

The *MTTF* for RAID 5+1 is approximately

$$MTTDL_{RG-RAID5+1} \approx \frac{\nu^3}{4D\cdot(D-1)\mu^4}\ . \tag{6.11}$$

To compare the MTTDL of the three redundancy schemes, the following equations which approximate (to within 1%) the MTTDL for each of the redundancy mechanisms are used. *MTTF* for one redundancy group is just the *MTTF* of a disk, ($MTTF_{disk}$), so we have

$$MTTF_{disk} = \frac{1}{\mu} \tag{6.12}$$

and *MTTR* (Mean Time To Repair) for a single redundancy group is:

$$MTTR_{RG} = S/\gamma = \frac{1}{\nu}\ . \tag{6.13}$$

for each of $N = \frac{Z}{S}$ redundancy groups.

For a system with N redundancy groups, since $\frac{1}{MTTDL_{RG}}$ is very small, we have

$$MTTDL_{system} = \frac{1}{1-(1-\frac{1}{MTTDL_{RG}})^N}$$
$$\approx \frac{MTTDL_{RG}}{N} \tag{6.14}$$

Using the above equations, we find the *MTTDL* of the whole system for each of three mechanisms is as follows:

$$MTTDL_{Mirror\,2} = \frac{\frac{MTTF_{disk}^2}{2 \cdot MTTR_{RG}}}{N} = \frac{MTTF_{disk}^2 \cdot \gamma}{2 \cdot Z}\ ; \tag{6.15}$$

$$MTTDL_{Mirror\,3} = \frac{\frac{MTTF_{disk}^3}{3 \cdot MTTR_{RG}^2}}{N} = \frac{MTTF_{disk}^3 \cdot \gamma^2}{3 \cdot S \cdot Z}\ ; \tag{6.16}$$

$$MTTDL_{RAID\,5+1} = \frac{\frac{MTTF_{disk}^4}{4 \cdot D \cdot (D-1) MTTR_{RG}^3}}{N} = \frac{MTTF_{disk}^4 \cdot \gamma^3}{4 \cdot D \cdot (D-1) \cdot S^2 \cdot Z}\ . \tag{6.17}$$

Using Equations 6.15, 6.16, and 6.17 and Table 6.1, we calculated the MTTDL for each redundancy mechanism using different sizes for a single redundancy group, as shown in Figure 6.4. For each mechanism, MTTDL for both $MTTDL_{disk} = 10^5$ hours and the manufacturers' claims of $MTTDL_{disk} = 10^6$ hours is shown in the figure.

6.3.2 Discussion of MTTDL under Varied Configurations

Our first result is that, as shown in Equation 6.15, MTTDL for the system does not vary with the size of a redundancy group. Though larger redundancy groups require more time for recovery, there are fewer of them, and the likelihood that any redundancy group falls as the total number of sets decreases. These two effects are balanced for Mirror 2, so the size of a single redundancy group does not affect overall system MTTDL. For Mirror 3 and RAID 5+1 mechanisms, however, MTTDL decreases as the size of a single redundancy group increases.

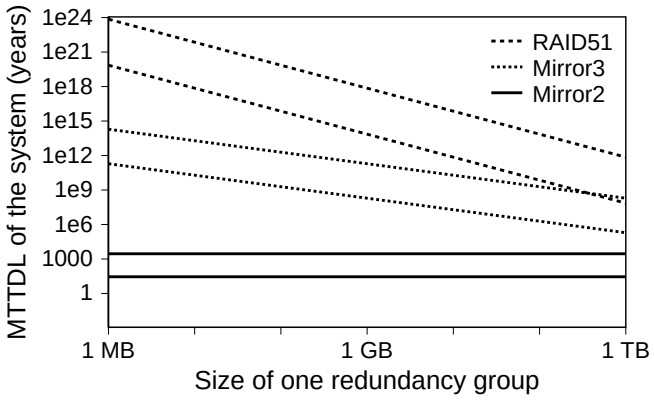


Figure 6.4: Mean time to data loss in a 2 PB storage system. The upper line for each configuration is for a system built from disks with a 10^6 hour MTTF, and the lower line for each configuration is for a system built from disks with a 10^5 hour MTTF.

In both cases, the decrease in reliability due to longer recovery time overwhelms the increased reliability from having fewer, larger redundancy groups.

Figure 6.4 seems to indicate that smaller redundancy groups provide longer MTTDL and higher reliability. However, this approach has several limitations. First, overall file system bandwidth will decrease if redundancy groups are too small because individual disk transfers will be too small. This sets a lower limit of 256 KB–4 MB for redundancy groups. Second, it is assumed that redundancy groups fail independently. If there are too many redundancy groups, however, many will share multiple disks, causing correlated failures. Third, the bookkeeping necessary for millions of small redundancy groups will be overwhelming. For all these reasons, it is unlikely that redundancy groups will be much smaller than 200 MB–1 GB.

Disk lifetime is another important factor in calculating MTTDL for the entire storage system. An order of magnitude improvement in $MTTDL_{disk}$ from 10^5 hours to 10^6 hours can improve overall MTTDL by a factor of 100 for Mirror 2, 1000 for Mirror 3, and 10,000 for

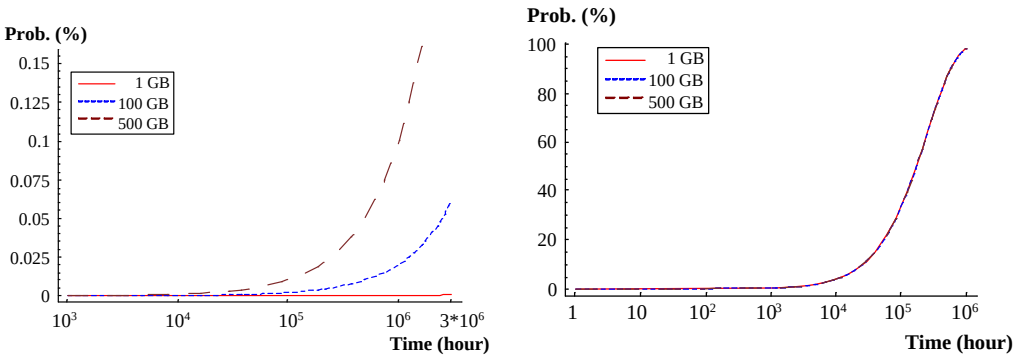
RAID 5+1. The use of a controlled environment to ensure longer disk lifetimes will result in a major benefit in overall system reliability.

Increasing the recovery rate γ can also improve overall system reliability. Placing a higher priority on disk transfer rate and faster recovery will greatly improve reliability by reducing the "window of vulnerability" during which the system may lose data. Doubling the recovery rate will double the reliability of Mirror 2, but will increase the reliability of RAID 5+1 by a factor of eight.

For a system with 2 PB of storage, Mirror 2 can provide sufficient redundancy at an acceptable cost. MTTDL for such a system will be about 30 years regardless of the redundancy group size, allowing us to use larger redundancy groups to reduce bookkeeping overhead. Mirror 3 and RAID 5+1 can provide much longer MTTDL—up to 2×10^8 years for Mirror 3, and 10^{14} years for RAID 5+1 if the size of one redundancy group is 1 GB. It is also interesting that when the size of one redundancy group gets big, close to 100 GB, the reliability achieved by using Mirror 3 will exceed that achieved by using RAID 5+1. This, however, assumes a 10^6 hour *MTTF* for Mirror 3 and a 10^5 hour *MTTF* for RAID 5+1. Although other schemes provide much greater MTTDL, Mirror 2 is considerably simpler to implement than Mirror 3 and RAID 5+1, and provides good reliability at relatively low cost.

6.3.3 Data Loss Distribution Analysis

For the concern of storage efficiency, we are particularly interested in a two-way mirroring (Mirror 2) redundancy scheme: each data block has two copies. The number of replicas is defined as *replication factor*, so Mirror 2 scheme has a *replication factor* of two.



(a) One Single Redundancy Group.

(b) The Whole System.

Figure 6.5: Data Loss Probability of One Redundancy Group and the Whole System

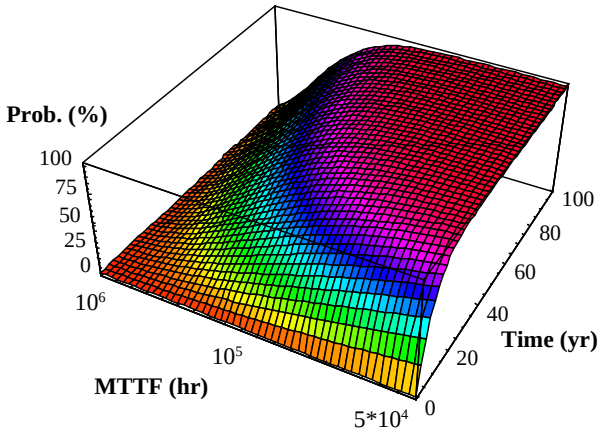


Figure 6.6: Data Loss Probability vs. Disk Lifetime (where the size of a redundancy group is 100 GB).

