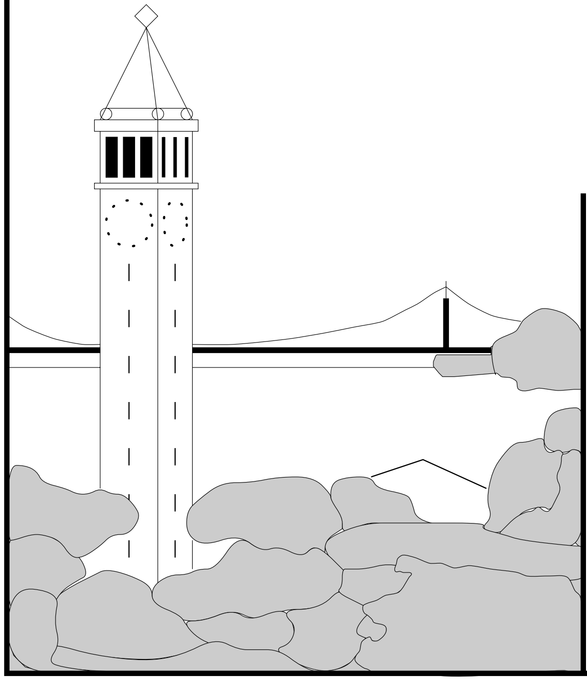
Reducing the Cost of System Administration of a Disk Storage System Built from Commodity Components

Satoshi Asami



Report No. UCB/CSD-00-1100

May 2000

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Reducing the Cost of System Administration of a Disk Storage System Built from Commodity Components

by

Satoshi Asami

B.S. (University of Tokyo) 1989
M.S. (University of Tokyo) 1991

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

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor David A. Patterson, Chair
Professor Robert Wilensky
Professor J. George Shanthikumar

2000

The dissertation of Satoshi Asami is approved:

Chair Date

Date

Date

University of California at Berkeley

2000

Reducing the Cost of System Administration of a Disk Storage System Built from Commodity Components

Abstract

Reducing the Cost of System Administration of a Disk Storage System Built from Commodity Components

by

Satoshi Asami
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor David A. Patterson, Chair

This dissertation explores how to reduce the system administration cost of disk storage systems. There are several reasons why reducing the operator's burden is the key to success of large storage systems. One is that the cost of system administration usually dominates the budget of storage systems. Another is that an operator error on storage systems can easily have disastrous results. In the field of physiology and psychology, there have been studies that show reducing mental and physical stress on the operator is crucial in preventing human errors.

This dissertation describes Tertiary Disk, a large-scale disk array system built from commodity components, and how we evaluated the feasibility of its design. Instead of incurring the cost of custom hardware, we attempt to solve various problems by design and software. Tertiary Disk is a cluster of storage nodes connected by switched Ethernet. Each storage node is a PC hosting a few dozen SCSI disks, running the FreeBSD operating system. The system is used as a web-based image server for the Zoom Project in cooperation with the Fine Arts Museums of San Francisco. Our system is fully redundant in both hardware and software, and is designed to avoid a single point of failure.

There are several approaches to lower the human cost of system administration. One is to make the system as autonomous as possible. I have designed a self-maintenance extension to the operating system to make the system run continuously in the event of failures. There are also several other improvements to the system to make the operator's job easier.

Finally, we will prove the feasibility of our system by evaluating it by simulation. Failure data that has been collected on Tertiary Disk over the course of several years were used to design an event generator. The second program, a simulator, models the system using a directed acyclic graph and computes its availability by solving a connectivity problem. The results have shown that our system performs as expected with the current set of parameters, and also expands nicely into the future.

Professor David A. Patterson
Dissertation Committee Chair

Dedication

To my parents—I wouldn’t be here without you.

Contents

List of Figures

List of Tables

Acknowledgements

During my years in graduate school many people have helped me professionally and personally. Foremost among those people is my advisor, David Patterson. He has taught me to look at the big picture and work hard. He has also taught be the value of patience and exercised it in guiding me through the program. I am extremely lucky to have had such an understanding and inspirational advisor.

I would also like to thank the other readers of this thesis, Robert Wilensky and J. George Shanthikumar, for their detailed and thoughtful comments. Robert also helped me find a new loving home for the images, so I can leave the university with a peace of mind. George's lectures were always theoretically sound while also down to the details, which I enjoyed immensely.

The other professors involved with the Network of Workstations (NOW) projects deserve my special thanks: Tom Anderson and David Culler. They have skillfully guided the toddling Tertiary Disk project in its infancy. Jim Gray of Microsoft was the instigator of the Tertiary Disk project. The project would not have existed without him.

The people at the Fine Arts Museums at San Francisco gave us a wonderful collection of pictures to store in our (initially empty) storage system. As they say in Japan, a statue is worthless without a heart. I would especially like to thank Bob Futernick, Dakin Hart and Sue Grinols for entrusting us with their precious pictures and helping us understand what it takes to handle fine art objects. Cal students Victor Wong, Tony Le, Wen-Kai Zhong, Gabriela Hernandez, Nicholas Huynh, Lila Tretikov, Abby Thompson and Shankara Lowe helped us prepare the images for public viewing.

I would also like to thank the many professors who have made my time here at Berkeley enjoyable with great classes. Randy Katz's VLSI design course was the best class I have taken in any level. Jim Pitman and Max Mendel helped me understand the basics and applications of statistics, with many outrageous examples which I will never forget. Richard Newton opened my eyes to finer points of computer design. Larry Rowe taught me, among other things, that a good user interface is just as important as the internals of the system itself. Eugene Lawler showed me the value of theoretical thinking. I wish he could be at the commencement to share the moment with us.

Terry Lessard-Smith, Bob Miller, Kathryn Crabtree and Peggy Lau also deserve special thanks for straightening out the many problems I had. Eric Fraser was very helpful in taking care of the machines during the project's early days, while Kevin Mullally was the one who first taught me the importance of system administration.

The people in the FreeBSD project were very helpful in providing fixes and workarounds to the problems we encountered. Justin Gibbs and Ken Merry deserve special thanks for their work on the SCSI system, Stefan Esser for his PCI bridging support, and Bruce Evans for his guidance through the kernel interface and build process.

I enjoyed sharing 477 Soda with a lot of people, including Nisha Talagala and Remzi Arpaci-Dusseau. Nisha has also been my project partner and was a great help in, among other wonderful things, breaking many disks. Remzi supplied the often dry room with a good sense of humor as well as a constant stream of soft drinks. Eric Anderson, like a good neighbor, was always around when I needed a little expert advice.

My friends May Cheng, Ritsuko Nakamura and Matt O'Hara helped me get through difficult times. I don't know where I would be without them. My parents, Fumi and Susumu Asami, have been very supportive throughout my entire life and have always believed in me. They have given their only son an opportunity to pursue his dreams on a foreign land. I am happy to be able to show them that I can finish what I started and hope that I have brought honor to the family name for them.

Last but not least, my wife Margaret Asami has been my best supporter in my last few years. No matter how busy she is at work, she always takes the time to listen, encourage, cheer me up and clean up after dinner, while enduring the many long nights I had to go through. I simply can’t thank her enough.

This research was funded by DARPA Roboline grant N00600-93-K-2481 and the State of California MICRO program. I would also like to thank IBM and Intel for donating disks and PCs, respectively, to help build the Tertiary Disk cluster.

Chapter 1

Introduction

Ten years ago, large disk storage systems were built as specialized, large-iron server systems. They combined the fastest central processor units (CPU) available on workstations with special buses and expensive memory systems. Server price tags were much higher than those of the average workstation. People were still content since all computers were expensive and they needed only one of these servers for dozens of workstations.

Since then, prices of computer parts have been dropping rapidly, especially in the past few years. For instance, the cost of disk per unit capacity, usually measured in dollars or cents per megabyte, has been steadily dropping at the rate of 60–100% per year in since 1991. Decreases in cost of semiconductor memory, CPU, and other parts are similarly significant.

1.1 Have Computers Really Become Cheaper?

However, the prices of computers haven't really gone down—we now simply have access to much better components for the same price. Ten years ago, you could get a decent computer for $1,000, and you can still get a decent computer for $1,000 today; the only thing that changed is the meaning of the the word "decent". Table 1.1 shows a comparison of a better-than-average PC that you can purchase for $2,000 in late 1991 and late 1999. The 1991 PC was the first PC I purchased when I came to the United States. There are some PCs now that sell much cheaper than $2,000, but they are not sufficient as building blocks for storage systems so I decided to not list them here.

To summarize, we can purchase a PC that has 32 times as much main memory, 85 times as much disk space and 128 times as much video memory for about the same price as we did eight years ago. The monitor is now capable of redrawing more than twice as many scan lines per second. Comparing the speed of computers is harder, as there are many factors that affect the *real* performance—for instance, you can't just say that the 600MHz Pentium Ⅲ in 1999 is 18 times faster than the 33MHz 80386 in 1991 just by comparing the clock rates, as they have different registers, instruction sets, cycles per instruction, and so on [HP96a]. However, from the numbers listed in Table 1.1 we can see, for instance, that the main system bus is capable of transferring eight times as much data per second and the disk sequential transfer speed has improved by a factor of 30. The improvement in speed of individual components appear to be by a factor of around 10 to 30 times.

It is also notable that the disk subsystem has improved significantly. The individual disks have increased their size by a factor of two magnitudes, and the data transfer rate of the bus has

Subsystem		Specification	
Category	Name	1991	1999
CPU	Class	80386	Pentium III
	Internal clock rate	33MHz	600MHz
	External clock rate	33MHz	100MHz
	External bus width	32 bits	64 bits
Cache	On-chip	(none)	32KB
	Off-chip	64KB	512KB
Main memory	Type	DRAM	SDRAM
	Size	4MB	128MB
System bus	Type	ISA	PCI
	Clock rate	8.3MHz	33MHz
	Width	16 bits	32 bits
Disk interface	Type	IDE	SCSI
	Bus width	8 bits	16 bits
	Max. transfer speed	3MB/s	160MB/s
	Devices per bus	2	15
Hard disk	Size	0.12GB	10.2GB
	Sequential access speed	0.3MB/s	10MB/s
	Average seek time	15ms	7ms
Video card	Video RAM size	0.25MB	32MB
Monitor	Screen size	14 inch	17 inch
	Max. horizontal sync	36kHz	82kHz

Table 1.1: Comparison of a $2,000 PC in 1991 and 1999

increased by a factor of 50. It is also now possible to attach more disks to a single bus.

1.1.1 Building Large Storage Systems

Given this new situation, it appears it is now feasible to build very large disk storage systems using commodity hardware. The PCI (Peripheral Component Interconnect) bus can transfer up to 132MB/s in burst mode and can be chained by PCI expansion boxes to an arbitrary depth to connect dozens of expansion cards to a single CPU [Sha95]. The SCSI (Small Computer Systems Interface) bus can connect up to 15 devices to a single host adapter and transfer 160MB/s total [X3T]. Combined with the expandability of the PCI bus, it is now possible to connect hundreds of disks to a single PC without using any specialized equipment. Although still behind in floating-point operations, today's PCs have enough integer processing power to put to shame server systems of a decade ago. The Tertiary Disk project, which I will explain in detail in Chapter 3, has built one such system entirely from commodity components [TAAP98].

1.1.2 System Administration Cost

On the other hand, the cost of administering storage systems has remained high over the years. For instance, a 1993 study by Strategic Research Corp. shows the annual cost of system administration of storage systems is almost three times that of the cost of system hardware itself [SR93]. Without significantly reducing the cost of system administration, any benefit of using commodity hardware to build storage systems will quickly diminish, especially if we are to build a system with a large number of disks to take advantage of the new interfaces, instead of purchasing an expensive turnkey solution like a hardware RAID box.

1.1.3 Our Approach

Our solution to this challenge is to make the storage system *self-maintaining* [ATP99]. Instead of having someone constantly look after the system, possibly with an around-the-clock pager as seen on some installations, our system is designed to mask problems and run continuously even when there are several independent component failures. The system administrator's job is only to come in for scheduled visits, for instance, once per week, to replace the failed components. Such mode of operation will also reduce the possibility of human errors, as people working during normal hours are less likely to commit errors than those who are paged and have to drive to work in the middle of the night. Indeed, Gray found that operator errors are as plentiful as hardware errors, so this scheme will improve availability as well [Gra90]. We also propose another possibility, with an operator with a pager but without the obligation to come in immediately—if the page arrives after normal work hours, that person can just show up next morning at 9 A.M. We will show that, with our design, such mode of operation will result in very good system availability, without adding significantly to the cost.

The main application of our system is a web server holding image data for objects of art. In that context, we have decided to adopt a policy called "repair by reload". For any single failure, our goal is to mask the failure within five seconds so that a retry by the client will succeed. The reason we picked five seconds is to make sure it is short enough so that if we can mask the problem within that time period, users will get the correct web page when they get impatient and hit the `reload` button on their browsers. Because of the nature of the Internet, such short-term failures are likely to be viewed as transient problems on the network users experience every day, rather than a problem on our end. It is virtually impossible for users to distinguish between intermittent network problems and very short server outages.

In our application, end users will not issue any writes, which also simplifies our task. We have used the approach to mirror all the data that is required for day-to-day operation. The design issue then simply becomes how to quickly detect failures and reroute user requests to the backup copy.

In addition to making the system self-maintaining, I have explored several other methods to reduce the cost of system administration. For instance, one important aspect of a self-maintaining system is that it can reboot quickly—the longer it takes to reboot, the larger the window of vulnerability and the less chance it will continue running without human intervention. By combining several techniques, I was able to reduce the boot time from 22 minutes to about 1 minute.

1.1.4 Applicability

Although some aspects of our design benefits from the read-mostly nature of the application and our position as a web server, we are hardly alone in providing such service. There are literally hundreds of large web sites on the Internet that specialize in read-mostly information contents, as well as more traditional anonymous ftp servers and gopher servers. To extend the horizon a little, the read-mostly nature is common in many other types of large storage systems such as certain kinds of database servers as well.

1.2 Outline of Dissertation

This section outlines the thesis and lists its overall results by chapter. The rest of this chapter lists the related work in the field. Chapter 2 expands on this chapter and illustrates the motivation behind the research.