Assume the initial state of the Markov model [116] for Mirror 2, which is shown as Figure 6.2(a), is that

$$P_0(0) = 1$$

and

$$P_1(0) = P_2(0) = P_3(0) = 0.$$

where $P_i(t)$ is the probability that a failure occurs at state i a given time t. We further get the differential equations 6.18.

$$\begin{cases} \dfrac{\partial P_0(t)}{\partial t} = -2\nu \cdot P_0(t) + \mu \cdot P_1(t) \\ \dfrac{\partial P_1(t)}{\partial t} = 2\nu \cdot P_0(t) - (\mu + \nu) \cdot P_1(t) \\ \dfrac{\partial P_2(t)}{\partial t} = \nu \cdot P_1(t) \end{cases} \tag{6.18}$$

By Laplace transform [116], the probability density function of data loss for Mirror 2 scheme is calculated as Equation 6.19:

$$f_X(t) = \frac{2\nu^2}{\alpha_1 - \alpha_2}(e^{-\alpha_2 t} - e^{-\alpha_1 t}) \tag{6.19}$$

where X is the Random Variable (RV) of the time to data loss and

$$\alpha_1, \alpha_2 = \frac{(3\nu + \mu) \pm \sqrt{\nu^2 + 6\nu\mu + \mu^2}}{2}.$$

Note that

$$\begin{cases} \alpha_1 + \alpha_2 = 3\nu + \mu \\ \alpha_1 \alpha_2 = 2\nu^2 \end{cases}, \tag{6.20}$$

Equation 6.18 can then be written as

$$f_X(t) = \frac{\alpha_1\alpha_2}{\alpha_1 - \alpha_2}(e^{-\alpha_2 t} - e^{-\alpha_1 t}). \tag{6.21}$$

From the probability density function (p.d.f.) shown as the Equation 6.21, it can be seen that data loss by Mirror 2 scheme follows *hypo-exponential* distribution, so that the mean time of data loss is

$$\begin{aligned} E(X) &= \frac{1}{\alpha_1} + \frac{1}{\alpha_2} \\ &= \frac{3\nu + \mu}{2\nu^2}, \end{aligned} \tag{6.22}$$

and the variance of data loss time is

$$\begin{aligned} Var(X) &= \frac{1}{{\alpha_1}^2} + \frac{1}{{\alpha_2}^2} \\ &= \frac{5\nu^2 + 6\nu\mu + \mu^2}{4\nu^4}. \end{aligned} \tag{6.23}$$

It can be further derived that the probability that data loss happens before time T of an individual RG is as Equation 6.24.

$$\begin{aligned} F_X(t) &= Pr_{RG}(0 < t < T) \\ &= \int_0^T \frac{\alpha_1\alpha_2}{\alpha_1 - \alpha_2}(e^{-\alpha_2 t} - e^{-\alpha_1 t})\, dt. \end{aligned} \tag{6.24}$$

where $F_X(t)$ is the cumulative density function (CDF) for time to data loss of one redundancy group.

For the whole storage system, the number of the redundancy groups (RGs) is $N = \frac{Z}{S}$, where Z is the total data capacity, and S is the size of a single redundancy group. Assuming the redundancy groups are independent, and Y is the RV of time to data loss for the system and G_Y is the CDF for the system, then data loss probability of the whole system before time

T is then expressed as Equation 6.14.

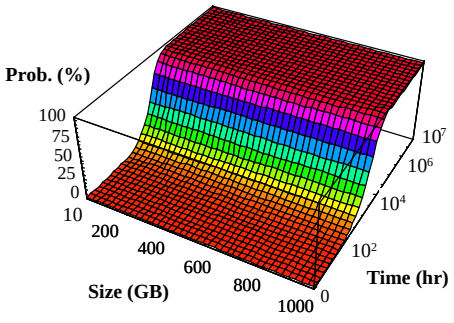
$$G_Y(t) = Pr_{system}(0 < t < T)$$
$$= 1 - (1 - F_X(t))^N. \quad (6.25)$$

Suppose p.d.f. of data loss for the whole system is $g_Y(t)$, which can be derived from Equation 6.21, 6.24, and 6.25.

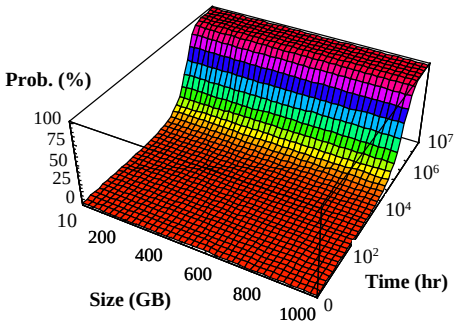
$$g_Y(t) = \frac{\partial G_Y(t)}{\partial t}$$
$$= N \cdot f_X(t) \cdot (1 - F_X(t))^{N-1}. \quad (6.26)$$

Figure 6.5 shows respectively the probability of data loss of a single redundancy group (corresponding to Equation 6.24) and that of the whole system (shown in Equation 6.25) as the size of an individual RG varies from 1 GB to 100 GB to 500 GB, supposing Mean-Time-To-Failure (MTTF) of a single disk is 10^5 hours. The data loss probability of a single RG increases as the its size grows due to longer rebuild time. However, for the whole system, the chance of data loss remains the same for variant RG sizes because the two effects, specifically, the number of redundancy groups and rebuild time for one redundancy group, are balanced.

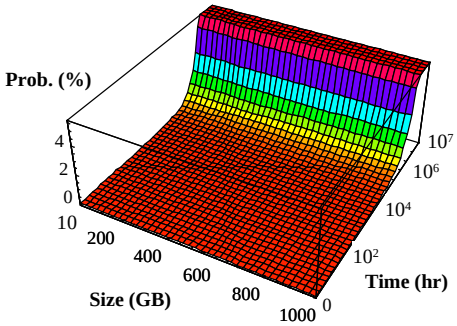
Figure 6.6 shows the relationship of data loss probability and disk lifetime which is represented by the $MTTF$ of a single disk. Also, the relationship of data loss probability and size of redundancy sets is shown in Figure 6.7, with three different assumptions on disk lifetime.



(a) $MTTF_{disk}$: 10^4 *hours.*



(b) $MTTF_{disk}$: 10^5 *hours.*



(c) $MTTF_{disk}$: 10^6 *hours.*

Figure 6.7: Data Loss Probability vs. Size of Redundancy Groups

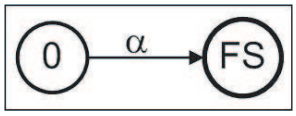
6.4 Disk Infant Mortality and Its Impact in Large-Scale Storage Systems

The calculations of Mean-Time-To-Data-Loss (MTTDL) and the distributions of data loss are only rough estimations of system reliability based on the assumption that failures are exponentially distributed. When a system is up to petabyte scale, the impact of infant mortality on the whole system must be taken into consideration of reliability estimation of the whole system.

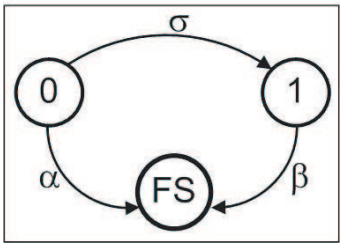
6.4.1 Hidden State Markov Model

A hidden state Markov model is proposed to model disk failure rate behavior. A trivial Markov model with two states and a single failure transition taken with rate α describes a disk with exponential failure rate, as shown in Figure 6.8(a). To obtain a more realistic failure rate, disk states are added in various states of burn-in, depicted in Figure 6.8(b), 6.8(b). The model has two types of transitions, *burn-in* transitions modeling the effect of aging, and *failure* transitions, taken with different rates that model the different probabilities of failure in each burn-in state. The complexity of a model of storage system reliability grows exponentially in the number of states of the disk model, but as we will see, three or four state models agree well with measured disk reliability.

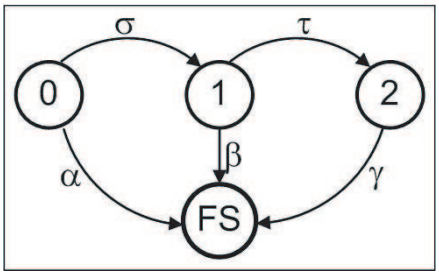
Using the Markov models in Figure 6.8, failure rates for individual disks can be calculated. The calculation starts from the state probability. Suppose we use P_0, P_1, and P_2 to represent the probability that the system is in the initial State 0, in State 1, and in State 2,



(a) Two-state model.



(b) Three-state model.



(c) Four-state model.

Figure 6.8: Hidden state Markov models of disk failure.

respectively. The failure rate in each state is α, β, and γ and the transition probability from State 0 to State 1 is σ and from State 1 to State 2 is τ. The non-failure state probabilities for the three-state model in Figure 6.8(b) are given by the differential equations:

$$\begin{cases} \frac{dP_0(t)}{dt} &= -(\alpha+\sigma)\cdot P_0(t) \\ \frac{dP_1(t)}{dt} &= -\beta\cdot P_1(t)+\sigma\cdot P_0(t) \end{cases}$$

It is given that $P_0(0)=1$ and $P_1(0)=0$, resulting in:

$$\begin{cases} P_0(t) = e^{-(\alpha+\sigma)\cdot t} \\ P_1(t) = \frac{\sigma\cdot(e^{-\beta\cdot t}-e^{-(\alpha+\sigma)\cdot t})}{\alpha-\beta+\sigma} \end{cases}$$

The failure rate of the three-state model is the weighted average of the failure transitions, or $\lambda(t) = (\alpha P_0(t)+\beta P_1(t))/(P_0(t)+P_1(t))$.

Similarly, the non-failure state probabilities for the four-state model in Figure 6.8(b) with the initial conditions $P_0(0)=1$, $P_1(0)=0$, and $P_1(0)=0$ are given by:

$$\begin{cases} P_0(t) &= e^{-(\alpha+\sigma)\cdot t} \\ P_1(t) &= \frac{\sigma\cdot(e^{-(\beta+\tau)\cdot t}-e^{-(\alpha+\sigma)\cdot t})}{\alpha-\beta+\sigma-\tau} \\ P_2(t) &= \frac{e^{-(\beta+\tau)\cdot t}\cdot\sigma\cdot\tau}{(-\beta+\gamma-\tau)\cdot(\alpha-\beta+\sigma-\tau)}+\frac{e^{-(\alpha+\sigma)\cdot t}\cdot\sigma\cdot\tau}{(\alpha-\gamma+\sigma)\cdot(\alpha-\beta+\gamma-\tau)}+\frac{e^{-\gamma\cdot t}\cdot\sigma\cdot\tau}{(\alpha-\gamma+\sigma)\cdot(\beta-\gamma+\tau)} \end{cases}$$

The failure rate for the four-state model is thus:

$\lambda(t) = (P_0(t)\alpha+P_1(t)\beta+P_2(t)\gamma)/(P_0(t)+P_1(t)+P_2(t))$.

6.4.2 Calculation of Markov Model Parameters

If a single disk fails with failure rate $z(t)$, then its probability $R(T)$—the *reliability* at time T—to be alive at time T is $R(T) = \exp(-\int_0^T z(u)du)$ [54]. If the failure rate were a

constant Z during this time, then its reliability at time T would have to be $\exp(-ZT)$ so that we have $Z = (1/T)\int_0^T z(u)du$ for the two reliabilities at T to be the same.

A survey of failure rates of "actually existing disks" shows that the curves do not always have the same shape [37, 38, 105]. Sometimes, the curves follow the bathtub curve closely; *i. e.*, the failure rate starts out at a high level and then sinks continuously to become eventually stable. More often, the failure rate starts at a relatively high value, but then increases even more, reaching a peak from which it then falls to eventually again reach a stationary value. Unfortunately, we do not possess measured values for the failure rate. For this reason, the IDEMA example [36, 112] is used, and a "realistic looking" failure rate is created,shown as "Real-Disk" in Figure 6.9(a). Another model using a linearly decreased disk failure rate in the first year of disk lifespan for comparison is labeled as "linear" in Figure 6.9(a).