The architecture of the system we are using for the experiment and the main application is illustrated in Chapter 3. The research was conducted on a cluster of PCs built by the Tertiary Disk group [TAAP98]. The system consists of 24 PCs and 364 8GB disks for a total raw capacity of 3.2TB. Four of the machines are various frontend and infrastructure servers; the disks are connected to the other 20. The disk servers are in two different configurations; the “disk-heavy” set with 35 disks per machine, and a “disk-light” set that has 16 disks per machine.

All the machines run the FreeBSD operating system [FB93]. I have been working with the developers of the FreeBSD project who have been helpful in fixing bugs and making enhancements to the system to make it suitable for use as a large-scale storage system. We explain and evaluate the applicability of such improvements to other systems in Chapter 3. They are also compared to similar methods proposed elsewhere.

Our main application is an art image database server. Cooperating with the Fine Arts Museums of San Francisco, we have been offering pictures of objects of art through their search engine [TAP$^+$98]. Our site holds large versions of the images that do not fit on their disk.

In Chapter 4, we clarify the concept of self-maintainability and explain how we approach the problem. The word “self-maintaining” has different meanings depending on the point of view of the two groups that interact with storage systems: users and system administrators. Users do not know what is going on inside the system; they just expect the system to be "up" at all time and will get easily annoyed if it isn't. Depending on the application, there are various types of users, with different requirements. For a public service site such as ours, it is essential to keep user annoyance to a minimum as unhappy users may never come back.

The system administrators are generally more understanding than the end user to the problem of the system—some of them are even architects of the systems themselves—but the amount of the time these people have to spend directly affects the cost of keeping the system up and running. Also, the more pressure you put on this person, the more likely there will be human errors, and human errors by system administrators can have disastrous results. Thus, it is very important to ease the job of the system administrator through whatever means possible.

We validated our system using a simulator. There were three steps in our validation methodology: collecting data, running simulation and evaluation. The first step is explained in Chapter 5, while the other two are detailed in Chapter 6. The data we needed to collect fell in two categories. One is failure information: the symptoms, error frequencies, and so on. The other is repairing information, i.e., how long it takes to conduct a repair on a certain component.

After analyzing the data, they were fed to an event-driven simulator to predict the behavior of the system over a longer period of time. The simulator builds a simple directed acyclic graph (DAG) to represent the system and checks the connectivity whenever there is a change to one of the components' state. We also changed some parameters to explore the design space. The parameters include numerical ones, such as repair interval and degree of redundancy, and qualitative ones such as the importance of double-ending. The results were compared with data available for other systems to prove the validity of our design.

Chapter 7 summarizes the results and observes future prospects of the research topic. This chapter also gives an insight on what we would have done differently if we were to build a similar system again.

1.3 Related Work

I have listed related work in the field of fast recovery, system administration of clustered systems, in particular monitoring and diagnosis, and large disk storage systems.

1.3.1 Disk Storage Systems

There are three classes of disk storage systems that are related to my research; hardware RAID, high-availability server systems and network-attached storage systems.

Hardware RAID

RAID (Redundant Array of Inexpensive Disks) is a method of building large disk storage systems from smaller disks [CGP$^+$88, Gib92]. There are six levels of RAIDs, 0 through 5, differing in redundancy schemes. Most widely used are RAID-0, which is simple striping with no redundancy, RAID-1, which is mirroring, and RAID-5, which is block-interleaved parity without a dedicated parity disk. There are many companies—EMC, Sony, IBM, BoxHill, StorageTek, Sun, DEC, to name a few—selling RAID-based systems on the market. There are more than 50 companies and 100 products in the RAID Consortium, and collectively they ship billions of dollars of systems each year.

The problems with current RAID systems are twofold. One is price, in that custom hardware required to build these boxes cost much more than commodity components as we are using. The other is expandability. These boxes have a fixed upper limit on number of disks, usually topping out at several dozen to a few hundred gigabytes, and will force the user to purchase another box when the limit is reached. However, increasing the number of RAID boxes opens up a whole new set of problems, as the reliability decreases linearly with the number of boxes while the maintenance cost and complexity increases.

As mentioned throughout this thesis, these are the problems we are trying to solve with our system.

High-Availability Servers

Many vendors offer complete server systems with a large storage capacity. Tandem has a long history in this field. Their NonStop servers are fully redundant and have hot-swappability of most components, thus it is not necessary to power it down even during repairs [BBC+90]. Their architecture, called "shared nothing", has no single point of failure, i.e., any one component can fail and the system will still function. Jim Gray did a study on their availability in the late 1980's [Gra90].

Their software uses a design called "process pairs" in which a primary process, doing all the work during normal operation, periodically sends synchronization messages to a backup process [Bar81, Tan97a]. When the primary process, or the hardware that supports it, fails, the backup process will take over. Since the backup process does not have to doing any heavy processing, there is very little overhead during normal operation. This design is similar to the way our frontend servers back up each other on a peer-to-peer basis.

Their system administration suite is called TMDS (Tandem Maintenance and Diagnostic System) [Tan97b]. It has an auto-diagnostics feature that uses an artificial intelligence program based on past knowledge, which compares the symptoms of the system to known failure modes and tries to identify the failures. TMDS can optionally dial up Tandem's technical support center with a report which Tandem technicians can consult while running remote diagnostics of their own.

Network-Attached Storage

Network Appliance [NA92] sells network servers built around Digital's Alpha chip. Their NFS servers are advertised to outperform a 4-way P6-200MHz Windows NT system by 2 to 10 times. They use RAID-4, block-interleaved parity with a dedicated parity disk, with NVRAMs to increase performance. They have a filesystem that can recover from crashes very quickly by using checkpointing and roll-forward logs.

Microsoft Tiger is a video server built from commodity PCs which they call "cubs" [BBD+96, BFD97]. Their goal is to tolerate the failure of any one cub or disk without noticeable degradation of service. They use mirroring for backing up data, since they cannot tolerate the runtime reconstruction overhead of parity-based redundancy schemes. They place the "primary" copies of the data on the outer tracks of the disks for better performance, and the backup copies are declustered to avoid loading a single machine or disk during failures. Tiger distributes all files across all disks on all cubs for maximum striping performance, but this method has a drawback of having to reconstruct data on the entire system when a new disk is added. This reconstruction is not very expensive as there is no parity calculation involved, but the system has to be shut down for a few hours while the reconstruction takes place.

A single controller, which has the only IP address known from outside of the cluster, serves as the entry point from clients. No image data passes through the controller so it is not likely that they will become performance bottlenecks. This design, with a machine with a single IP address known to the outside world serving as a frontend that does not handle data, is similar to our system, except we have two frontend machines backing up each other for extra availability.

They take great care to ensure that sufficient bandwidth is available during the entire course of the playback, and will delay start of playback if necessary. They use a distributed schedule management protocol for scalability. Each cub has a partial, potentially outdated view of the schedule which they pass around, updating it along the way. Cubs use deadman protocol for fault detection—each cub sends periodic ping to the cub on its right.

Petal is another network-attached storage system [LT96]. Petal is a collection of distributed servers, each containing multiple disks. They use a method called *chained declustering* to avoid having the load increase 100% on a machine when its mirrored counterpart fails [HD89]. Petal is a block server. The authors of Petal have also designed a distributed file system called Frangipani to run on top of Petal [TML97].

CMU's Network Attached Secure Disks (NASD) is another example of network attached storage [GNA$^+$96, GNA$^+$98]. Their disks have more intelligence than current disks, and their focus is on security and safety of data.

A final sample is xFS, a combination of network striping and a Log-structured Filesystem (LFS [SBMS85]), which is a method of reducing the disk latency on random writes [ADN$^+$95].

Our system avoids the complexity of distributed filesystems by using HTTP redirects and IP masquerading to distribute user requests and mask failures. In addition to Tiger, this approach is also similar to the initial method employed by the Inktomi project at UC Berkeley, which was a large search engine built from a small cluster of Sun workstations.

There have been many studies that express complex systems using a graph. For example, Ren and Dugan proposed a method to dynamically trim fault trees to analyze system behavior efficiently [RD98]

1.3.2 Fast Recovery

Mary Baker has studied fast reboot of systems after a crash for her Ph.D. thesis [Bak93]. Most of her work is about distributed state recovery of the Sprite operating system, but she has also done considerable amount of work to streamline the normal UNIX start-up procedure to improve the reboot time. Her thesis is that a system that recovers very quickly can be as available as a system that crashes less frequently but recover more slowly. I have made similar improvements to our boot process as can be found in section 3.3.2.

1.3.3 Monitoring and Diagnosis

Eric Anderson of the Network of Workstations (NOW [ACPtNT95]) Project has been studying several aspects of system administration of clusters [And97]. His most recent work introduces a system called CARD (Cluster Administration using Relational Databases) [AP97]. He proposes using relational databases to build an extensible monitoring system. CARD uses a hybrid push-pull protocol to collect data from individual machines. His emphasis is on monitoring and diagnosis.

Swatch [HA93] is a push system, in which each machine sends a periodic update to a centralized server. Swatch modifies UNIX programs to emit more detailed logs, especially pertaining security information. It then uses `syslog` to collect the information and send them to the server. They provide a collection of `perl` scripts to filter the logs to make it human-readable. Swatch also allows administrators to define their own functions using scripts to focus on some parts more carefully.

TkIned [Sch95] is another centralized system. TkIned has an extensive collection of methods for gathering data. Because it is distributed with complete source code, it can be extended by modifying the program. Since the data is not accessible outside of the TkIned program, new modules either have to be added to TkIned, or have to repeat the data gathering. TkIned provides simple support for visualization and does not aggregate data before displaying it. TkIned's centralized pull model limits its scalability.

Pulsar [Fin97] uses *pulse monitors*—small Tcl/Tk [Ous94] scripts–to collect information about systems. If the measured value is not within the predefined bounds, a report is sent to a central server. Pulsar is extensible but not fault tolerant due to its centralized design. We are using Tcl/Tk and its extension Expect for some of our system administration tools.

Nisha Talagala of the Tertiary Disk Project has studied the failures we encountered on our system as part of her Ph.D. thesis [Tal99]. She has concentrated on six months of the logs from the system and has analyzed them in detail. I have done a similar analysis in Chapter 5 for a longer period of time and a different emphasis as a basis of my simulation.

SCSI-3 has a new feature called SCSI Environmental Services (SES). It can be used to monitor information such as status of enclosure power supply or fan and temperature of various components, as well as send commands to control devices such as enclosure buzzers and LEDs. It would have been useful for our project if it were available when we built the prototype.

1.3.4 Self-Management of Storage

Most of the literature that deal with reducing the cost of system administration of storage accomplish the goal by automated management of storage devices. Although the emphasis of this thesis is not in that aspect of system administration, some of the most prominent projects are mentioned here as we share the common goal of reducing the system administrator's burden.

The Storage Systems Program (SSP) at HP is working on a system called "self-management" of storage [BGM$^+$96]. Their focus is on automatic assignment of storage devices; humans do not have to worry where to put what. Their prototype system is capable of assigning several thousands of objects to devices in a few minutes.

Sun's Jini makes devices, including disks, identify themselves as part of their automatic component discovery paradigm [Wal98]. The devices either have enough intelligence built in them to be able to identify themselves to the clients, or they will download the drivers from a server. There is also a recent project in Sun called Jiro, which defines a storage management interface to let components manufactured by different vendors cooperate to provide an easily manageable environment. Their draft is currently undergoing a review every two months or so [Jir99].

MicroSoft's Windows 95/98 have an "autorun" feature on CD-ROM drives, in which a special file is checked when a CD is inserted and is run automatically when it is found. This mechanism is similar to our "script" approach of setting up disks mentioned in Section 3.3.2.

1.4 Research Issues/Contributions

To summarize, these are the contributions of this research:

- Analyzing and evaluating various methods to reduce system administration cost of large storage systems
- Designing and implementing a self-maintaining storage system
- Several small but helpful enhancements to the operating system to make the system easier to manage
- A guide to designing systems that recover quickly
- Cataloging and analyzing the component failures we have seen over two years the prototype has been in operation
- An evaluation of the reliability of our design by simulation and exploration of the design space
- Insights into the relationship between system availability and the reliability of a single disk
- Insights into the relationship between system availability and the frequency of visits for repair
- Use of DAG to represent system to simplify simulation of availability
- A list of concerns when building large-scale storage systems from commodity components

Chapter 2

Motivation

This chapter explains further why reducing the burden on the system administrator is important for storage systems. The point we would like to emphasize is that the system will be cheaper, safer, and more available if it can mostly manage itself rather than rely on being watched by a human operator.

2.1 Overview

Most storage systems today are *ad hoc* collection of components, growing over the years as the organization's needs increase. The two most common methods for protection against data loss are backups and redundancy. I will briefly overview them here, focusing on the system administrator's point of view. The type of failures they protect the system from is also illustrated.

2.1.1 Backups

Backups are typically done to magnetic tapes. Tapes have very low unit cost, moderate bandwidth and slow random access. These characteristics make them ideal for backups since the backup tapes are usually written in one pass and almost never read back.

The reasoning behind doing backups are twofold. One is to safeguard against hardware failures that result in data loss. The other is to help recover data from human errors.

Hardware Failures

If the storage system suffers a crash, the operator can recover the filesystem from backup tapes. In some organizations, the range of failures to safeguard against also includes catastrophic disasters—the only way to prevent losing all data when there is a fire in the computer room is to make a copy of data and send it to a physically remote location. The cheapest way to accomplish this kind of redundancy is to ship full backup tapes to another site periodically.

The system administrator usually does not have to spend much time making backups after the initial setup, although the amount of daily tape-changing that has to be done depends on the characteristic of the tape backup system. For instance, a large organization with a tape library can have fully automated daily backups; a smaller one with a stacker might require an operator to change tapes daily.

Human Errors

Since few filesystems in existence today offer versioning support, even if the storage system functions exactly as designed without any failures, users can still “lose” files because of their own mistakes. By having a backup done periodically, the system administrator can manually recover at least some version of the lost file—if backups are done daily, it will probably be yesterday’s version.

Recovering from human errors could be a much more time-consuming process for the system administrator, as the user might not even know when the file was last written. Given information such as “I think I deleted a file named `handler.c` in this directory last week”, the operator has to go through several tapes trying to find the latest version of the file in question. The situation is further complicated when the system has incremental backups, since yesterday’s version of the file is not necessarily found in yesterday’s daily backup tapes—it could have been written months ago and therefore only show up in the last full backup tape set.

Again, the amount of time the operator has to spend depends on the system. One with a tape library and a good backup management software might make answering above requests a one-line command; others might require the operator to manually go back and forth between the terminal and the tape unit and issue several commands to find the right version of the files.

Note that even with backups, some data loss is inevitable. In particular, there is no way to get back modifications done since the last backup. Also, backups have no effect on system availability in an event of hardware failure except that it will help restart the system quicker. Basically, the system will be down until the operator comes in and restores the filesystem.

2.1.2 Redundancy

There are many ways to build disk storage systems with some degree of redundancy to prevent data loss when hardware failures occur. The most common approach, called RAID (Redundant Array of Inexpensive Disks), is a method of building large disk storage systems from smaller disks. There are six levels of RAIDs, 0 through 5, differing in redundancy schemes. Here are brief descriptions of most commonly used RAID levels.

RAID-0

Level 0 is simple striping. Obviously, this is not “redundant” at all; it is included in RAID levels just as a reference point. RAID-0 is optimal for non-essential data where read/write speed is most important, as this suffers from no overhead due to redundancy-based protection schemes.

RAID-1

Level 1, or *mirroring*, takes data on one set of disks and makes identical copies of them on another set. RAID-1 can tolerate multiple failures as long as one of the disks is alive on all mirror pairs. RAID-1 has a 100% cost penalty as the system can only hold half the data it would without redundancy. There is also a penalty on writes, since all data blocks have to be written on two disks instead of one.

RAID-5

Level 5, block-interleaved parity without a dedicated parity disk, offers a good balance of performance and cost. With N disks in a group, the cost penalty is $(1/N \times 100)$%. Writing to RAID-5 stripes can be very slow if it involves random accesses, since each write has to be done as two reads (original data, original parity) and two writes (new data, new parity). The performance suffers greatly when one of the disks fails, as many operations will now require the system to access $(N - 1)$ disks on an N-disk group. Recovery is time-consuming, as contents of all $(N - 1)$ disks have to be read to reconstruct the failed disk.

In addition to these traditional RAID levels, there are redundancy schemes that combine more than one RAID level, such as RAID 0+1, and those that use a more complex algorithm to protect against multiple disk losses in a single redundancy group, such as P+Q RAID.

There are many manufacturers selling RAID-based systems on the market. There are more than 50 companies and 100 products in the RAID Consortium, and collectively they ship billions of dollars of systems each year.

In addition to hardware RAID, some systems implement RAID in software. They are inherently slower than hardware RAID but require no extra hardware and therefore are much cheaper. Many operating systems have software RAID support, either as a standard feature or an add-on module, sometimes supplied by a third-party vendor.

In summary, RAID will safeguard against certain type of hardware failures, in particular, a certain number of disk failures, but not others. They do not provide protection against human errors in ways backups do. RAID systems have good availability as long as the failures are within their design specs.

Administering RAID systems are generally easier but human operators still need to take steps to help the system recover. Depending on the design of the system and user interface, managing spare disks can be a time-consuming task.

2.2 The Impact of System Administration

There are several reasons why system administration is the key to the success of a large storage system. One is that the cost of system administration usually dominates the budget of storage systems. Without cutting down the system administration cost, it will not be possible to reduce the overall cost of storage systems significantly. Another is that the impact of bad or nonexistent system administration for storage systems will negatively affect the organization in many ways.

2.2.1 Comparison with Hardware Cost

The cost of computer hardware has been dropping steadily for the past few decades. However, the cost of administering storage systems has remained high, as growth in demands for storage has exceeded the capacity increase of a single disk, making the systems more complex than before. For instance, as mentioned earlier, a 1993 study by Strategic Research Corp. shows that the average annual cost of system administration of storage systems is almost three times that of the cost of system hardware itself [SR93]. Amdahl's law suggests that, without significantly reducing the cost of system administration, any benefit of using commodity hardware to build storage systems will completely diminish [HP96b]. For instance, if the system administration cost is three times that of hardware cost, halving the hardware price will only save 12.5% of the overall cost; even if the hardware price can be reduced to zero, the savings will only be 25% for the overall cost.

Note that their study is already targeting fairly expensive storage systems, as most organizations surveyed were using large fileservers or hardware RAID boxes. For low-cost storage systems such as ours, the ratio of system administration against hardware cost is even higher.

2.2.2 The Risk Factor

The importance of system administration transcends that of the salaries of the system administrators, as the system can do only so much to safeguard against operator errors. In general, it is very easy for an operator to make a mistake that will cause a lot of data to be lost or the system to be unavailable for an extended period of time. The “routine” part of the job of an operator, the everyday work that doesn’t involve repairs and other non-standard operations, is generally not very dangerous—it is possible to safeguard against typical mistakes made during those times fairly effectively by normal means such as backups. However, the “emergency” work that needs to be done by system administrators has a much higher risk than normal operations for several reasons.

Vulnerability

The system is already in a vulnerable state when it is being repaired. For instance, consider a RAID-5 volume with a failed disk. If the operator replaces one of the good disks instead of the faulty one and then runs the reconstruction script, the entire dataset could be lost. Another example is recovering a system from backup tapes. If the operator writes to the tapes instead of reading from them, all data in them will be lost irrecoverably.

These may seem like unlikely events, but stranger things happened in real life. For instance, in our fully mirrored system, I managed to lose an entire dataset by replacing the wrong disk during a repair and then running the copy in the wrong direction. Moreover, the point is not about how likely it is for these events to happen—it is that the system is in a vulnerable state, and a simple error by the human operator can cause a lot more damage when the system is down to one last copy of good data.

Familiarity

System administrators don’t get to practice doing these out-of-routine work because of the rare nature of the problems. Operators either follow written manuals or improvise along the way, and both of these methods are prone to mistakes if the person is not familiar with the actions.

Repair Hours

A system administrator who has just been called in by a pager in the middle of the night is more likely to make mistakes that one that conducts repairs only during regular work hours. This lack of sleep was one of the factors that contributed to the egregious disk copying mistake I mentioned earlier.

There have been many research in the field of physiology and psychology on the subject of sleep deprivation and human performance. In Mullaney et al. [MKF83] and Angus and Heslegrave [AH85], researchers confirm our intuition, that people are less likely to perform tasks correctly and efficiently after a long period of work and also during normal sleeping hours.

Hockey et al. [HWS98] conducted a series of tests in which operators were asked to monitor the state of a complex machinery control system and make necessary adjustments to keep the system functional. Other than the expected result in which operators without enough sleep did not perform as well, they found an interesting tendency, in which sleep-deprived subjects tended to exhibit *reactive* behavior rather than perform preventive, model-based strategy in dealing with problems. To quote a sentence from their conclusion, "Operators make more frequent interventions in order to stabilize the system when faults occur, sometimes without a clear idea of what is wrong." It is worth pointing out that this kind of careless reflective actions is very dangerous on a storage system in a vulnerable state, and can easily lead to data loss.

In another study, Fairclough and Graham [FG99] found that sleep deprivation is similar to alcohol consumption in terms of having a detrimental effect on subjects' driving ability, although one notable difference was that sober but sleep-deprived subjects knew that their performance was suffering.

Mental Pressure

The operator will be under mental pressure to repair a system if it is already down and it has to be brought up as soon as possible. Under such mental stress, people are more likely to make mistakes they won't make without pressure.

Psychologists have been studying the effects of human presences for over a century. The earliest experiment in this topic was done by Triplett in 1898. Studies in the first half of the 20th century offered conflicting results as to whether human presences help or hurt performances. In 1965, Zajonc attempted to classify experiments to explain the differences—his conclusion was that simple, repetitive and well-practices tasks are helped by the presence of other humans, while more complex tasks that require thinking and experimenting are inhibited—the person was found to be more prone to errors when there were others observing the task [Zaj66].

Since Zajonc's paper, there have been many attempts to verify or disprove his arguments, or offer a different explanation for the disparity; for a summary, see Paulus' book [Pau89]. They all agree that complex tasks are disturbed by the presence of others. We can deduce from these results that the knowledge that there are users who are patiently waiting for the system to be repaired can make the operator to make mistakes the person otherwise won't make.

The problem illustrated here is that the system administrator is more likely to make errors when they can be least afforded, which is obviously not a desirable situation. We think the self-maintaining system will improve the situation, and thereby improve availability.

2.2.3 Availability

A system that relies on a human operator to function continuously will not be as available as one that maintains itself. A system without enough redundancy, even with an operator with an around-the-clock pager that can be called in even during the middle of the night, cannot be expected to recover in a few minutes, let alone a few seconds like our design. It usually takes much longer for a human operators to repair problems than machines do [GS91].

2.3 Our Approach

I am proposing a “self-maintaining” approach to system administration, in which the storage system maintains itself with only minimal help from a human operator. Instead of having someone constantly on call look after the system, our system is designed to mask problems and run continuously even when there are several independent component failures. The system administrator’s job is only to come in for scheduled visits, for instance, once per week, to replace the failed components.

As administration is most of the cost in running a large server, such a reduction will also save a great deal of money. In 1999, a system administrator’s salary is around $100,000 per year, plus benefits adding another $20,000 per year. Suppose there is an organization that hires such a person to administer a cluster of computers. If this person can work part-time, by only coming to work once per week, an organization can reduce the salary to $1/5$, or save $96,000 per year.

If the system administration cost was three times that of storage hardware cost as was found in Strategic Research’s study, and we can reduce the system administration cost to $1/5$, then we can calculate the the overall storage system cost reduction to be 60% using the following equations:

$$\begin{aligned}
\text{Cost}_{maintenance} &= \text{Cost}_{hardware} \times 3 \\
\text{Cost}_{original} &= \text{Cost}_{hardware} + \text{Cost}_{maintenance} \\
&= \text{Cost}_{hardware} \times 4 \\
\text{Cost}_{self-maintaining} &= \text{Cost}_{hardware} + \text{Cost}_{maintenance}/5 \\
&= \text{Cost}_{hardware} + \text{Cost}_{hardware} \times 3/5 \\
&= \text{Cost}_{hardware} \times 8/5 \\
\frac{\text{Cost}_{self-maintaining}}{\text{Cost}_{original}} &= \frac{\text{Cost}_{hardware} \times 8/5}{\text{Cost}_{hardware} \times 4} \\
&= 2/5
\end{aligned}$$

Our system accomplishes this goal of self-maintenance by a combination of redundancy and design. The next chapter describes the system in detail.

2.4 Summary

Reducing the burden on the system administrator is important for storage systems. The system will be cheaper, safer, and more available if it can manage itself rather than rely on being watched by a human operator. I am proposing a “self-maintaining” approach to system administration to attain that goal, as well as reducing the burden on the human operator by other means.

Chapter 3

The System

This chapter illustrates the system on which my research was conducted. The application, as well as the hardware and software architecture of the system, are described here.

3.1 The Application

The main application for the Tertiary Disk prototype is an image database holding pictures of objects of art [TAP$^+$98]. The "Thinker" site (`http://www.thinker.org/`), run by the Fine Arts Museums of San Francisco, has been providing access to art images through the Internet since October 1996. They implemented a searchable index of their art objects, through which the user can query their database of over 70,000 images for keywords, such as artist name, title and description. The user is presented with a page of thumbnails of images that fit the search criteria, and by clicking on the thumbnails they can view larger versions of the images. The largest images they provide are about 500 pixels on one side, many of them converted from color to gray-scale JPEG to save space. The original files were much larger, up to 3,096 $\times$ 2,048 pixels, but due to disk space constraints and lack of a way to adequately present them to users over the Internet, they decided to only provide relatively small images.

3.1.1 Problems

Understandably, one of the most common complaints from their users was "can't you make the pictures bigger?" We provide disk space for those larger versions of the images. There were several reasons why it was not possible for the museum to provide larger versions of the images initially.

Disk Space The full-size version of a color JPEG averages a little less than 1MB per image. According to advertisements on an August 1996 edition of MicroTimes, the largest 3.5-inch drives available on the market at that time were around 4.3GB and their prices ranged from $800 to $1,000. To hold 70,000 of 1MB JPEG images, they would have had to build a system with 20 of these drives, which would have cost at least $16,000 for raw disks only. The people at the museum neither had the budget nor expertise to maintain a storage system of that scale. The TIFF versions, which have higher quality than JPEGs, average about 12MB per image. To hold TIFF versions of all images, the space requirement will balloon to 840GB.

User's Network Connection Many people connect to the Internet using modems. A standard connection using a 56kbps modem can transfer at most 4KB per second. It takes over four minutes to download a 1MB JPEG and almost an hour for a 12MB TIFF.

Memory Usage JPEG is a compressed image format; when uncompressed for display, they will expand to about the same size as a TIFF image. In other words, there may only be 1MB transferred over the network link, but every full-size image the user displays requires 12MB of local memory just for raw image data. A few of these images will quickly expand the size of the browser to 100MB or more, making them unacceptably slow.

Screen Size There is no computer screen that is large enough to display an image 3,000 by 2,000 pixels, so it is not feasible to just dump the entire image to the user's web browser even if the user has enough memory and a fast network connection.

3.1.2 GRIDPIX

To solve the problems mentioned in the previous section, I wrote a web-based image viewing system called "GRIDPIX". GRIDPIX uses tiled, layered JPEG images with multiple resolution levels and simple HTML to achieve a "zooming" effect [Asa99].

The GRIDPIX File Format

Figure 3.1 shows a three-layer GRIDPIX file. Tiles 1 and 2 form the smallest layer; tiles 3 through 8 are the middle layer, and the largest layer consist of tiles 9 through 23.

Figure 3.2 shows the interaction between the client and the server. When the client requests an image, a CGI script (`mkhtml.cgi`) returns an HTML page describing the page layout. The individual image tiles are retrieved by a separate CGI script (`gettile.cgi`). GRIDPIX is completely HTML-based, so any graphical web browser can function as a client.