We can now determine parameters of the Markov models that best fit the IDEMA example [36, 112]. There are three parameters to fit into five values: the four IDEMA failure rates (0–3 months, 3–6 months, 6–12 months, and 12–72 months) as well as the overall failure rate, for which the exact agreement is obtained. Two slightly different sets of parameters are calculated. Group (A) minimizes the sum of square distance of the "free" failure rates whereas group (B) minimizes the same distance weighted according to the length of the period. For the three state model, accurate agreement is guaranteed for the failure rates in the initial three months. For the four state model, good visual agreement is achieved with some, but not all actual disk failure rates [37, 38, 105] that show a sharp increase in the rate for very young disks followed by a gradual decline to a constant value. Table 6.2 shows the parameters, Table 6.3 shows the fitted failure rates. Unsurprisingly, the hidden state Markov model cannot model the

parameters / model type	α (%/1000h)	β (%/1000h)	γ (%/1000h)	σ (/year)	τ (/year)
HMM-3state-A	0.618059	0.198044	-	2.796275	-
HMM-3state-B	0.619518	0.198261	-	2.829445	-
HMM-4state-A	0.350385	0.636888	0.198788	53.450	3.0400
HMM-4state-B	0.350385	0.638304	0.198809	49.981710	3.067866

Table 6.2: Parameters for the three- and four-state Markov models.

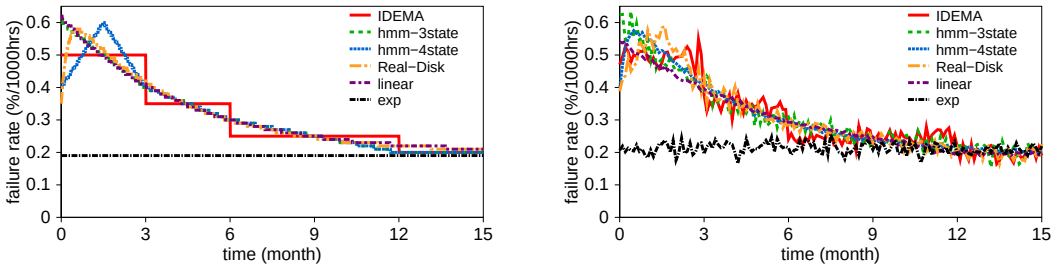
time period (month) / model type	0–3	3–6	6–12	12–72	0–72
HMM-3state-A	0.5	0.347508	0.253143	0.199810	0.222917
HMM-3state-B	0.5	0.346378	0.252261	0.199955	0.222917
HMM-4state-A	0.500108	0.349901	0.250027	0.1999988	0.222917
HMM-4state-B	0.500109	0.350131	0.249821	0.199996	0.222917
IDEMA	0.5	0.35	0.25	0.2	0.222917

Table 6.3: Model fits of three- and four-state Markov models. Failure rate is measured in percent per thousand hours.

sample numbers completely accurately, but it reaches agreement within one percent even for the three state model.

6.4.3 Influence of Failure Rate Curve on System Reliability

Figure 6.9 shows the result of simulating the disk failure rate under the varied failure models for 20,000 disk drives for 15 months. Our simulations ran for six simulated years; however, failure rates for all of the compared models tend to flatten out after disks are more than a year old, so the figure only shows the failure rate for the initial 15 months in Figure 6.9(b). The models being compared include the IDEMA model, three-state hidden Markov model ("hmm-3state"), four-state hidden Markov model ("hmm-4state"), the "realistic looking" failure rate ("Real-Disk"), the linear failure rate model, and the model based on exponential failure rate ("exp"). All of the models except the exponential model show high failure rates for disks at



(a) Failure rate in the first 15 months of a disk's lifetime under under various disk failure models.

(b) Simulated disk failure rate in the first 15 months under various disk failure models (based on 20,000 samples).

Figure 6.9: Comparison of failure rate model of a single disk: the stair-step model proposed by IDEMA, three- and four-state Hidden Markov Models, real disk model, linear model, and exponential model.

young age, although their curve shapes differ.

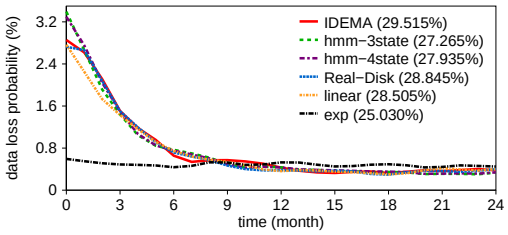
To ascertain the influence of the shape of the failure rate curve, we explored the effects of infant mortality in large-scale storage systems using simulations. We first varied the number of disk drives in a storage system and found that the distribution of data loss probability over six years differs when infant mortality is taken into account. Next, we studied the various disk replacement strategies, and showed which strategies provided the best overall system reliability when infant mortality was considered. Based on these studies, we propose an adaptive data redundancy scheme to reduce the impact of infant mortality.

6.4.3.1 Simulation Methodology

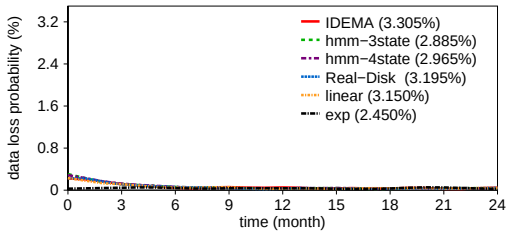
To examine the effect of infant mortality on large storage systems, discrete event-driven simulations built with the PARSEC simulation tool [77] were conducted.

In a simulated storage system, data is distributed randomly by a placement algorithm – RUSH [55], which probabilistically guarantees that data will be distributed evenly across all of the disk drives. RUSH supports data redundancy and gives a list of disk IDs where the current pieces in each redundancy group reside on along with the additional disk IDs which will be used as the locations of the recovery targets during data recovery processes. We inject the varied failure models for individual disk drives, including IDEMA, Hidden Markov Models, and the exponential model, into the simulated storage system composed of thousands of disks. Based on these models, the time to disk failure can be simulated, which is used to schedule the failure events. Whenever a failure event occurs, a data recovery event will then be triggered and the data on the failed disk will be copied from the recovery sources to the recovery targets. When there is additional failure(s) happening during the data recovery process and the data cannot be rebuilt anymore, it is counted as one occurrence of "data loss" and the timestamp is recorded.

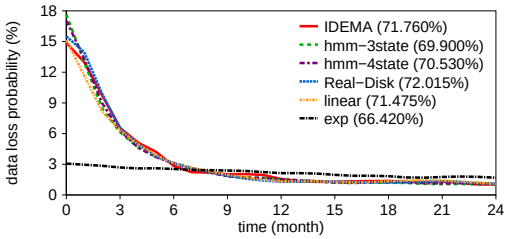
Several parameters are varied in our simulation, including the number of disk drives in a simulated system, the disk failure model, data redundancy configuration (mirroring or RAID-5 like parity scheme), and data recovery method (distributed recovery with FARM or traditional disk rebuilding method referred as "Non-FARM"). For each configuration, we simulate the system over a period of six years and repeat the simulation for thousands of times



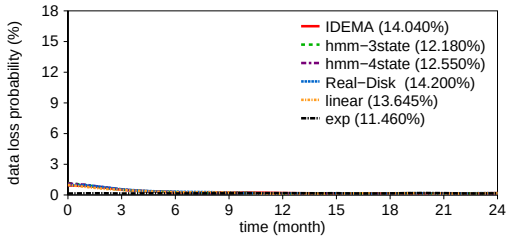
(a) Non-FARM, two-way mirroring.



(b) FARM, two-way mirroring.



(c) Non-FARM, RAID 5



(d) FARM, RAID 5

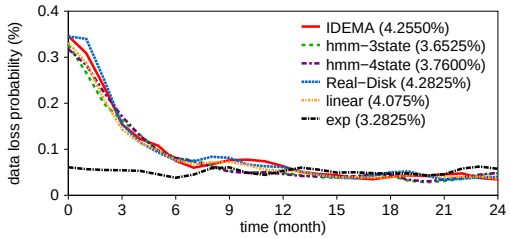
Figure 6.10: Distribution of data loss probability for a system with 10,000 disk drives over six simulated years. Only first two 24 months are shown in the graphs; the percentages on the labels give the data loss probability over six years for each model.

to gather data samples. The data loss probability is calculated as the total occurrence of data loss divided by the total number of simulated samples. According to the recorded timestamps of data loss occurrence, we also calculate the distribution of the data loss time and plot the probability density function curves of the data loss distribution.

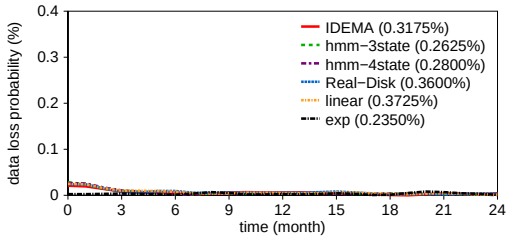
We examined the distribution of data loss probability over six simulated years, comparing the systems with FARM for distributed recovery and those that used traditional dedicated data recovery ("non-FARM"). Our results, shown in Figure 6.10, indicate much higher data loss probability at an early stage using models that consider infant mortality than using models with the exponential failure rate. The total data loss probabilities over the six simulated years are also listed along with the legends in Figure 6.10. It has been observed that, the data loss probability among various models which take the "infant mortality" into account only differs slightly, implying that the exact shape of the bathtub curve is less critical. Further, it shows a significant difference between the exponential model with constant failure rate and the models with non-constant failure rate which consider infant mortality. It is also noted that FARM can greatly reduce the probability of data loss and thus decreases the effect of infant mortality on large storage systems.

6.4.3.2 Effects of Infant Mortality

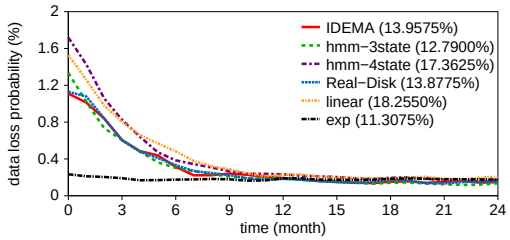
We simulated systems with various redundancy mechanisms and either 1,000 disks or 10,000 disks over a period of six years, assuming that initially all drives were not burnt in, with the results shown in Figures 6.11 and 6.10. Two kinds of data redundancy schemes were simulated: two-way mirroring and 4+1 RAID 5. We configured the system with FARM,



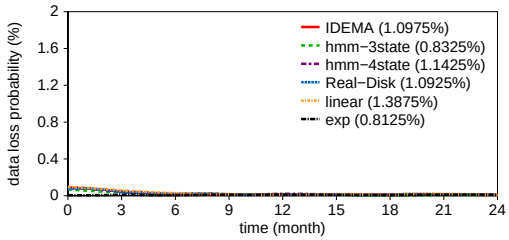
(a) Non-FARM, two-way mirroring.



(b) FARM, two-way mirroring.



(c) Non-FARM, RAID 5



(d) FARM, RAID 5

Figure 6.11: Distribution of data loss probability for a system with 1000 disk drives over six simulated years. We only show the first two 24 months in the graphs; the percentages on the labels give the data loss probability over six years for each model.

described in Chapter 4 and without FARM respectively. The figures only include the first 24 months in the graphs to emphasize the differences between the naïve model of disk failure and our model that includes infant mortality. After the first 24 months, the data loss probability remains stable since disk drives become mature and their failure rates stay the same after the infant period until the end of disk lifespan (six simulated years). In the graphs, the percentages on the labels give the data loss probability over six years for each model. We observed the presence of higher data loss probability during the first 12 months under the disk failure models that take infant mortality into consideration. This effect does not appear under the exponential model which assumes disk failure rate remains unchanged in a disk's lifespan. The effect of infant mortality is more pronounced for larger systems, as can be seen by comparing the systems with 1,000 and 10,000 disks, implying that designers of large storage systems that store petabytes of data must consider the effects of infant mortality. Note that the Y-axis scales in Figure 6.11 and 6.10 are not the same. It also have been noticed that our system reliability did not differ considerably under the five models of infant mortality: IDEMA, 3-state hidden Markov, 4-state hidden Markov, "real disk," and linear model. This shows that, although disk infant mortality itself is very important, the precise shape of the failure rate curve is less crucial.

6.5 Data Protection Based on the Age of Disk Drives

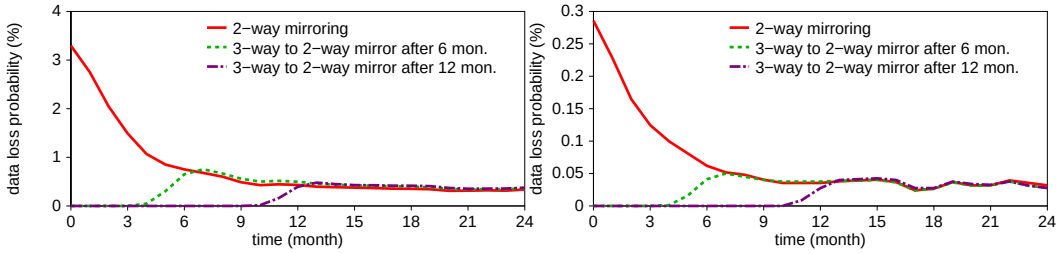
In addition to judicious replacement of drives, we can also adjust the redundancy to the risk inherent in the system. We considered two simple schemes. First, instead of simply

Table 6.4: Data Loss Probability for Adaptive Redundancy Schemes.

	without FARM	with FARM
2×mirror	27.935%	2.96%
$3 \rightarrow 2\times$ after 6 mo.	21.02%	1.96%
$3 \rightarrow 2\times$ after 12 mo	18.38%	1.72%
4+1 RAID5	70.53%	12.55%
4+2 $\rightarrow$ 4+1 after 6 mo.	62.14%	9.12%
4+2 $\rightarrow$ 4+1 after 12 mo.	58.17%	7.96%

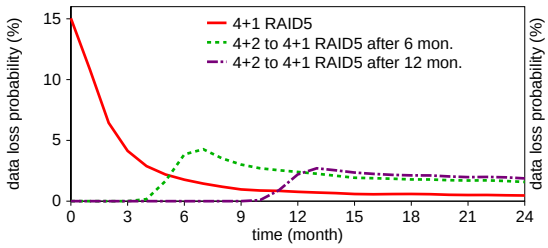
mirroring objects when one object is stored on disk that is less than T months old, we kept three copies of the object. Second, we use $m+k$ redundancy unless one of the object groups is stored on a disk less than T months old. If this is the case, we add one additional parity object. In both schemes, if there is only one disk that is less than T months old and if the randomly selected new disk is older, then we merely replace the not yet burnt in disk with the older one.

In our first experiment, we simulated a system with 10,000 initially new disks, all using the failure rate predicted by our 4 state Markov model. We used replication and $m+k$ RAID 6 as our redundancy scheme. In the adaptive scheme, we switched from a more aggressive redundancy scheme (triplication, 4+2) to a less aggressive scheme (mirroring, 4+1 RAID 5) after six and twelve months respectively for a total six years of simulation time. The data loss probabilities over the six years under varied configurations (based on 20,000 data points for each scheme) are shown in Table 6.4. The curves of the probability density function of data loss distribution over time are further plotted to examine the infant mortality effect under the adaptive data redundancy schemes. As shown in Figure 6.12, the data loss probability during the early stage of the storage system is greatly reduced by the adaptive data redundancy schemes as compared to the static redundancy scheme.

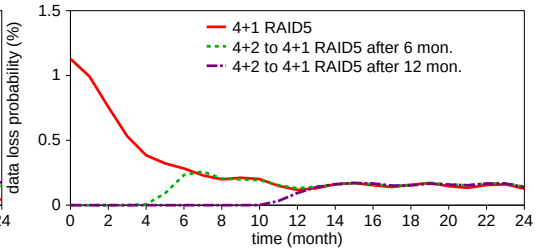


(a) Non-FARM, two-way mirroring.

(b) FARM, two-way mirroring.



(c) Non-FARM, RAID 5



(d) FARM, RAID 5

Figure 6.12: Distribution of data loss probability for a system under adaptive data redundancy schemes.

6.6 Summary

System modeling has long been a research subject for analyzing reliability of storage systems, but reliability of a petabyte-scale system has not yet been well studied. In this chapter, one of the important metrics of system reliability, Mean-Time-To-Data-Loss, has been estimated by Markov models under several different redundancy configurations. By introducing Markov models with hidden states, the impact of infant mortality on system reliability has been analyzed and simulated. The results indicate a high probability of data loss during the early stage of a large-scale storage system due to the effect of infant mortality. In response, an adaptive data redundancy scheme is presented. By adjusting the level of data redundancy based on the age of disk drives, the adaptive scheme minimizes the impact of infant mortality and thus decreases the probability of data loss in the system.

Chapter 7

Failure Impacts on Network Interconnection

Chapter 4, 5 and 6 focus on the schemes and analysis of recovery and data layout mechanisms to deal with disk failures, which lead to data loss. In a petabyte-scale storage system with thousands of nodes and a complicated interconnect structure, other components also fail, such as routers and network links. Strong robustness in network interconnections is highly desired but difficult to achieve. This chapter discusses the topologies of network interconnection and their robustness under various failures.

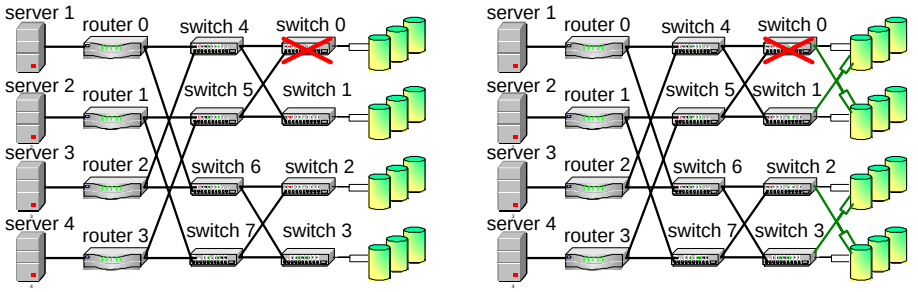
7.1 Robustness in Network Interconnection

Failures, which appear in various modes, may have several effects on a large-scale storage system. The first is connectivity loss: requests or data packets from a server may

not be delivered to a specific storage device in the presence of link or switch failures. The result is disastrous: many I/O requests will be blocked. Fortunately, today's storage systems include various levels of redundancy to tolerate failures and ensure robust connectivity. The second effect is bandwidth congestion caused by I/O request detouring. Workload analysis of a large Linux cluster with more than 800 dual processor nodes at Lawrence Livermore National Laboratory [120] shows that I/O requests are very intensive in supercomputing environments. They arrive on average about every millisecond. The average size of a single I/O request can be as large as several megabytes, as plotted in Figure 3.3 and 3.4. Suppose that such a large system suffers a failure on a link or delivery path on an I/O stream. In this case, the stream has to find a detour or come to a temporary standstill. The rerouting will bring I/O delays and bandwidth congestion and might even interrupt data transfer. The I/O patterns particular to supercomputing demand a network architecture that provides ultra-fast bandwidth and strong robustness simultaneously. The third effect is data loss caused by the failure of a storage device. As disk capacity increases faster than device bandwidth, the time to write and hence to restore a complete disk grows longer and longer. At the same time, the probability of single and multiple failure increases with the number of devices in the storage system.

7.2 Network Interconnection Architectures

Modern supercomputing systems require a high bandwidth storage network storing petabytes of data. Traditional storage architectures, such as RAID [21], Storage Area Network (SAN) [91], and Network Attached Storage (NAS) [44] cannot meet the needs for bandwidth



(a) A failure on switch 0 disconnects several devices.

(b) Add spare links between switches and devices.

Figure 7.1: Butterfly Networks Under Failures.

and scale of such a large storage system. Network topologies for massively parallel computers are better suited to build large storage networks [56], as they are capable of delivering high performance and dealing with very large scale.

There are several potential strategies for interconnect architectures, such as a simple hierarchical structure, butterfly networks, meshes, and hypercubes [7]. The analysis of the total system cost and comparison of system bandwidth under failure-free status of each strategy were discussed in [56]. This work focuses on the failure impact under these topologies.