One important characteristic of the GRIDPIX server is that it requires very little processing cycles during runtime. The images are already divided up into tiles by the TIFF-to-GRIDPIX converter, so the numerous calls to `gettile.cgi` only require two accesses to the file; to get the offset and size of the tile from the header, and to retrieve the tile itself. In particular, there are no JPEG encoding/decoding required on the server side. Moreover, the GRIDPIX files are very compact, so after a few accesses, most of the requests will be handled by the server's on-memory disk cache, not the disk surface.

3.1.3 Status

The site opened to the public on March 2, 1998 with about 20,000 images [TAP+98]. We currently have over 70,000 images available, occupying about 2.5TB of storage space when fully mirrored. Each images are stored in several different formats, from the original PhotoCDs, which are about 5MB per image, to human-processed TIFFs, which average about 12MB per image, to GRIDPIX, which are about 1.2MB per image.

The reason why we couldn't initially provide all the images as converted from PhotoCDs is because the photographs are not of good enough quality to be presentable, and require manual work to crop, reorient and color correct them before they can be presented to visitors. According to

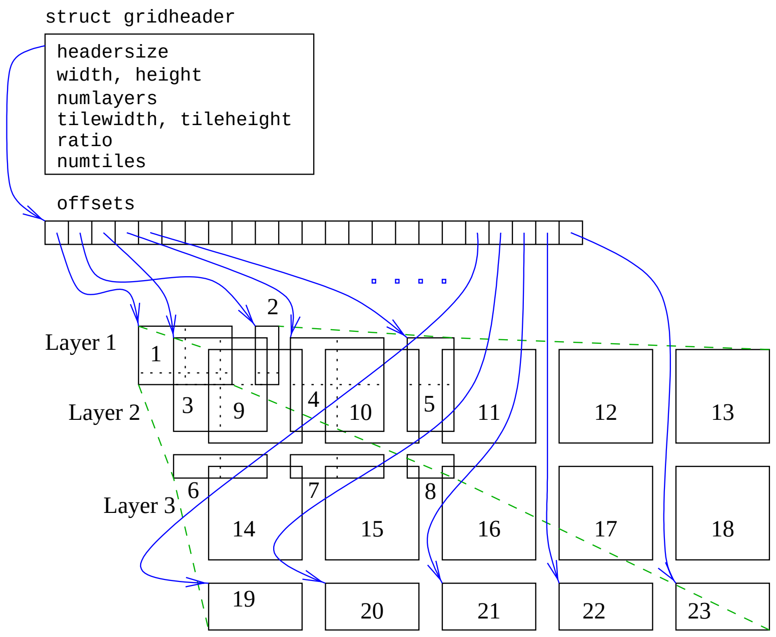


Figure 3.1: Three-layer GridPix image

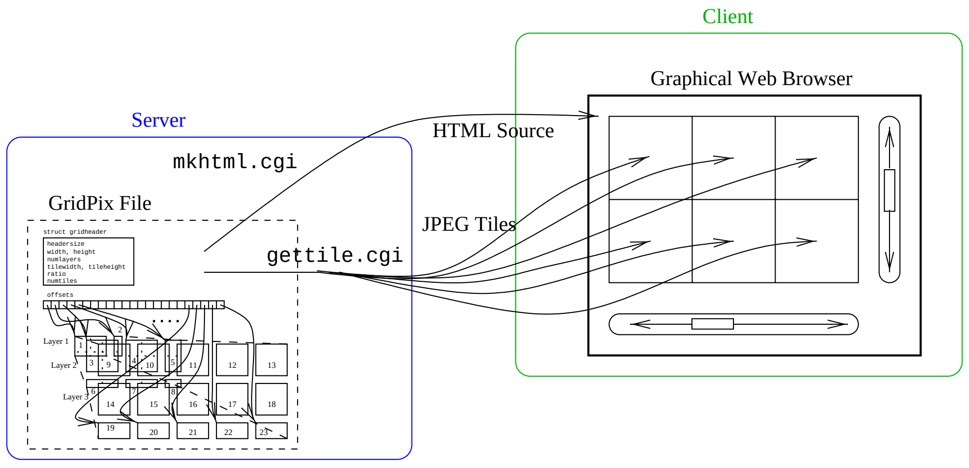


Figure 3.2: GRIDPIX interface

the people in the museum, the response from the public has been very favorable. Also, we have yet to experience any down-time other than campus-wide network or power failures despite individual machines crashing or being rebooted many times.

I have also gotten a few inquiries from people who are interested in using GRIDPIX to present their own images.

3.2 Tertiary Disk Architecture

The Tertiary Disk group has constructed a prototype disk storage system, on which I have conducted this research. The prototype's total capacity is 3.2 terabytes. The system occupies seven racks, each 7 foot tall and 19 inches wide. Figure 3.3 shows the system configuration. Here are some highlights:

- 20 PCs (200MHz Pentium Pro with 96MB of memory each) as disk servers
- 364 8.4 gigabyte IBM DCHS 7,200RPM Ultra-Wide SCSI disks with SCA (Single Connector Attach) connectors
- 52 Trimm Technologies model 381 8-disk SCA enclosures with serial interface
- 44 Adaptec 3940UW twin-channel Ultra-Wide SCSI adapters
- 2 16-port (4×4) 100Mbps fast Ethernet switches
- 4 24-port serial terminal servers for PC consoles and disk enclosure interfaces
- PCs run FreeBSD operating system, with minor modifications
- Double-ending of SCSI buses for high availability
- Redundant front-ends use HTTP redirect to route user requests
- 6 uninterruptible power supply (UPS) units
- Remotely-controllable power switches on all PCs
- 2 PCs (133MHz Pentium) as HTTP frontends
- 2 PCs (200MHz Pentium Pro) as infrastructure servers

There are two different configurations with regards to PCs and disks. Four PCs are configured in a "disk-heavy" configuration, called A-nodes, with 70 disks per PCs. The remaining 16 are in a "CPU-heavy" configuration, B-nodes, with 32 disks per PCs. SCSI buses on B-nodes are running at 20MHz (Ultra-SCSI speed), while the A-nodes are running at 10MHz (Fast-SCSI speed). The reason for this restriction is because the longer cables and high capacitance from having the maximum number of devices per bus on A-nodes made them unstable with 20MHz operation.

Serial ports act as system consoles for all but one of the computers. The serial ports enable the computers' consoles to be accessible to automated maintenance scripts. One of the frontend machines has a standard monitor and keyboard to provide access to the system when the operator is in the machine room.

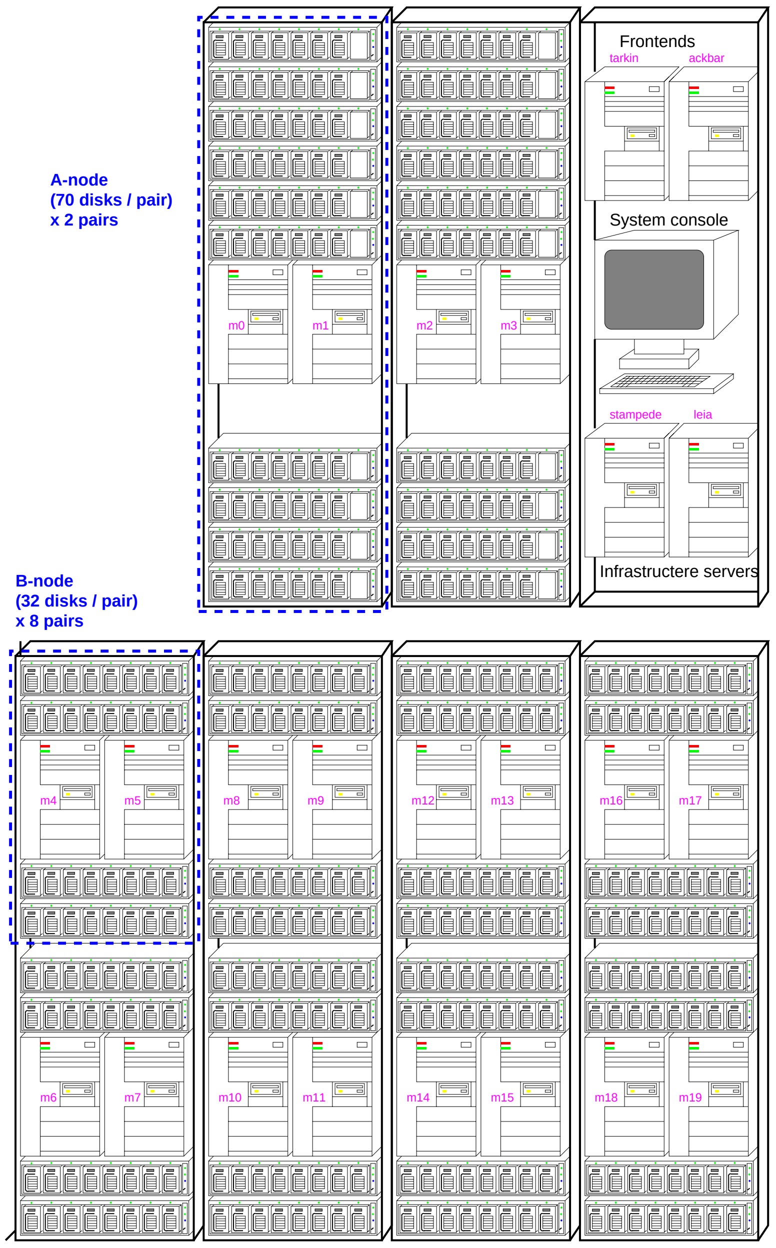


Figure 3.3: Tertiary Disk Final Prototype

3.2.1 Commodity Components

The most important feature is that this whole system is built only from commodity, off-the-shelf, components. The use of commodity components lowers the cost of storage by factors of two to four compared to standard RAID boxes. At the time of construction of the system in the summer of 1997, large RAID arrays cost about 60 cents per megabyte; our system cost about 20 cents per megabyte using street prices of components.

3.2.2 Redundancy

In designing the TD prototype, we have taken care to ensure it does not have any single point of failure.

Power

Multiple UPS units provide power for the system. There are power rails on either side of the racks, connected to different UPS units. UPS units also provide 10 minutes of standby power to survive temporary glitches in the main power line.

Each enclosure has two power supplies. They are connected to power rails on opposite sides of the racks, and thus to different UPS units. Each power supply has a built-in fan, and there is a third fan in the enclosure to ensure that the airflow around the disks will not be reduced to half when one power supply fails.

Double-Ending of SCSI Chains

Double-ended PC pairs are connected to different power rails. They are also connected to different network and serial switches. Termination on the SCSI bus is supplied by the SCSI adapters. Figure 3.4 describes double-ending.

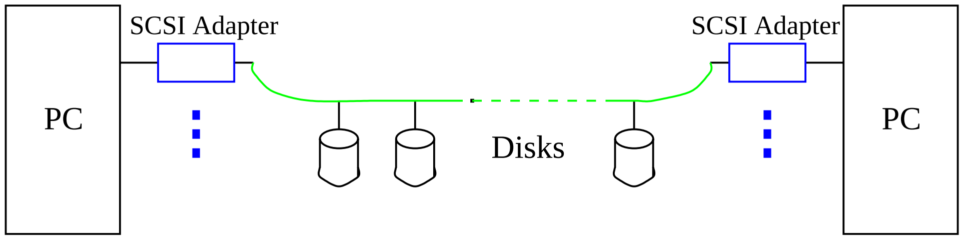


Figure 3.4: Double-Ending of SCSI disks

Our initial design used feed-through terminators to provide termination so the bus integrity is not compromised even if one of the PCs lose power (Figure 3.5). However, the final prototype was built without the external terminators. Terminators were omitted because of physical constraints of the particular parts we purchased—the terminators could not be installed on some of the connectors on the twin-channel SCSI adapters. Also, the SCSI adapters had special jumpers which would enable constant termination, i.e., the adapter will supply termination to the bus even when the machine is turned off.

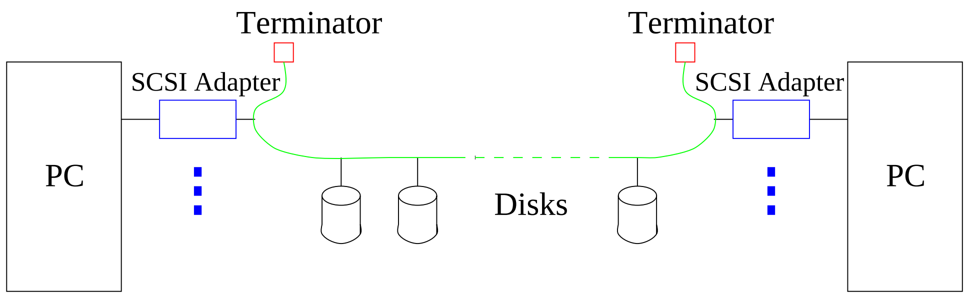


Figure 3.5: Double-Ending with feed-through terminators

Having feed-through terminators affects the system in two ways. One is that it is possible to remove a PC while still having the disks being accessed by its double-ended pair. However, this is not possible in our setup, since the SCSI components are not properly shielded from power and signal glitches, and disconnecting a cable can cause incorrect functioning or component damage. We plan to do any repair work only after powering off both machines as well as all the disk enclosures involved, so having feed-through terminators will not help in this way. We have sufficient redundancy so that powering down these two PCs will not deny access to images.

The other effect is that the disks on the SCSI bus are still accessible when one of the SCSI adapters have failed to the point that it is no longer able to supply proper termination. Similar situations can arise if the SCSI adapter does not have a consistent termination setting, a PC power supply, power rail or UPS unit failure, or the machine being turned off by the operator. Given the failure frequencies we have observed, these four kinds of failures are very unlikely. The last possibility—one of the two PCs being powered off—has happened only when there was a part replacement, so is already discussed in the previous paragraph.

Data Layout

Most data are mirrored. Stable data, not necessary for the system's day-to-day operation, are backed up by CD-ROMs.

The rightmost column of Figure 3.6 shows how the data are mirrored. Each of the small boxes indicate disk enclosures holding 7 or 8 SCSI disks; for instance, the `m0-m1` pair has 10 disk enclosures and the `m4-m5` pair has 4.