The simple tree-based hierarchical structure requires large routers, which make the configuration very expensive [56]. Also, it cannot provide sufficient fault tolerance as it suffers from the failures of higher-level nodes. Butterfly networks cannot offer high reliability either, because there is only a single path from a server to a storage device. For example, when a failure occurs at switch 0, as shown in Figure 7.1(a), a number of storage devices will lose

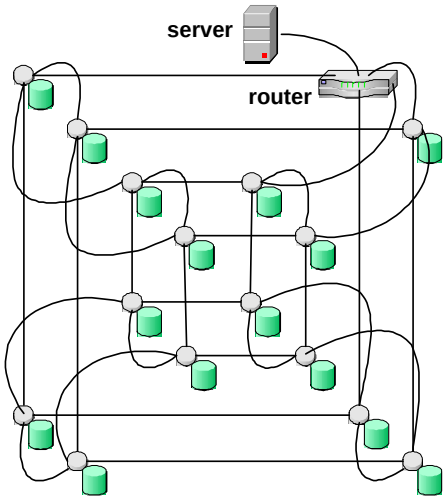


Figure 7.2: A Hypercube Structure.

their connections to the system. One way to solve this problem is to add spare links as shown in Figure 7.1(b). However, switch 1 then becomes over-loaded as all the requests to the storage devices attached with switch 0 will now go through it. It also requires more expensive switches with more ports for spare links. Furthermore, there may be additional link outages on the path from a server to a storage device, which will break the connection.

Cube-like architectures, such as meshes (Figure 7.3(a)), hypercubes (Figure 7.2), and torus are structured with multiple routes between servers and storage devices, and thus more resilient to failures. However, the system cost for these cube-like structures is higher than that for simple tree structure and butterfly networks.

7.2.1 Failure Scenarios

Three types of failure scenarios are considered: link failure, connection node failure, and storage device failure.

I. *link failure*: The connection between any two components in a system can be lost. If there exists only one path between two components, a system is at risk when any link along this single path is broken. A robust network interconnection must be tolerant of link failures. Multiple paths between two components will decrease the vulnerability of a single-point of failure and effectively balance I/O workload.

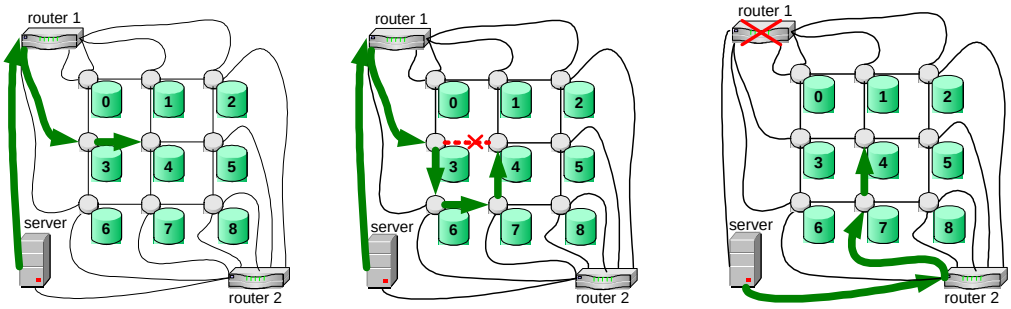
II. *connection node failure*: Connection nodes include switches, routers, and concentrators that link servers to storage nodes. They are used for communications and do not store any data. Compared with link outage, failures on an entire switch or router are more harmful for network connection since a number of links that were attached on the switch or the router are simultaneously broken, but losing connection nodes will not directly lead to data loss.

III. *storage device failure*: When a storage device fails, it cannot carry any load. Further, additional traffic for data reconstruction will be generated. The increase in bandwidth utilization brought by data construction is of great concern when data is widely declustered in such a system. Several redundancy mechanisms for such very large storage systems have been discussed in Chapter 4 and 6.

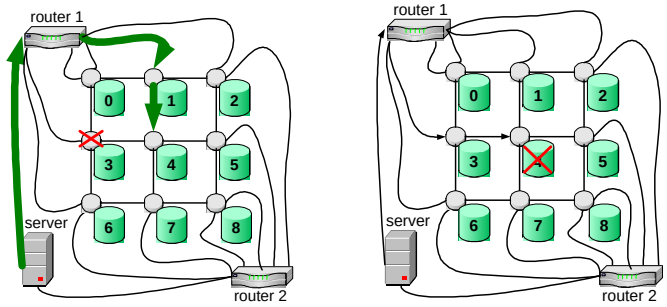
As a case study, four kinds of possible failures were examined in a 3×3 2D mesh storage system shown in Figure 7.3. In this example, there are nine storage devices (labeled from 0 to 8), two routers and one server. Assume a specified I/O stream will be sent from the server to a storage device, say node 4. The path of the I/O stream is traced in various scenarios. At the initial state without any failures, the I/O request can be simply transferred via router 1 and node 3 as indicated in Figure 7.3(a). However, a rerouting strategy has to be introduced if failure occurs. If the link between node 3 and node 4 is broken, as shown in Figure 7.3(b), the I/O request has to take an alternate route to get to its target. It can pick up a path (i) $\{$router 1$\rightarrow$node 3$\rightarrow$ node 6$\rightarrow$node 7$\rightarrow$node 4$\}$ as shown in Figure 7.3(b), or (ii) $\{$router 1$\rightarrow$node 1$\rightarrow$node 4$\}$, or (iii) $\{$router 2$\rightarrow$node 7$\rightarrow$node 4$\}$, etc. The choice of the new path is determined by the routing protocol and system status at that moment. Figure 7.3(c) and 7.3(d) show further examples of detouring due to a router and a switch failure respectively. The worst case in failure modes is that the target node fails, as shown in Figure 7.3(e). If either the switch fails or the disk drive crashes on the target node, the I/O request cannot be delivered.

7.3 Evaluation of Failure Impacts on Network Interconnection

It is expensive to build a real petabyte-scale storage system in order to evaluate the impact of failures on interconnection networks. Instead, we use simulations of large-scale systems to develop a better understanding of the impact of failures. Several network interconnection topologies are simulated and varied kinds of failures are injected to evaluate system



(a) initial state (b) link between node 3 and 4 fails (c) router 1 fails



(d) switch on node 3 fails (e) node 4 fails

Figure 7.3: A Storage System Structured as a 3×3 2D mesh Under Degraded Modes

behavior under degraded mode and estimate system reliability of a petabyte-scale storage system.

7.3.1 Assumptions

Generally, nodes are classified into two types: *storage nodes* that contain data, such as disk drives; and *connection nodes* that are used only for communication, such as routers and switches. Node failure and link failure are investigated in our system. In reality, there are many other types of failure, such as power failure and software failure. Our failure model simply focuses on network interconnection, but does not consider the Byzantine failure model under which arbitrary or malicious failures would appear. It is also assumed that all failures be detected in a timely manner.

I/O requests are assumed to be very intensive and in large size. User data is spread out over the whole system evenly.q Dijkstra's algorithm [28] is used as the routing algorithm. This algorithm helps us understand the network status and trace the path of each I/O request, although it cannot scale to large networks due to its dependence on global information. Some failure-resilient routing techniques, such as wormhole routing [82], can also be used in this system. The buffer/cache issues of routers and switches are not considered for simplification.

7.3.2 Simulation Methodology

The impact of failures on a petabyte-scale storage system is evaluate under various interconnection configurations by event-driven simulation. The simulator, implemented in C++, can evaluate the failure impact on a system under various configurations. There are three

Table 7.1: Parameters for butterfly, mesh and hypercube topology.

parameter	butterfly	mesh	hypercube
number of servers	128	128	128
number of disks	4096	4096	3968
number of routers	128	8	128
total number of links	7552	16,392	23,612
total number of nodes	4736	4232	4224

main pieces in our simulator: topology, failures, and requests. The network interconnection architecture was implemented as a class object Graph with the functions for building the network topology, *i.e.* build_nodes and build_links. The function inject_failures sets up the degraded system mode under which one or more failures happen in an overlapped time period. Servers send out I/O requests under a synthetic workload based on our analysis of a Linux cluster for supercomputing applications [120].

Three kinds of interconnection topologies have been simulated: a multi-stage butterfly network, a 64×64 2 D mesh shown in Figure 7.3(a) and a 6 D hypercube shown in Figure 7.2. Hospodor and Miller [56] estimated the required number of nodes and ports for a petabyte-scale storage system using butterfly, mesh and hypercube topology. Based on their estimation, Table 7.1 lists the parameters set up in the simulator. The butterfly network is a hierarchical structure with one level of routers, three levels of switches with 128 switches per level. In the 64×64 2D mesh, each router is connected to the edge nodes and the interior nodes are connected with four other nodes. In the hypercube topology, each storage device is attached to a 12-port switch and each router has two additional ports connected to servers.

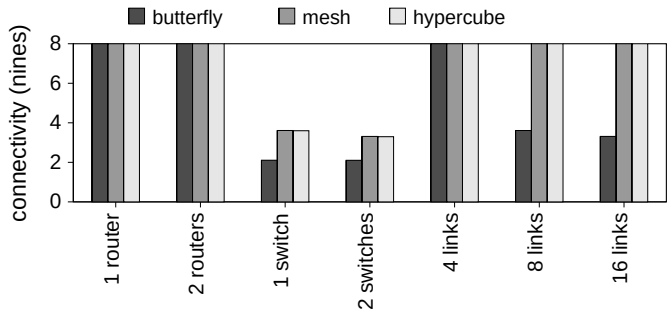


Figure 7.4: I/O path connectivity

7.3.3 Simulation Results

In our simulated system, servers send I/O requests in parallel to storage devices at an inter-arrival rate of 1 millisecond. Several degraded scenarios are set up for three network topologies—butterfly network, 64×64 2D mesh, and hypercube topologies, including varied number of link failures and node failures. Each I/O request is traced to count the number of hops that the request has walked and to record the load on each link in the system. The ratio of the requests which cannot be delivered to the target device due to failures under various degraded modes is also calculated, which helps to measure how well the system is connected. We also show the average number of hops of I/O requests under varied degraded modes and compare the link load in the neighborhood of failures with average links.