The `m0-m1` pair and `m2-m3` pair, the A-nodes, back up each other, since they have more disks than the other 16 disk servers with the CPU-heavy configuration. The remaining pairs, the B-nodes `m4-m5` through `m18-m19` are also matched up with pairs with the same number of disks.

3.2.3 Topology

This section explains the topology of power and Ethernet connections in our system.

Power Switches

Figure 3.6 also shows how the PCs are connected to the remotely controllable power switches.

To ensure that a power switch failure will not take down both PCs of a double-ending pair, the machines are hooked up in a crisscrossing pattern.

As a reference, Figure 3.7 shows how the PCs would be connected if we did not have feed-through terminators or SCSI adapters with constant termination jumpers. Since a power switch failure would take down all SCSI buses connected to the PCs on that switch, it is pointless to connect the PCs that form a double-ended pair to different switches. In this case, the only focus should be to make sure that the two double-ended pairs that hold mirrored copies of the same data (e.g., `m4-m5` and `m12-m13`) are not connected to the same power switch.

Ethernet Switches

Figure 3.8 shows how the PCs are connected to our two 100Mbps Ethernet switches. Ours are 4×4 switches, meaning the 16 ports are arranged in four groups of four ports each, with the ports in each group sharing a common bus and a switch at the base to route between each groups. For instance, with the topology in Figure 3.8, `m0`, `m4` and `m6` can send or receive a *total* of 100Mbps of data, while `m4`, `m8`, `m12` and `m16` can all transfer at 100Mbps *simultaneously*.

The connections are designed to balance the load in case one of the PCs of the double-ended pair fails, or one of the mirrored copies become unavailable.

3.2.4 Disk Interface

There were several disk interfaces to choose from at the time we built the prototype. This section briefly compares them and explains why we decided to use SCSI to build our system.

IDE

The most basic and cheapest of the alternatives, IDE, is very popular as the system disk for PC-based systems. However, it is not sufficient for server systems such as ours. For instance, at the time we built the prototype, most IDE systems did not support bus-mastered DMA (direct-memory access), although today's IDE systems support it. IDE only allows two disks per bus, which limits the design space since the possible maximum number of disks per PC is much lower than all other alternatives. IDE also does not allow multiple initiators on a single bus, making it infeasible to use with double-ending.

SCSI

Small Computer Systems Interface (SCSI) is a parallel bus protocol that is widely used for disk servers due to its availability on the market and flexibility. SCSI disks usually cost 50–80% more than their IDE counterparts but are cheaper than FibreChannel disks. The maximum number of devices (including the SCSI host adapter) is equal to the bus width, so you can have 8 devices on an 8-bit ("narrow") bus and 16 devices on a 16-bit ("wide") bus. The SCSI standard allows multiple initiators on a single bus.

SCSI has been doubling the transfer rate every three years, from 5MHz to 10MHz to 20MHz to 40MHz to 80MHz in bus frequency and from 8 bits to 16 bits in bus width. The fastest SCSI bus today runs at 160MB/s, with 80MHz bus frequency on a 16-bit bus. However, the higher clock rates has put stricter demands on cabling and electrical characteristics of components.

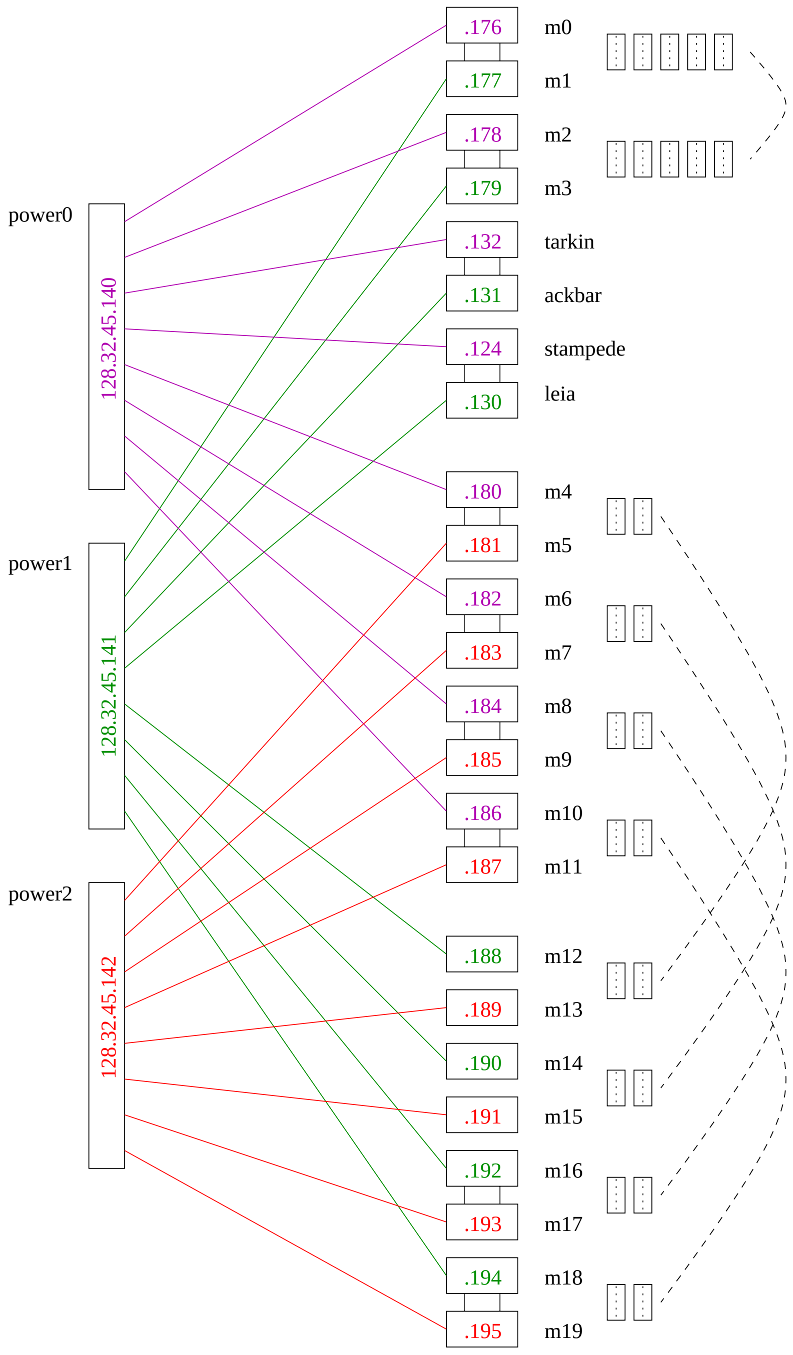


Figure 3.6: Power connection and mirroring topology on a system with constant termination

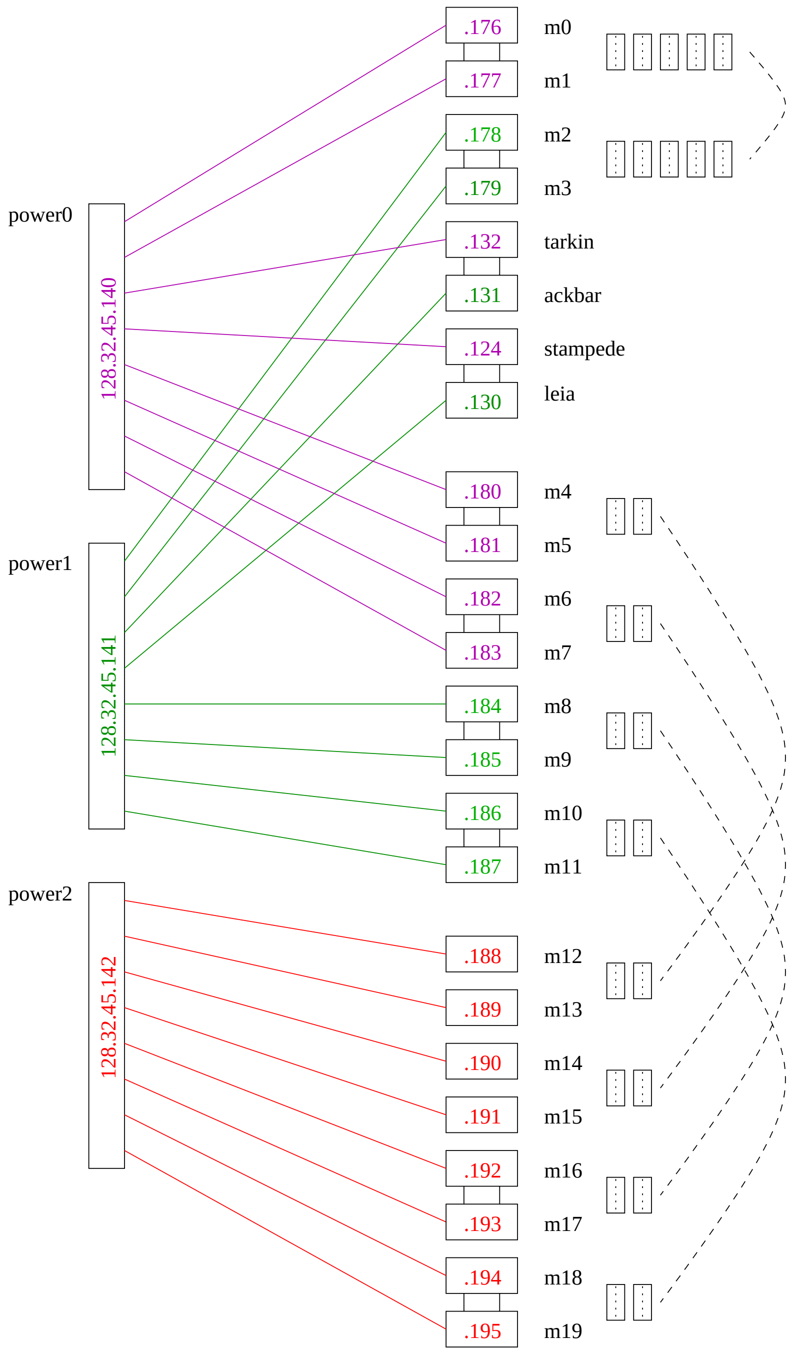


Figure 3.7: Power connection and mirroring topology on a system without constant termination

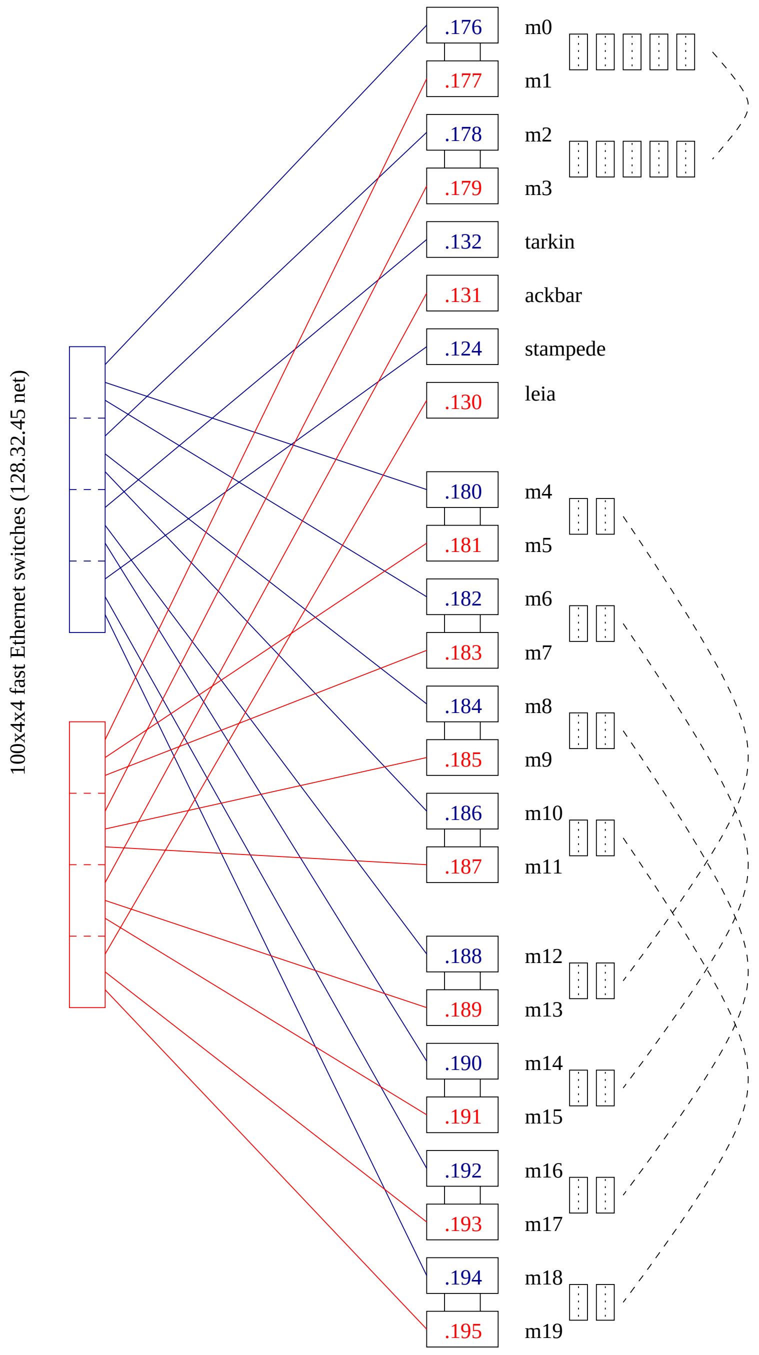


Figure 3.8: Ethernet connection and mirroring topology

For example, the maximum cable length for 20MHz operation, the maximum possible for our disks and adapters, on a traditional “single-ended” SCSI bus, is 1.5 meters.

The current standard in the market is “differential” SCSI, which allows a much longer cable by pairing signal and ground pins instead of sharing a common ground signal. The maximum cable length is 12 meters for 20MHz operation using differential SCSI. Several external SCSI RAID boxes used differential SCSI to connect the RAID chassis to the host for this reason even before differential SCSI was of widespread use.

The new serial interfaces, described in the next two paragraphs, also mitigate the cable-length problem.

SSA

Serial Storage Architecture (SSA) is a point-to-point serial interface. SSA allows up to 20MB/s to be transferred in both directions at any given link, and supports up to 126 devices on an SSA segment. By connecting devices in a circle, it is possible to transfer up to 80MB/s total (40MB/s read, 40MB/s write) on one host adapter by moving data in both directions simultaneously. SSA has the characteristic that a single link failure in a loop will *not* compromise the integrity of the entire segment, as the loop will simply become a string when the failed link is logically disconnected by the SSA circuits on the devices on both sides of the failed link.

FC-AL

Another standard, FibreChannel-Arbitrated Loop (FC-AL) is a loop-based serial interface. FC-AL allows up to 127 devices on a single loop. There are several versions of the interface, varying in cost and performance. The fastest implementation can transfer up to 1.06 Gbits per second, or about 100MB/s. One of FC-AL’s weaknesses is that a single link failure will cause the entire loop to cease functioning. This problem can be avoided by integrating two loops, one in each direction, into one, or by using a repeater hub with automated failure isolation features.

In 1998, the SSA and FC-AL groups have agreed to merge their standards. The new standard is mostly FC-AL based but also incorporates SSA’s link-to-link characteristics for safety. The new SCSI-3 standard will run on several physical transport layers, including SSA/FC-AL.

At the time we were testing equipment for our prototype, IDE and SCSI were the only interfaces for which devices were widely available and affordable. Between those two, SCSI was the easy choice due to technical limitations of IDE. Also, SSA and FC-AL standards were still highly fluid at that time.

3.3 Software Architecture

There are several aspects of software architecture that need to be discussed. Here I will explain how we handle user requests, and what modifications we made to the operating system to support our system.

3.3.1 Handling User Requests

Our main application, as described in section 3.1, is a web server for fine art images. Figure 3.9 shows how user requests are handled. When a user at the museum's site selects an option to display the GRIDPIX version of an image, the request first arrives to a frontend machine. The frontend will then look up a table and return an HTTP redirect message to the user's client, which subsequently reconnects to the backend machine holding the image. From that point on, the client interacts directly with the backend machine until the user has finished examining that image.

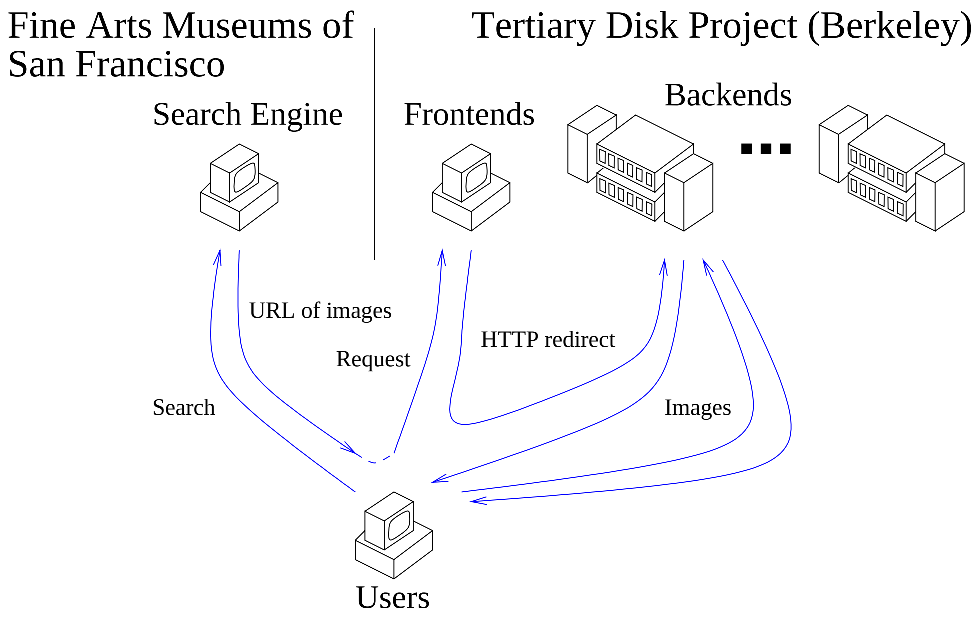


Figure 3.9: Handling user requests

Masking Failures

There are several levels of masking we can do for failures. In one extreme, a system can have non-volatile RAMs holding state information for open TCP connections so a machine crashing and rebooting will not cause any connection to be lost. The other extreme, like NFS, is to have a stateless server with clients retrying until the server replies. This model causes the client to lock up until the server recovers, but will not lose any requests. It is even possible to replace a failed server with another machine as long as the device IDs and i-node numbers of the filesystems stay the same—from the client, it will just look like an extremely long network failure or server reboot.

Those two models are required in a local-area network environment where correctness of response is most important. Our premise, as mentioned in Section 1.1.3, is that since the Internet is so unreliable, it doesn't make sense to try to mask all of the failures. Our goal then becomes how to implement the "repair by reload"; in other words, how to make sure the system will be able to mask any failure within a few seconds to allow retries from end-users to succeed.

Frontend Switching

Figure 3.10 shows how we mask frontend failures from users. We have two frontends, `tarkin.cs.berkeley.edu` and `ackbar.cs.berkeley.edu`, backing up each other using IP aliasing. The canonical address, `gpx.cs.berkeley.edu`, is usually an alias of `tarkin`. The other machine, `ackbar`, checks every 5 seconds over the local area network to see if `tarkin` is up. When it can't contact `tarkin`, it will take over `gpx`. `ackbar` will still keep checking for `tarkin` every five seconds, and will release the `gpx` alias as soon as it finds `tarkin` back up.

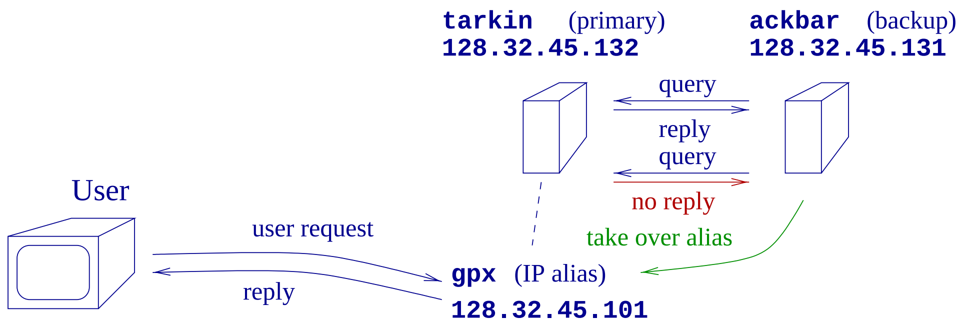


Figure 3.10: Frontend switching protocol

This simple method will handle all four of the cases where `tarkin` or `ackbar` are down or their network interfaces are not functional. Table 3.1 summarizes them. Note that when `ackbar`'s network interface is broken, `ackbar` will attempt to take over, but the clients, which can only reach `tarkin`, will continue to connect to `tarkin` without any ill effects.

Cause	Response	Clients connected to
`tarkin` down	`ackbar` takes over	`ackbar`
`tarkin` network failure	`ackbar` takes over	`ackbar`
`ackbar` down	(none)	`tarkin`
`ackbar` network failure	`ackbar` takes over	`tarkin`

Table 3.1: Frontend switching effects

Backend Failures before the User Connects

Both `tarkin` and `ackbar` check all the GRIDPIX servers every 5 seconds. This checking is done by fetching a particular file from the machine; the file will not be available if the HTTP server is down or the disk is having problems. When a frontend discovers problems on one of the machines, it will automatically forward any subsequent requests from users to the backups. Since the CPU load on the frontends are very low, and the scripts are very simple, it is very unlikely that these scripts will not detect problems quickly.

Backend Failures while the User Is Connected

The above two methods will cover most cases except for one—users already in a GRIDPIX session when a server goes down. There are two ways to mask this case: to implement something similar to the frontends, having two servers back up each other, or have another machine, possibly one of the frontends, to take over the IP address temporarily and forward the request to the backup. The latter way is better for two reasons. First, the backends do not need to know which machines they are backing up. When data is moved around, or when additional mirrors are added, the first method will require updates of file lists on the backends in addition to the frontends. The second reason is that is that it is easier to implement, since the frontend already has support for querying the status of another machine and taking over, as well as forwarding requests to a different machine.

Load Balancing

Currently we are just redirecting the user to the first machine on the list, since the load is very low compared to the machines' capacity. However, our frontend-backend protocol can also be used to balance the load among backend machines if it becomes necessary. By having the backends put some information in the file fetched by the frontends, they can easily communicate to the frontends how much load they are experiencing.

3.3.2 Operating System Support

In this section, I will describe what modifications we had to make to the operating system in order to build our system.

SCSI/Disk Subsystems

The SCSI and disk subsystems were the ones that caused most problems for us. It is understandable because commodity hardware is not usually designed with a large server system like ours in mind. Experience with scale is one place where specialized storage system manufacturers have an advantage.

The exact nature of the problems and our fixes to them are not part of the main focus of my research; therefore, they are not covered in this dissertation.

Disk Identification

One problem with having several hundred disk drives, all looking identical, is that it is very easy for the operator to confuse them. SCSI disks are identified by their SCSI IDs within their bus, which are distinguished by the host adapter number within the machine, which are in turn identified by hostnames and IP addresses. Each host keeps a static configuration table, typically called `/etc/fstab`, which describes which disk is mounted on which directory in the filesystem hierarchy. In cases of concatenated or striped sets, there may be extra configuration files describing the setups of those virtual disks. It is essential that relationship between the disk IDs and the actual disks kept consistent.

The problem is that if two disks are exchanged, the operating system may not be able to tell the difference; it will just use the disks until something fails. This problem is especially true for SCA disks, as SCSI IDs are specified on the disk enclosures, and if a pair of disks are accidentally swapped, their IDs will change with the location in the enclosure. There could be several different consequences ranging from OS crash to application error; some of them involve corruption of filesystems and are extremely dangerous, especially when the disk is part of a striped set.

We needed to design a system in which the operating system will be able to notice when disks are installed incorrectly. There are various ways to implement this, with trade-offs in the effects and complexity of design. Here are the few alternatives we considered.

Serial Numbers

Each disk has a unique serial number written in its permanent memory at the factory. By reading this number, the operating system can ensure that all the disks are in correct locations by keeping a list of serial numbers of the disks and comparing disks to it upon each boot.

Comment: It is easy to implement, but will not let us do more than simple safe-guarding against operator errors.

Disk Label

Each disk has a "disk label" which describes the partitioning of the disks as well as some other information about the disk, such as rotational latency and geometry [BSD94]. It is possible to expand the format of the disk label to include some more information, such as expected bus/SCSI IDs and mount points.

Comment: There are more features we can implement with this than with the previous option; it can be used for simple safe-guarding with bus/SCSI IDs to more complex tasks such as automatic mounting. Note it doesn't even require a table to be kept on the system, as the disks record on themselves where they are supposed to be mounted on the system. However, since the disk label has a fixed size, there is still a limit on the complexity of what we can do. For instance, we'll need to add special fields to describe if the disk is part of a striped set, and if it is, how many other disks are there, where in the set this disk appears, and so on. It is not very extensible, and in order to make new options on how to use the disk available, the kernel or the utility to read the disk label needs to be recompiled.

Script

The last option is for each disk to have a "known" location—possibly a fixed partition—on which there is a filesystem where there exists a script that is to be executed in turn when the system boots.

Comment: What we can do with this option is virtually unlimited. The scripts can be used to check the disk's identity—the system boot process just needs to call the disk's script with its bus/SCSI IDs as arguments. It can be used to auto-mount necessary filesystems, or it can be used to construct more complicated entities such as striped arrays. Note that in either case, there is no per-host configuration required on the boot disks themselves; the boot disks just read in the scripts and execute them in turn. If the scripts are written cleverly, it may even not matter what order the disks are inserted in a particular enclosure.

If a disk is moved to a different machine, it is not possible for our system to rectify the situation without an aid of an operator; we would need some kind of distributed filesystem to handle such cases.

I have implemented the last option, the script method. It is now possible to move disks around within the same machine and still have them mounted correctly, both for the single filesystem case and striped array case. The script is made to flag an error when the disk is inserted in an inappropriate enclosure. The scripts also automatically select the correct PC of the double-ended pair upon startup, and will refuse to be configured if the disk is connected to a wrong machine. The checking is done by having the startup process pass an extra argument, the machine name, to the disks' scripts.

Fast Reboot

It is important to reduce the time required for rebooting the system in order to minimize the window of vulnerability. Table 3.2 summarizes the effects of various improvements I have made, or have explored on our system. The numbers are based on a system with three SCSI adapters.

Reboot step	Improvement	Old time	New time	Reduction
Check disks	Use soft updates	20 minutes	28 seconds	20 minutes
System BIOS	BIOS-less boot	78 seconds	0 seconds	78 seconds
SCSI probe	Reduce SCSI delay	24 seconds	11 seconds	13 seconds
Device probe	Remove components	31 seconds	21 seconds	10 seconds
Total		22 minutes	60 seconds	21 minutes

Table 3.2: Reboot steps and timings

These are more detailed explanations on each item, in decreasing order of significance.

Fast `fsck` The largest amount of time the machine takes after a crash is spent running `fsck`, the UNIX file system consistency check and repair program [McK85]. It takes about 20 minutes to run `fsck` on our 15-disk striped arrays. In the past, `fsck` checked all the filesystems during every reboot, but in recent versions of UNIX, this has become only a problem after an unclean shutdown—`fsck` will check the filesystem flags and will take no further actions if the filesystem was unmounted cleanly, so unless the operating system crashed or the machine hung and was power-cycled, `fsck` will complete in less than a second. Note that multiple `fsck` processes can be run in parallel, so the number of such arrays do not change the total amount of time it takes to check all filesystems.

Kirk McKusick, the author of `fsck` and architect of the original BSD Fast Filesystem [McK84], has been working on a project called "soft updates", in which by changing the way dirty data is written to the FreeBSD filesystem, both the performance and reliability improves greatly. These filesystems also do not need `fsck` to be run even after a crash [McK98]. The filesystem may lose some space, but the data structures on the filesystem will be consistent with each other. We still need to run `fsck` from time to time to reclaim the space, but it can be done at any time, not right after a crash.