7.3.3.1 I/O path connectivity

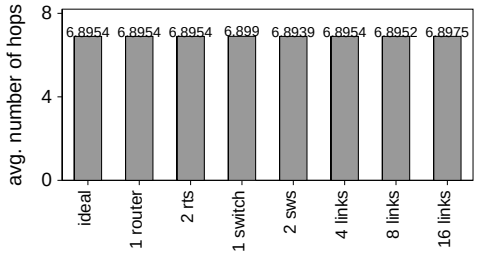
The first and the most important aspect of robustness of network interconnection is that an I/O request can be delivered to its target storage device. Such an ability is referred as *I/O path connectivity*. We borrow the metric of system availability [32] and measure the

connectivity in units of "nines," which is defined as $-log_{10}(1-P)$, where P is the fraction between the number of I/O requests that can be successfully sent to the targets and the total number of I/O requests during a period of time. Three "nines" connectivity means that 99.9% of the I/O requests can be delivered to their targeted storage devices.

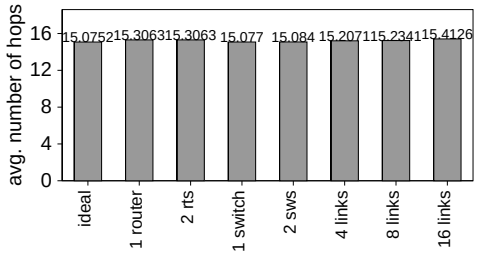
All the I/O request are traced sent in 60 seconds under seven failure modes: with one, two routers failed, with one, two switches failed, and with four, eight, and sixteen links failed. Failures cannot be repaired during 60 seconds even if they can be detected. Our results are reported in Figure 7.4. It is found that failures on switches have greater influence than those on routers and links. The butterfly network suffers greatly from broken switches, as discussed in Section 7.2. As expected, the 6D hypercube and 2D mesh structure achieve a better connectivity than the butterfly network, although up to 0.05% of the requests did not arrive at their target devices when two switches failed. As for link failures, every I/O request found a healthy path in the presence of up to sixteen broken network links under 2D mesh and 6D hypercube topologies, but about 0.048% of the requests were not delivered successfully when 16 links were broken under the butterfly network structure. Within 60 seconds, on the order of 10^8 I/O requests were sent from the servers in our simulation. As a result, the accuracy of our reliability measurement is up to eight nines ($-log_{10}(1-\frac{10^8-1}{10^8}) = 8$).

7.3.3.2 Number of hops for I/O requests

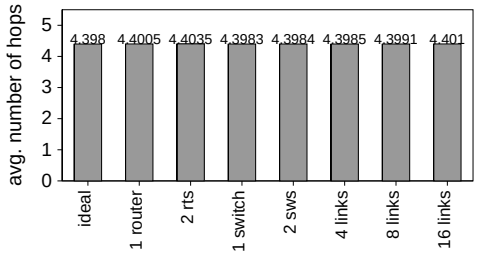
The number of hops is calculated as the number of links that an I/O request has to travel through the system to arrive at its targeted device. It is an important metric for both I/O latency and system bandwidth. The minimum number of hops is measured in the



(a) Butterfly networks.



(b) 64×64 2D mesh.



(c) 6D hypercube with 4,096 nodes.

Figure 7.5: Average number of hops per I/O request

simulator; while in reality, an I/O request may go through more steps than the minimum for the considerations of load balance.

An ideal fault-free case (labeled as “ideal”) is compared with seven degraded modes: with one, two routers failed, one, two switches failed, and with four, eight, and sixteen links failed (Figure 7.5). The case when there is no path for an I/O request is not counted in the calculation of the average number of hops. Compared with the ideal connection, the average number of hops is only slightly higher under all degraded modes in all three topologies. There are two underlying reasons for this: first, the proposed butterfly, mesh, and hypercube structures provide redundant paths and thus lead to good fault tolerance; second, the possibility that a failed component is on the path of many I/O requests is small due to the limited number of I/O requests during a short period time. As a result, the number of average hops remains nearly at the same level under the examined degraded modes. For a well-chosen topology which does not suffer from a single point of failure, the system would be robust unless many components fail at once. This occurrence only happens under certain circumstances such as large-scale power outages, which can easily pull down any local-area networks.

7.3.3.3 Failure impact on network neighborhood

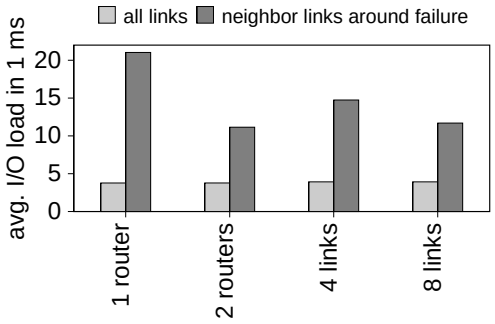
One of the important system behaviors after failures is request rerouting. The failure impact on its neighborhood links/nodes is not negligible. An abrupt increase in network load around the neighborhood of failures can overload a certain number of nodes and links, such that I/O requests may not be delivered successfully. In order to analyze the failure impact on the network neighborhood, the network links around failures are monitored and their I/O load

is compared with the average load on all the links in the system. We observed a pronounced increase in the average I/O load on neighboring links around failures under four degraded modes: with one router, two routers, four links, and eight links failed (as Figure 7.6 shows.)

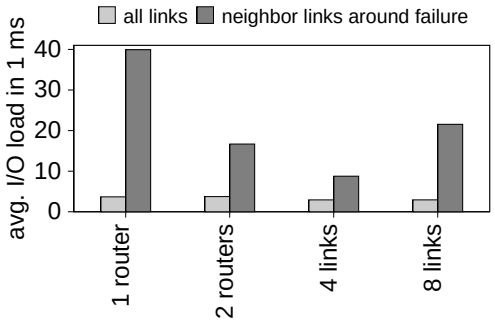
Comparatively, neighbors around a failed router carry more I/O load than those around a failed link in most cases. This phenomenon comes from the different functionalities of routers and links. It is also noted that neither the butterfly network nor 2D mesh structure balances the load around failures, but the hypercube topology handles it well. The link load around failures is four to thirteen times higher than average link load in the butterfly network and 2D mesh system, whereas it is not obviously higher than the average link load in the 6D hypercube structure. This is because there are fewer network links and much weaker path redundancy in the butterfly network and 2D mesh structure than those in the 6D hypercube topology. Our results indicate that for a petabyte-scale storage system, although butterfly network and mesh structure can provide decent I/O path connectivity without increasing the number of hops, they cannot deal with neighborhood load increase as gracefully as the 6D hypercube structure.

7.3.4 Result Discussion

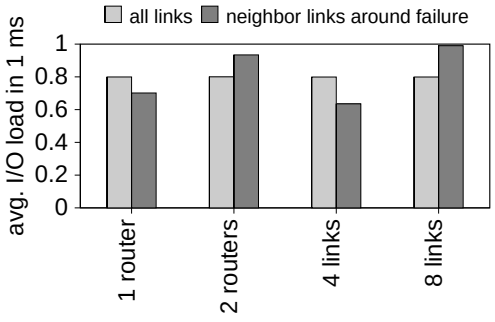
In our simulated petabyte-scale storage system connected by a butterfly network, a mesh or a hypercube architecture, four to sixteen link failures do not result in an obvious increase in the number of hops for I/O requests. This shows good fault tolerance to link failures under these three network interconnection topologies. Switch failures are much more likely to cause I/O path disconnect. The results show that the butterfly network can survive



(a) Butterfly networks with 4,096 disk drives.



(b) 64×64 2D mesh.



(c) 6D hypercube with 4,096 nodes.

Figure 7.6: I/O load comparison of the neighborhood links around failures and all the links in a system.

under router failures but is very sensitive to switch failures and sixteen link outages; while the average number of I/O hops in the 2D mesh structure is one to two times higher than that in the butterfly network and the hypercube structure. This indicates that different network topologies have their pros and cons under different system degraded modes. The impact of failures on neighborhood links is significant, especially for the multi-stage butterfly network and 2D mesh structure, which may lead to network congestion and the slowing down of data transfers. Based on our results, the hypercube structure pays off its higher system cost and outperforms the butterfly network and 64×64 2D mesh in the robustness of network interconnection.

7.4 Summary

Robust network interconnects are essential to large-scale storage systems. Various failure scenarios and their impacts on a petabyte-scale storage system are studied in this chapter. The fault-tolerance capacity of three potential network topologies, namely, multi-stage butterfly network, 2D mesh, and 6D hypercube structures, has been evaluated by simulation. We examined I/O path connectivity, the number of hops for I/O requests, and the I/O load on neighborhood links around failures. Our simulation results have shown that a well-chosen network topology is capable of ensuring a high tolerance of failures. Router and switch failures have a larger impact on network robustness than link failures, and the neighborhood around failures suffers more greatly than average link load, especially for butterfly network and 2D mesh. Our simulator can flexibly investigate various network topologies with injection of any degraded modes, which enables us to estimate robustness of network interconnections

and helps the system architects with their design decisions on building reliable petabyte-scale storage systems.

Chapter 8

Conclusions

The increasing challenge and need for data storage have driven our research on petabyte-scale storage systems. As system scales up, new reliability phenomena emerge in multitude. With the aim of improving system reliability, three main topics of particular interest have been address in this thesis: (1) fast data recovery, (2) advanced reliability modeling and estimation, and (3) robust network interconnection.

Traditional redundancy schemes can not guarantee sufficient reliability for petabyte-scale storage systems. Simply increasing the number of replicas is cumbersome and costly for read/write systems; using erasure coding (m out of n) schemes, while cheaper, increases complexity. In response, FARM, a fast recovery scheme that distributes redundancy on demand, is presented. Using FARM, data objects are collected into redundancy groups that may use various replication and erasure-coding schemes. These redundancy groups are then distributed across the whole system according to a deterministic data placement algorithm, RUSH. The results from event-driven simulations have demonstrated that FARM effectively improves re-

liability for large-scale storage systems. The discussions of data layout schemes have shown that FARM can work in concert with more advanced data layout mechanisms such as two-level hierarchical arrays to further improve system reliability.

There are many unsolved problems of precisely modeling the reliability of disk drives and a whole storage system. The effect of disk infant mortality has been examined on storage systems with tens of thousands of disks in this thesis. By comparing the data loss probability of a two-petabyte storage system under different disk models, it has been shown that the *exact* shape of the failure rate curve is less important than the mere consideration of infant mortality. This opens the possibility of using analytical Markov models to evaluate the reliability of systems of this size. To avoid the "cohort effect" brought by disk infant mortality, new disks ought to be introduced into a system in reasonably small batches—when disks need to be replaced because of their age, a gradual approach is noticeably better. In addition to limiting the cohort effect, the data protection scheme can be adjusted in response to its presence. It has been shown that an adaptive data redundancy scheme that adds a stronger data protection for a batch of disk drives at their young ages will greatly reduce the high data loss rate brought by the effect of infant mortality.