I have been using soft updates on our machines for over a year with no ill effects. When I started using soft updates, it was supplied as external "snapshots" from McKusick's repository; the code is now in FreeBSD's main source tree.

BIOS-Less Boot I have also explored the possibility of implementing a “quick boot” option of the operating system, which doesn’t use a hardware reset. Our systems take 78 or 57 seconds, depending on whether they have three SCSI adapters or two, just to go through the BIOS. Most of the time is spent while SCSI cards are initializing themselves, but this is not really necessary for a warm boot. Also, about 5 seconds is spent checking available memory, which is obviously useless too if we are rebooting a previously-running machine.

By having a reboot just jump to the initialization code in the kernel, we can completely eliminate the lengthy BIOS process. However, various kernel modules are not necessarily written to be reentrant, and although I did get some favorable responses when I mentioned this in the FreeBSD developers’ mailing lists, I did not have time to actually implement this.

Reduce Time of Wait for SCSI Devices The delay for SCSI probe after a SCSI bus reset was 8 seconds per bus, but for our system, it could be reduced to 2 seconds without any noticeable effect. The default delay was based on a pessimistic expectation that the user may have an old device that takes a long time to reset itself after a SCSI bus reset is issued.

Remove Unneeded Kernel Components By removing unneeded kernel components, the kernel becomes much smaller and uses less memory during normal operation. It is also faster to load by about half a second and takes much less time to boot because it does not have to probe devices that do not exist on our system. The removal of unneeded components has reduced about 10 seconds from the boot time, from about 31 seconds to 21 seconds. Of the 21 seconds, 11 seconds are used searching for the second IDE drive—the drive our machines don’t have, but are in the kernel so we can make duplicates of system disks quickly when there is a IDE disk drive failure. By removing this driver, it can be further reduced to 10 seconds.

3.4 Summary

The Tertiary Disk group has built a 3.2TB storage system consisting of 364 SCSI hard drives and 24 PCs, including frontends and infrastructure servers. We have also written an application, a web-based image database server, and modified the operating system in order to run our application more reliably and make it easier to maintain. The system has been online since March of 1998 and has received favorable reactions from users. Other than campus-wide network or power outages, Tertiary Disk has not had an outage that made images unavailable to outside users.

Chapter 4

The Cost of System Administration

As mentioned earlier, while storage system hardware has gotten cheaper over the years, the cost of system administration has remained high. Since the decrease in cost of hardware is not showing any signs of slowing down, this situation is not likely to change in the near future. If system administration cost is many times that of hardware cost, clearly there is little or no point in trying to reduce the cost of hardware. Thus, for our system to make sense, it is necessary that we reduce the cost of system administration by at least an order of magnitude. There are several means to accomplish this. We focus on two methods in this dissertation.

One is to make the system *self-maintaining*. By having the storage system maintain itself, we will be able to reduce the burden on the system administrator, which in turn results in lower cost and higher reliability. The issue of system administration is not only that of monetary cost. A system that functions continuously without an aid of a human operator has a higher availability than one that requires human intervention to keep it running. Also, the pressure on system administrators to fix the problems immediately often results in human errors that compound the hardware problems. With large storage systems, results of such operator errors can be disastrous. The self-maintaining aspect of the system is discussed in Section 4.1.

The other is to make the system *easier for the operator to maintain*. These methods are discussed in Section 4.2. It is ideal to have the system be completely care-free, but there are many aspects of storage system administration that cannot be made fully automatic. However, there are other ways to reduce the burden of the system administrator. Although these improvements are hard to quantify, they are nonetheless useful in increasing the availability of the system, thus indirectly reducing the cost of administration. For instance, a system which safeguards against operator errors will have a better uptime than a system that requires the operator to make no errors. An example of such a “safety belt” is the automatic disk identification scheme mentioned in Section 3.3.2. Another way to make the system easier to maintain is to reduce the time it takes to upgrade the operating system, a topic that has been researched in the past in the context of binary upgrades to commercial operating systems [SMH95]. If the system administrator has to work late into the night to upgrade a cluster, the person is more likely to make mistakes near the end.

One thing my research does *not* address is management of data on a large storage system. Management of data involves keeping track of where everything is located, reallocating space as necessary, and so on. Although it is an interesting research field, there are other projects that are pursuing it and I did not think we could make my approach generic enough to be of a wide interest

[BGM+96, Wal98, Jir99]. For a summary of other research projects on this topic, please refer to Section 1.3.4.

4.1 Self-Maintaining System

In this section, I will define what "self-maintaining" means, and outline the requirements on how to construct such a system.

4.1.1 Definition of Self-Maintaining System

There are two aspects to self-maintainability of a system, depending on whether the perspective is that of the system administrator or of the user.

System Administrator's Perspective

To the system administrator, a self-maintaining system is one that does not require constant attention. For the purpose of my research, I define it as a system that is:

- designed to function with only pre-scheduled, say, weekly maintenance, and
- for which, at maintenance time, the required tasks are clearly defined.

There is nothing special about the repairs having to be weekly. The visits can be twice a week, monthly, or even daily. The important part is that the visits are pre-scheduled and not driven by events such as hardware failures. My research goal is to show that it is possible to build a system that runs with a reasonably-spaced scheduled visits by humans.

There are two major benefits of this approach:

Reduce Operator Errors
Regularly scheduled visits will help the operator's performance and reduce mistakes. An operator that is working during regular hours are less likely to make errors than one that was paged at, say, 3 A.M. Also, not being under the pressure of having to fix the system right away will likely reduce the chances of operator errors. Studies supporting these claims can be found in Section 2.2.2.

Reduce Operator Cost
As mentioned in Section 2.3, another benefit is that it is not necessary to pay a full-time wage for a system administrator of such a system. We can either hire someone part-time or have the same person administer many systems instead of just one or two.

The second rationale is similar to that behind Tandem's TMDS [Bar81, Tan97b]. Tandem has engineers at the support center 24 hours a day, and systems having problems will automatically dial up Tandem to contact one of the support persons. A few hours after the failure, the support person will show up at the customer's site with necessary repair parts. Thus, the companies that purchase support contracts do not have to hire operators by themselves; they are in effect "sharing" the operators with other companies through Tandem.

Our system will take this one step further. There is no need for anyone to be at anywhere in the middle of the night; in fact, there is no need for anyone to be at any central support center. The operators can travel from one customer site to another throughout the week, or they can be doing something else during most of the week. By hiring a part-time operator that only comes to work once per week, an organization can cut the human cost of administration to one fifth.

User's Perspective

There are two classes of users on a web-based storage system like ours.

End Users

These are the people who use the system from the Internet. They know nothing about the internals of the system, will easily get annoyed if something doesn't work, and may never return to our site if they are unhappy. They usually only issue reads to the system.

Content Providers

There are relatively few people whose job is to update the contents of the web server. They are part of the project, and can wait for a while if the system cannot allow writes at the moment. They have a good knowledge on how to use the system, but usually know little about the internal workings.

To reduce end-user frustration, it is important to reduce the system's down-time as seen from across the Internet. Note that a system that relies on an operator to keep it running is not as available as one that maintains itself, as it will take minutes or maybe even hours for the operator to actually be able to repair the damage [GS91]. Our goal is to have the system repair any interruption of service within a few seconds, and continue to function unattended until the next scheduled visit by the operator. For a web server application such as the one we are running, this is illustrated by the slogan "repair by reload".

As Mary Baker said in her Ph.D. thesis, "in the limit, as recovery time approaches zero, a system with fast crash recovery is indistinguishable from a system that never crashes at all" [Bak93]. Her research was about file servers on distributed operating system, and the above statement was qualified that it is only appropriate in systems that can tolerate short periods of down-time, as a cluster of workstations in a typical engineering or research environment.

I believe our application, a web server, is another example where a system with fast crash recovery is just as good as a system that never crashes. This claim is based on the observation that since the Internet is so unreliable, it will be impossible for the user to distinguish problems on our servers from the daily transient problems of the network.

For the content providers, the situation is a little different. It is permissible to have the system be in a state that it cannot receive input from them for a short while. However, repeated problems will affect their productivity as well as delay the update of the contents, so it is desirable to keep downtimes as short and as infrequent as possible.

In order to partially shield problems from the content providers, we offer a *portal* to which they can upload the new images. Once the data is in the portal, they can go on to their work. However, if there is a problem with some other part of the system that disallows writes, the new images may not be available on the web until the problem is fixed.

4.1.2 Requirements

There are several requirements for building a self-maintaining system. Here the requirements will only be listed; the implementation details can be found in Sections 3.2 and 3.3.

No Single Point of Failure
Such a system is not allowed to have any single point of failure. This restriction makes it possible to decouple the time of repair from the time of failure, allowing the system to run under an existence of a failure. Clearly, depending on how many failures the system should tolerate, it may be necessary to have even more redundancy. Not having a single point of failure is a minimum requirement for any system that is designed to function continuously in an event of a component failure.

Constant and Reliable Monitoring
The system should be constantly and reliably monitored so corrective actions are taken very quickly after failures. We use very simple shell scripts for monitoring, and have an interval of four seconds of sleeps between monitoring. The monitoring scripts are autonomous, so a failure on one machine will not cause a monitoring script on another machine to malfunction.

There are some characteristics in our application that makes it easy to build a self-maintaining system. Although these are not hard requirements, they nonetheless have helped us simplify the design of the system.

End-users Issuing Only Reads
One important aspect of our system is the read-mostly nature of end-user accesses. We believe this is not unique to our application; many other applications with similar terabyte-capacity scale share the same characteristics. By not having to allow writes in degraded mode, immediate recovery is only a matter of locating the backup and rerouting user requests there while more lengthy recovery procedures can take place.

Little Internal Communication
The application needs very little internal communication to handle user requests. See Section 3.3.1 for details on how user requests are handled. This low usage simplifies the recovery as there are only few messages or connections that might be lost due to a failure. Also, the low internal communication overhead enables the system to scale up nicely in the future.

4.2 A System That Is Easy To Maintain

There are several other improvements we have made to our system to reduce the burden on the system administrator. Although they do not relate directly to the "self-maintaining" aspect of the system, they are nonetheless important for the maintainability of the system and are described here as part of the effort to reduce the overall cost of maintenance.

As mentioned earlier, one of the most dramatic failures of our system was caused by the operator replacing a wrong disk and attempting to reconstructing a striped filesystem, destroying the only good copy of mirrored data. In order to avoid making any more similar mistakes, I implemented a disk identification mechanism that will automatically detect when a disk is connected to the wrong machine. It has also been useful when we had to replace disk enclosures—no matter what order the disks are inserted into the enclosure, the striped set will always be configured correctly. If the ordering of disks doesn't matter, it means one less thing to worry about when the operator is performing a time-consuming and stressful operation like replacing a disk enclosure, therefore reducing the chance of human errors in subsequent steps.

4.2.1 Disk Identification

As mentioned in Section 3.3.2, our system will not lose data if a disk is inserted in a wrong location in the cluster, even if it is part of a striped array. The method I implemented to accomplish this is very generic and extensible.

4.2.2 Taking Advantage of Redundancy

There are several ways to take advantage of redundancy in ways not possible on a traditional centralized server. I will describe a couple of those methods in this section.

Preventive Maintenance

People at Inktomi have found that their system can be made more reliable by restarting the application periodically [BL98]. The reliability problems are caused because of thread leak problems in the server process. Instead of spending many man-weeks trying to track down the last thread leak bug, it is much easier to clear the problems by a restart. Since their servers are fully redundant, it will not cause any interruption of service for the users.

Large servers are known to develop some bad processes from time to time. Anyone who has been in a clustered computing environment has seen many messages from the system administrator saying something like:

> "The file server is going to have a quick reboot today at 12:15 to clear wedged NFS processes—please send mail by 11:30 if you have a problem".

One solution to this is to periodically reboot the server on off-peak hours, instead of waiting for problems to develop. It will be even easier for both the users and the system administrator if the machines can be rebooted without any interruption of service. System administrators will especially benefit from it, since they will no longer have to worry about when to schedule the reboot, check users' mails to see if it is really ok to reboot it, and then wait at the console hoping the machine will come up in three minutes as scheduled.

We have discussed implementing similar methods with our system. In our case, it is not the software, but the hardware that benefits from reboots. The SCSI disks we are using sometimes enter a certain state from which it doesn't reply to SCSI commands anymore; it requires a reboot to fix. I suspect they are due to bugs in disk firmware that cause the state machine to go into a situation that is not foreseen by the designer. I have observed that the problem is caused much more often when there is a lot of traffic to several disks on the SCSI bus. Most of the time, the problems are recoverable by sending SCSI bus resets from the disk driver—we have worked with FreeBSD developers and had them add code to do just this—but sometimes the disks will not respond to bus resets, and the machine either loses access to the disk or crashes.

Since these problems are usually preceded by a distinct kernel message, we can prevent further damage by automatically rebooting the machine at first sight of the problem.

Testing New Systems

When new software, either operating system or application, arrives at a site, it is usually installed on few test machines that are set aside for this purpose. Only after the system administrator is confident about the reliability of these systems, they will be deployed in the main cluster.

There are several problems to this approach. First, the performance of the test system is not necessarily a good indication of the real system. Even with artificially generated high loads, the test is never going to be equivalent to the real workload. It is possible to trace the real workload and simulate it on the test machine, but this is very expensive and will require an experienced administrator to do it right. Second, the test-bed is not contributing to the overall performance of the system, as it does not serve user requests. In essence, the price and administration cost of the test-bed is an extra overhead in the cost/performance equation.

However, with a fully redundant and independent system like ours, it is possible to just go ahead and install the new version of software on few of the production machines. Since the system as a whole is designed to tolerate one or more machines going down, even the result of a completely dysfunctional software upgrade is not as disastrous as it used to be.

Note that this method only applies to subsystems which are expected to either work or fail, with no middle ground. In particular, if there are updates to the data through a particular subsystem, it will still be too dangerous to use a version of unknown quality as this could result in corrupted data.

4.2.3 Modular Upgrading

Upgrading a cluster of computers is always difficult, and one that doesn't have a simple solution. People usually write custom scripts to do it semi-automatically [SMH95].