Robust network interconnects are essential to large-scale storage systems. Various failure scenarios and their impacts on a petabyte-scale storage system are studied. The fault-tolerance capacity of three potential network topologies, namely, multi-stage butterfly network, 2D mesh, and 6D hypercube structures, has been evaluated by simulation. Also, I/O path connectivity, the number of hops for I/O requests, and the I/O load on neighborhood links around failures are examined. The simulation results have shown that a well-chosen network

topology, such as 2D mesh or 6D hypercube, is capable of ensuring a high tolerance of failures. Router and switch failures have a greater impact on network robustness than link failures, and the neighborhood around failures suffers more greatly than average link load, especially for butterfly network and 2D mesh.

This thesis as a whole has addressed several problems regarding to high reliability of very large-scale storage systems. By developing data redundancy, recovery schemes, advanced reliability modeling, and robust network interconnection, this work makes efforts towards understanding and coping with failures in large-scale storage systems and helps system designers ensure high reliability for petabyte-scale storage systems.

Bibliography

[1] A. Adya, W. J. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, Dec. 2002. USENIX.

[2] G. A. Alvarez, W. A. Burkhard, and F. Cristian. Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 62–72, Denver, CO, June 1997. ACM.

[3] D. Anderson. Object based storage devices: A command set proposal. Technical report, National Storage Industry Consortium, 1999.

[4] D. Anderson, J. Dykes, and E. Riedel. More than an interface–SCSI vs. ATA. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST)*, San Francisco, CA, Mar. 2003.

[5] D. C. Anderson, J. S. Chase, and A. M. Vahdat. Interposed request routing for scalable network storage. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2000.

[6] T. Anderson, M. Dahlin, J. Neefe, D. Pat-terson, D. Roselli, and R. Wang. Serverless network file systems. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95)*, pages 109–126, Copper Mountain Resort, Colorado, Dec. 1995.

[7] W. C. Athas and C. L. Seitz. Multicomputers: message-passing concurrent computers. *IEEE Computer*, 21:9–24, Aug. 1988.

[8] S. H. Baek, B. W. Kim, E. J. Joung, and C. W. Park. Reliability and performance of hierarchical RAID with multiple controllers. In *Proceedings of the Twentieth ACM Symposium on Principles of Distributed Computing (PODC 2001)*, pages 246–254. ACM, 2001.

[9] J. T. Blake and K. S. Trivedi. Reliabilities of two fault-tolerant interconnection networks. In *Proceedings of the 18th International Symposium on Fault-Tolerant Computing (FTCS '88)*, pages 300–305, 1988.

[10] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computers*, 44(2):192–202, 1995.

[11] P. J. Braam. The Lustre storage architecture, 2002.

[12] S. A. Brandt, L. Xue, E. L. Miller, and D. D. E. Long. Efficient metadata management in large distributed file systems. In *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 290–298, Apr. 2003.

[13] A. B. Brown. Accepting failure: Availability through repair-centric system design. Qualifying Proposal, 2001.

[14] A. B. Brown and D. A. Patterson. Embracing failure: A case for recovery-oriented computing (ROC). In *2001 High Performance Transaction Processing Symposium*, Oct. 2001.

[15] W. A. Burkhard and J. Menon. Disk array storage system reliability. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS '93)*, pages 432–441, June 1993.

[16] L.-F. Cabrera and D. D. E. Long. Swift: Using distributed disk striping to provide high I/O data rates. *Computing Systems*, 4(4):405–436, 1991.

[17] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*, 1999.

[18] M. Castro and B. Liskov. Proactive recovery in a Byzantine-fault-tolerant system. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, 2000.

[19] F. Chang, M. Ji, S.-T. A. Leung, J. MacCormick, S. E. Perl, and L. Zhang. Myriad: Cost-effective disaster tolerance. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pages 103C–116, 2002.

[20] P. M. Chen, E. K. Lee, A. L. Drapeau, K. Lutz, E. L. Miller, S. Seshan, K. Shirriff, D. A. Patterson, and R. H. Katz. Performance and Design Evaluation of the RAID-II Storage Server. *Journal of Distributed and Parallel Databases*, 2(3):243–260, July 1994.

[21] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2), June 1994.

[22] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. An empirical study of operating system errors. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 73–88, Lake Louise, Alberta, Canada, Oct. 2001.

[23] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[24] H. Company. HP white paper: an analysis of RAID 5DP - a qualitative and quantitative comparison of RAID levels and data protection, July 2001.

[25] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST)*, pages 1–14. USENIX, 2004.

[26] P. F. Corbett and D. G. Feitelson. The Vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, 1996.

[27] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 202–215, Banff, Canada, Oct. 2001. ACM.

[28] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[29] Disk drive specification: IBM Deskstar ^{TM} 180GXP hard disk drives.

[30] J. R. Douceur. The Sybil attack. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, Mar. 2002.

[31] J. R. Douceur and R. P. Wattenhofer. Competitive hill-climbing strategies for replica placement in a distributed file system. In *Proceedings of the 15th International Symposium on Distributed Computing (DISC)*, pages 48–62, Oct. 2001.

[32] J. R. Douceur and R. P. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *Proceedings of the 9th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '01)*, pages 311–319, Cincinnati, OH, Aug. 2001. IEEE.

[33] J. R. Douceur and R. P. Wattenhofer. Optimizing file availability in a secure serverless distributed file system. In *Proceedings of the 20th Symposium on Reliable Distributed Systems (SRDS '01)*, pages 4–13, New Orleans, LA, Oct. 2001. IEEE.

[34] J. Duato. A theory of fault-tolerant routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):790–802, 1997.

[35] J. G. Elerath. Specifying reliability in the disk drive industry: No more MTBF's. In *Proceedings of 2000 Annual Reliability and Maintainability Symposium*, pages 194–199. IEEE, 2000.

[36] J. G. Elerath. Specifying reliability in the disk drive industry: No more MTBF's. In *Proceedings of 2000 Annual Reliability and Maintainability Symposium*, pages 194–199. IEEE, 2000.

[37] J. G. Elerath and S. Shah. Disk drive reliability case study: dependence upon head fly-height and quantity of heads. In *Proceedings of 2003 Annual Reliability and Maintainability Symposium*, pages 608–612. IEEE, 2003.

[38] J. G. Elerath and S. Shah. Server class disk drives: How reliable are they? In *Proceedings of 2004 Annual Reliability and Maintainability Symposium*, pages 151–156. IEEE, 2004.

[39] G. R. Ganger, B. L. Worthington, R. Y. Hou, and Y. N. Patt. Disk subsystem load balancing: Disk striping vs. conventional data placement. In *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences*, volume I, pages 40–49, 1993.

[40] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, NY, Oct. 2003. ACM.

[41] G. A. Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*. PhD thesis, University of California at Berkeley, 1990.

[42] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 92–103, San Jose, CA, Oct. 1998.

[43] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. M. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka. File server scaling with network-attached secure disks. In *Proceedings of the 1997 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Seattle, June 1997. ACM.

[44] G. A. Gibson and R. Van Meter. Network attached storage architecture. *Communications of the ACM*, 43(11):37–45, 2000.

[45] T. J. Glover and M. M. Young. *Pocket PCRef*. Sequoia Publishing, 4th edition, 1994.

[46] B. V. Gnedenko. *Mathematical Methods in Reliability Theory*. Academic Press, New York, Moscow: english translation edition, 1968.

[47] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is not sloth. In *Proceedings*

of the Winter 1995 USENIX Technical Conference, pages 201–212, New Orleans, LA, Jan. 1995. USENIX.

[48] G. Goodson. *Efficient, scalable consistency for highly fault-tolerant storage*. PhD thesis, Carnegie Mellon University, 2004.

[49] E. Grochowski and R. D. Halem. Technological impact of magnetic hard disk drives on storage systems. *IBM Systems Journal*, 42(2):338–346, 2003.

[50] A. Haeberlen, A. Mislove, and P. Druschel. Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005. USENIX.

[51] J. H. Hartman and J. K. Ousterhout. The Zebra striped network file system. *ACM Transactions on Computer Systems*, 13(3):274–310, 1995.

[52] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson. Coding techniques for handling failures in large disk arrays. *Algorithmica*, 12:182–208, 1994.

[53] J. L. Hennessy and D. A. Patterson. *Computer Architecture—A Quantitative Approach*. Morgan Kaufmann Publishers, 3rd edition, 2003.

[54] M. Holland, G. A. Gibson, and D. P. Siewiorek. Architectures and algorithms for on-line failure recovery in redundant disk arrays. *Journal of Parallel and Distributed Databases*, 2(3):795–825, July 1994.

[55] R. J. Honicky and E. L. Miller. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In *Proceedings of the 18th International Parallel & Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, NM, Apr. 2004. IEEE.

[56] A. Hospodor and E. L. Miller. Interconnection architectures for petabyte-scale high-performance storage systems. In *Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 273–281, College Park, MD, Apr. 2004.

[57] R. Y. Hou and Y. N. Patt. Using non-volatile storage to improve the reliability of RAID5 disk arrays. In *Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97)*, pages 206–215, 1997.

[58] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan. Improved disk-drive failure warnings. *IEEE Transactions on Reliability*, 51(3):350–357, 2002.

[59] K. Hwang, H. Jin, and R. Ho. RAID-x: A new distributed disk array for I/O-centric cluster computing. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 279–286, Aug. 2000.

[60] IBM Corporation. Storage consolidation for large workgroups and departments: An IBM SAN business value solution, 2002.

[61] http://www.archive.org/web/researcher/data_available.php.

[62] M. Ji, A. Veitch, and J. Wilkes. Seneca: Remote mirroring done write. In *Proceedings of the 2003 USENIX Annual Technical Conference*. USENIX, June 2003.

[63] T. Joyce. NAS gateways allow IP access to SANs. Network World Fusion, Apr. 2004.

[64] R. W. Kembel. *Fibre Channel: A Comprehensive Introduction*. Northwest Learning Associates, Inc., 2000.

[65] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Cambridge, MA, Nov. 2000. ACM.

[66] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 84–92, Cambridge, MA, 1996.

[67] W. Litwin, J. Menon, and T. Risch. LH* schemes with scalable availability. Technical Report RJ 10121 (91937), IBM Research, Almaden Center, May 1998.

[68] W. Litwin, M. Neimat, G. Levy, S. Ndiaye, T. Seck, and T. Schwarz. LH*$_S$: a high-availability and high-security scalable distributed data structure. In *Proceedings of the 7th International Workshop on Research Issues in Data Engineering, 1997*, pages 141–150, Birmingham, UK, Apr. 1997. IEEE.

[69] W. Litwin and M.-A. Neimat. High-availability LH* schemes with mirroring. In *Proceedings of the Conference on Cooperative Information Systems*, pages 196–205, 1996.

[70] W. Litwin and T. Schwarz. LH*$_{RS}$: A high-availability scalable distributed data structure using Reed Solomon codes. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 237–248, Dallas, TX, May 2000. ACM.

[71] W. Litwin and T. Schwarz. Algebraic signatures for scalable distributed data structures. Technical Report CERIA Technical Report, Université Paris 9 Dauphine, Sept. 2002.

[72] D. D. E. Long. A technique for managing mirrored disks. In *Proceedings of the 20th IEEE International Performance, Computing and Communications Conference (IPCCC '01)*, pages 272–277, Phoenix, Apr. 2001. IEEE.

[73] D. D. E. Long and L.-F. Cabrera. Exploiting multiple I/O streams to provide high data-rates. In *Proceedings of the Summer 1991 USENIX Technical Conference*, pages 31–48, Nashville, June 1991. USENIX.

[74] P. Lyman, H. R. Varian, P. Charles, N. Good, L. L. Jordan, and J. Pal. How much information? 2003. http://www.sims.berkeley.edu/research/projects/how-much-info-2003/.

[75] J. Menon and D. Mattson. Distributed sparing in disk arrays. In *Proceedings of Compcon '92*, pages 410–416, Feb. 1992.

[76] M. Mesnier, G. R. Ganger, and E. Riedel. Object-based storage. *IEEE Communications Magazine*, 41(8), Aug. 2003.

[77] R. A. Meyer and R. Bagrodia. PARSEC user manual, release 1.1. http://pcl.cs.ucla.edu/projects/parsec/.

[78] E. L. Miller and R. H. Katz. RAMA: A file system for massively-parallel computers. In *Proceedings of the 12th IEEE Symposium on Mass Storage Systems*, pages 163–168, Apr. 1993.

[79] E. L. Miller and R. H. Katz. RAMA: An easy-to-use, high-performance parallel file system. *Parallel Computing*, 23(4):419–446, 1997.

[80] MIT RON (Resilient Overlay Networks) Project. http://nms.lcs.mit.edu/ron/.

[81] C. Mohan and I. Narang. Recovery and coherency-control protocols for fast intersystem page transfer and fine-granularity locking in a shared disks transaction environment. In *Proceedings of the 17th Conference on Very Large Databases (VLDB)*, pages 193–207, 1991.

[82] P. Mohapatra. Wormhole routing techniques for directly connected multicomputer systems. *ACM Computing Surveys*, 30(3):374–410, 1998.

[83] R. R. Muntz and J. C. S. Lui. Performance analysis of disk arrays under failure. In *Proceedings of the 16th Conference on Very Large Databases (VLDB)*, pages 162–173, 1990.

[84] J. Muppala, M. Malhotra, and K. S. Trivedi. Markov dependability models of complex systems. In S. Ozekici, editor, *Reliability and Maintenance of Complex Systems*, pages 442–486. Springer Verlag, Berlin, 1996.

[85] Network Appliance Inc. Network Appliance fibre channel SAN storage solutions.

[86] S. W. Ng. Crosshatch disk array for improved reliability and performance. In *Proceedings of the 21st International Symposium on Computer Architecture*, pages 255–264, Chicago, IL, 1994. ACM.

[87] D. A. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, and N. Treuhaft. Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and

Case Studies. Technical Report UCB//CSD-02-1175, University of California, Berkeley, Mar. 2002.

[88] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software—Practice and Experience (SPE)*, 27(9):995–1012, Sept. 1997. Correction in James S. Plank and Ying Ding, Technical Report UT-CS-03-504, U Tennessee, 2003.

[89] D. K. Pradhan. *Fault-Tolerant Computer System Design*. Prentice Hall, Inc., 1996.

[90] K. W. Preslan, A. Barry, J. Brassow, M. Declerck, A. J. Lewis, A. Manthei, B. Marzinski, E. Nygaard, S. V. Oort, D. Teigland, M. Tilstra, S. Whitehouse, and M. O'Keefe. Scalability and failure recovery in a Linux cluster file system. In *Proceedings of the 4th Annual Linux Showcase and Conference*, Oct. 2000.

[91] W. C. Preston. *Using SANs and NAS*. O'REILLY, 2002.

[92] W. Rash. Downtime Is Expensive: It Could Cost You Your Business. Internet Week, Nov. 2000.

[93] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: the OceanStore prototype. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST)*, pages 1–14, Mar. 2003.

[94] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, pages 40–49, Sept. 2001.

[95] M. Ripeanu, A. Iamnitchi, and I. Foster. Mapping the Gnutella network. *IEEE Internet Computing*, 6(1):50–57, Aug. 2002.

[96] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 188–201, Banff, Canada, Oct. 2001. ACM.

[97] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. FAB: Building distributed enterprise disk arrays from commodity components. pages 48–58, 2004.

[98] Y. Saito and C. Karamanolis. Pangaea: A symbiotic wide-area file system. In *Proceedings of the 2002 ACM SIGOPS European Workshop*. ACM, Sept. 2002.

[99] Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming aggressive replication in the Pangaea wide-area file system. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, Dec. 2002.

[100] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pages 231–244. USENIX, Jan. 2002.

[101] M. Schulze, G. Gibson, R. Katz, and D. Patterson. How reliable is a RAID? In *Proceedings of Compcon '89*, pages 118–123. IEEE, Mar. 1989.

[102] T. J. Schwarz. Generalized Reed Solomon codes for erasure correction in SDDS. In *Workshop on Distributed Data and Structures (WDAS 2002)*, Paris, Mar. 2002.

[103] T. J. Schwarz and W. A. Burkhard. Reliability and performance of RAIDs. *IEEE Transactions on Magnetics*, 31(2):1161–1166, 1995.

[104] T. J. Schwarz, Q. Xin, E. L. Miller, D. D. E. Long, A. Hospodor, and S. Ng. Disk scrubbing in large archival storage systems. In *Proceedings of the 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '04)*, pages 409–418. IEEE, Oct. 2004.

[105] S. Shah and J. G. Elerath. Disk drive vintage and its effect on reliability. In *Proceedings of 2004 Annual Reliability and Maintainability Symposium*, pages 163–165. IEEE, 2004.

[106] D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems Design and Evaluation*. The Digital Press, 2nd edition, 1992.

[107] D. J. Smith. *Reliability Engineering*. Harper and Row Publishers, Inc. Barns and Noble Import Division, New York, 1972.

[108] M. Stonebraker and G. A. Schloss. Distributed RAID—a new multiple copy algorithm. In

Proceedings of the 6th International Conference on Data Engineering (ICDE '90), pages 430–437, Feb. 1990.

[109] Sun Microsystems. High availability fundamentals. in Sun BluePrint Online, 2000.

[110] A. Takefusa, O. Tatebe, S. Matsuoka, and Y. Morita. Performance analysis of scheduling and replication algorithms on grid datafarm architecture for high energy physics applications. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 34–43, Seattle, WA, June 2003. IEEE.

[111] N. Talagala and D. Patterson. An analysis of error behaviour in a large storage system. In *Workshop on Fault Tolerance in Parallel and Distributed Systems*, 1999.

[112] The International Disk Drive Equipment & Materials Association (IDEMA). R2-98: Specification of hard disk drive reliability.

[113] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A scalable distributed file system. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP '97)*, pages 224–237, 1997.

[114] K. S. Trivedi. Software aging and rejuvenation. Keynote *at the Workshop on Self-Healing, Adaptive and Self-Managed Systems (SHAMAN 2002), New York City, NY, June 2002*.

[115] K. S. Trivedi. *Probability & Statistics with Reliability, Queuing and Computer Science Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[116] K. S. Trivedi. *Probability & Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley & Sons, 2nd edition, 2002.

[117] K. S. Trivedi and V. Kulkarni. FSPNs: Fluid stochastic petri nets. In *Proceeding of 14th International Conference on Applications and Theory of Petri Nets*, pages 24–31, 1993.

[118] A. S. Vaidya, C. R. Das, and A. Sivasubramaniam. A testbed for evaluation of fault-tolerant routing in multiprocessor interconnection networks. *IEEE Transactions on Parallel and Distributed Systems*, 10(10):1052–1066, 1999.

[119] N. H. Vaidya. A case for two-level distributed recovery schemes. *ACM SIGMETRICS Performance Evaluation Review*, 23(1):64–73, 1995.

[120] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. McLarty. File system workload analysis for large scale scientific computing applications. In *Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 139–152, College Park, MD, Apr. 2004.

[121] R. Y. Wang and T. E. Anderson. xFS: A wide area mass storage file system. In *Proceeding of Workshop on Workstation Operating Systems*, pages 71–78, 1993.

[122] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, Cambridge, Massachusetts, Mar. 2002.

[123] R. O. Weber. Information technology—SCSI object-based storage device commands (OSD). Technical Council Proposal Document T10/1355-D, Technical Committee T10, Aug. 2002.

[124] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller. Dynamic metadata management for petabyte-scale file systems. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04)*, Pittsburgh, PA, Nov. 2004. ACM.

[125] M. Wiesmann, A. S. F. Pedone, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS '00)*, pages 464–474, 2000.

[126] W. W. Wilcke, R. B. G. H. Huels, and C. Fleiner. IceCube – a system architecture for storage and Internet servers. IBM Corporation, http://www.almaden.ibm.com/StorageSystems/autonomic_storage/CIB_Hardware/, 2002.

[127] K. H. Yeung and T. S. Yum. Dynamic multiple parity (DMP) disk array for serial transaction processing. *IEEE Transactions on Computers*, 50(9):949–959, 2001.

[128] C. Zhang, X. Yu, A. Krishnamurthy, and R. Y. Wang. Configuring and scheduling an eager-

writing disk array for a transaction processing workload. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, 2002.

[129] L. Zhang. Fault tolerant networks with small degree. In *Proceedings of the 12th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 65–69. ACM, 2000.