Problems with the Build Process

FreeBSD is quite unique in this respect, as not only is the full source tree provided, it is also carefully maintained so users can rebuild the entire system with one command (`cd /usr/src; make world`). This command can be used to upgrade the system just by obtaining the latest sources. FreeBSD also offers the full revision history of the source tree, which enables compiling the entire system as it was at any point in time in the past, making it possible to select a "known good" source tree to build a system from after watching the mailing lists for a few days.

However, this approach has its drawbacks. If you want to compile the operating system and not extract it from a binary distribution, a fairly complicated process is required to make sure all the tools and dependencies are built in the right order. For instance, all binaries need to be built with new libraries, which may need new compilers, which may need new text formatters to process the header files, and so on.

This problem used to be solved by installing necessary components first on the host system, then running a full compilation. Other than the danger of having a partially upgraded system when the compilation fails in the middle, another drawback of this approach is that it puts a high load on the network when a cluster of computers is sharing the same NFS filesystem for sources.

Separating Build and Install Phases

I made it possible to segregate the build and install processes by changing the build process to install necessary components in a temporary tree instead of the host system's system directories. The old "`world`" target is now composed of two targets, "`buildworld`" and "`installworld`", which don't necessarily have to be run on the same system. "`buildworld`" will rebuild the entire system *without* affecting the host system; "`installworld`" takes the result of "`buildworld`" and installs it. Using this method, I have successfully upgraded our systems three times. By running "`buildworld`" on a fileserver and then "`installworld`" on each of the 20 clients, the entire system can be upgraded in a few hours, with only a minimal amount disk space used.

Note that under the old method, I would either have had to run "`world`" on one of the clients, causing the entire system to be compiled over NFS, or duplicate the source tree on all the machines. Running a large compilation takes a long time; for instance, on our machines, the "`buildworld`" takes about 1 1/2 hours if the source tree is on the local disk, and almost 4 hours when it is on an NFS-mounted remote disk, for an improvement of 2 1/2 hours.

Also, the new method makes it possible to recover from faulty systems easier, since building is done on a completely separate environment. If I installed a version of a system that is unreliable, all that is required is a few tools (`make`, `install`, etc.) and functional NFS to go back to any known good snapshot.

4.3 Summary

There are two benefits in reducing the system administrator's burden by making the system maintain itself. One is that a system that automatically maintains itself will be more available than a system that relies on humans to keep it running. The other is that a system that is easy to maintain can be cheaper because a full-time system administrator's salary is quite expensive.

This chapter analyzed these benefits and described some of the improvements I made to our system to reduce the cost of system administration. Those include; taking advantage of redundancy using preventive maintenance and field-testing of new systems, and modular upgrades of a cluster of machines.

Chapter 5

Failures

We have been observing our system carefully since we constructed it. The objective is to collect enough data to run a simulation to prove that the system will actually run continuously and flawlessly in the existence of failures. The failure records are during the 24-month period from July 1997 to June 1999. In this chapter, I mention how the data was collected and presents the results of analysis on the data. The simulation itself is described in the next chapter.

Nisha Talagala has presented another analysis of the same system [Tal99]. The data in this section differs in that I have analyzed data for a longer period of time, intending to collect as much data as possible for the simulation I was going to run, while she made a more in-depth investigation on each of the failures, with extra emphasis on how components gradually deteriorate over time until they fail. The data that is included in my summary and not in hers include repair actions, amount of time it takes for repair, and clustering of failures. The results that are common between our data are consistent with each other.

5.1 Collecting Data

We tried to keep the log of all the failures we experienced. Many errors have distinct entries in system logs. I have modified the system to keep the logs for much longer than the default. In addition to observing the components under normal use, we subjected some disks to artificial loads to see if it will make any difference in failure rates. This experiment in part augments the low loads we're seeing on the Zoom project.

There are two logs in the system that can be used to observe component behavior. They are the main system log, and the HTTP server log. In addition, I have kept notes of most of the repairs that I performed.

5.1.1 System Log

The main system log (called `/var/log/messages` in FreeBSD) is where all the kernel messages, as well as messages from any process using the `syslog` facility, go. Disk problems usually first show up here as a continuous stream of retries and failures. Processes getting killed due to various reasons are recorded here too. Most of them are segmentation faults due to programming bugs, while there are some others, such as pager faults, that are caused by hardware problems.

```
Jul 24 10:40:09:  (da73:ahc4:0:9:0):  READ(10).  CDB: 28 0 0 50 54 cf 0 0 80 0
Jul 24 10:40:09:  (da73:ahc4:0:9:0):  RECOVERED ERROR info:505546 asc:18,2
Jul 24 10:40:09:  (da73:ahc4:0:9:0):  Recovered data - data auto-reallocated sks:80,12
Jul 24 11:02:11:  (da40:ahc2:0:8:0):  SCB 0x25 - timed out while idle, LASTPHASE == 0x1,
Jul 24 11:02:12:  SCSISIGI == 0x0 SEQADDR == 0x8
Jul 24 11:02:12:  (da40:ahc2:0:8:0):  Queueing a BDR SCB
Jul 24 11:02:12:  (da40:ahc2:0:8:0):  Bus Device Reset Message Sent
Jul 24 11:02:12:  Bus Device Reset Completed.
Jul 24 11:02:12:  (da40:ahc2:0:8:0):  no longer in timeout
Jul 24 11:02:12:  ahc2:  Bus Device Reset delivered.  2 SCBs aborted
Jul 24 11:02:22:  (da40:ahc2:0:8:0):  SCB 0x25 - timed out in command phase,
Jul 24 11:02:22:  SCSISIGI == 0x4 SEQADDR == 0x146
Jul 24 11:02:22:  (da40:ahc2:0:8:0):  BDR message in message buffer
Jul 24 11:02:24:  (da40:ahc2:0:8:0):  SCB 0x8e - timed out in command phase,
Jul 24 11:02:24:  SCSISIGI == 0x14 SEQADDR == 0x146
Jul 24 11:02:24:  (da40:ahc2:0:8:0):  no longer in timeout
Jul 24 11:02:24:  ahc2:  Issued Channel A Bus Reset.  4 SCBs aborted
Jul 24 22:41:29:  (da78:ahc4:0:14:0):  SCB 0x51 - timed out while idle, LASTPHASE == 0x1,
Jul 24 22:41:29:  SCSISIGI == 0x0 SEQADDR == 0xb
```

Figure 5.1: Sample system log

Unfortunately, not all of the system logs are still available. Some of them have been lost to disk failures, while some others have been deleted due to operator carelessness. At the beginning of the project, we were not expecting the logs to be very precious. Table 5.1 shows how long back the logs go on each of our machines. As can be seen in the last line, even though a fair number of earlier logs are lost, there are still an average of over 17 out of 24 months of system logs remaining.

Host	Log From	Months	Host	Log From	Months
m0	Feb 1998	17	m10	Feb 1998	17
m1	Jul 1997	24	m11	Sep 1998	10
m2	Feb 1998	17	m12	Jul 1998	12
m3	Feb 1998	17	m13	Feb 1998	17
m4	Sep 1998	10	m14	Apr 1998	15
m5	Feb 1998	17	m15	Apr 1998	15
m6	Dec 1997	19	m16	Jul 1997	24
m7	Sep 1997	22	m17	Sep 1998	10
m8	Feb 1998	17	m18	Jul 1997	24
m9	Nov 1997	20	m19	Aug 1997	23
			Total		347
			Average		17.35

Table 5.1: Log durations

As an example of what they look like, Figure 5.1 shows some lines from one of the logs for the machine `m0.cs.berkeley.edu`. The year is 1998. To improve readability, I have removed some fields to fit the lines here without excessive wrapping.

The following is what we can deduce from this particular fragment. First, at 10:40 AM, the disk `da73`, with SCSI ID 9 on SCSI bus 4 had a single read error that was successfully recovered by the disk drive's controller. The disk's controller automatically reallocated the data so the sector in question will never be read again. About 20 minutes later, a different disk, `da40`, with SCSI ID 8 on SCSI bus 2, had a few unexpected timeouts. These timeouts eventually led to the SCSI driver issuing a SCSI bus reset in an attempt to restart the disk. The bus reset cleared the problem since there have been no more problems reported until 10:41 PM of the same day. This is a typical "timeout-recovery" sequence that is explained in detail in Section 5.3.3.

After removing non-essential messages such as login history, there were still over 250,000 lines in the system logs. I wrote a program to analyze the logs.

5.1.2 HTTP Server Log

HTTP servers write their own logs. There is one file to record accesses (`httpd-access.log`) and another for errors (`httpd-error.log`). Most of the entries in the error log are for missing files, but there is some information such as when the HTTP servers were restarted. This log would have been useful if the HTTP servers crashed more often. However, as it turned out, our HTTP servers were very reliable and crashed only once, so I didn't have to look into this file for information on restarts.

5.1.3 Repair Records

Since I did not know I would be using the repair records for my dissertation during most of the project, I did not keep a repair log *per se*—however, I believe I have been able to identify most of the repairs I conducted. Other than my own memory, the information came from the following sources.

Mail Messages

I have kept all the mail messages during the course of the project; there were over 4,000 messages originating from myself in the archive, and almost 10,000 total. Since I was the only person conducting repairs on the cluster, I had to only check my own mail. When there was a problem with the cluster, I often sent out mail to people in the Tertiary Disk project detailing the problem and the fix/repair I conducted. Also, when there was an item such as a disk enclosure that had to be sent back to the manufacturer for repair and I asked the secretary to talk to the manufacturer, or when I had to coordinate with a contractor to have the faulty Ethernet cables repaired, the exchange was in the mail archive.

Post-It Notes

When I replaced a bad SCSI disk, I wrote down the date and a brief summary of the problem on a Post-It note and attached it to the component before storing it away. Since these disks were never sent back for repair, I could read off the notes to get a summary of the replacements I made.

Combining these with the log files generated automatically, it was possible to correlate almost all of the failures and repairs with error messages.

5.2 Failure Summary

The frequency of failed and replaced components we have seen in 24 months of operation is shown in Table 5.2. MTBF1 shows the particular component's individual MTBF (mean time between failures) and MTBFn is the collective MTBF of all components of the same type. Values of MTBF are shown in years.

TTR (time to replace) is the approximate time to replace a component in minutes. Most of the numbers were measured by taking the wall clock time during the repairs I conducted. Note that this is just the time it takes to do the actual replacement of parts, intended to be used in calculation of the amount of time the operator has to be physically be present in the machine room. The MTTR (mean time to repair) as often noted in literature is the time it takes the operator to get to the machine room plus the TTR numbers here. Since our system is designed to function with regularly scheduled visits by the operator, we can assume MTTR is half the visit interval for all components in which we have spare parts—the TTR can be ignored in the MTTR calculation, as it is much smaller than the visit interval. For those without spare parts, it will be a function of the visit interval and the time it takes for replacement parts to be shipped.

Infant mortalities are not included in the table. Initially, there were 6 enclosures with bad power supplies, 9 enclosures with bad SCSI buses and about 20 bad Ethernet cables. Also, components without any failures, such as CPU and enclosure fans, are not listed.

Component	Total	Failed	Ratio	MTBF1	MTBFn	TTR
IDE drive	24	7	29.2%	6.9	0.29	60
SCSI adapter	44	1	2.3%	88.0	2.00	10
SCSI cable	39	2	5.1%	39.0	1.00	2
SCSI drive	364	8	2.2%	91.0	0.25	10
Disk enclosure (SCSI)	46	3	6.5%	30.1	0.67	60
Disk enclosure (power)	92	3	3.3%	61.3	0.67	5
Ethernet adapter	20	1	5.0%	40.0	2.00	10
Ethernet switch	2	1	50.0%	4.0	2.00	20
Ethernet cable	24	3	12.5%	16.0	0.67	10

Table 5.2: Frequencies of failures

Number of components and failures we observed are shown. MTBF1 is components' individual MTBF and MTBFn is the collective MTBF of all components of same type. TTR is time it takes to replace failed component. MTBF is in years, TTR is in minutes.

Here are some observations:

- The Ethernet switch has too small a sample size to draw any meaningful conclusion. Nonetheless, it still underlines the importance of avoiding having a single point of failure—a lot of our data would have been unreachable if we hadn't designed the system in a way that there is at least two distinct paths from any data to the outside world.

- On the other hand, the disk enclosures' SCSI bus integrity and IDE hard drives are major causes of concern. As can be seen from the TTR numbers, these are also the two hardest components to replace. We have investigated methods to boot the machines with CD-ROMs as system disks to avoid the IDE hard disk problem altogether.

Another experience worth noting is that it is very hard to diagnose SCSI bus integrity problems. The problem can be in any of the SCSI host bus adapter, cable, disks, enclosures or the terminator. In addition, simply replacing parts one by one is often not enough to find the real cause—for instance, sometimes a faulty disk enclosure backplane will temporarily start working again when a cable is replaced, giving the false impression that a cable is at fault. This behavior is due to the nature of these problems, as they are usually caused by slightly loose connections somewhere in the bus and can come and go with the slightest change. As can be seen in Table 5.2, many of our problems were caused by the enclosures—we did not experience any SCSI disk failure that resulted in the whole SCSI bus being reliable.

- Compared to the IDE drives, SCSI drives have been much more reliable, as 91 years MTBF translates to about 800,000 hours. We suspect the difference is due to two factors: IDE drives being of lower quality in general, and the superior cooling of external disk enclosures that house the SCSI drives.

Also, we have been artificially loading the system to see if will cause more component failures. The load didn't seem to have any effect in component failure rates.

5.3 Hardware Error Details

More details on errors in Table 5.2 are given in this section. The actual methods of repairing various components are also explained.

5.3.1 PC

Except for the internal IDE hard drives where the operating system resides, the computers themselves have been very reliable, with no errors on motherboards, CPU, memory, and other components.

IDE Hard Drive

The internal IDE hard drives turned out to be the only unreliable part of the PCs. There have been 7 disks that had to be replaced. It happened in August 1997, May, July, September, and October (two) 1998, and June 1999.

It is not clear whether there is any correlation between these failures. There appears to be a certain amount of clustering in July–October 1998, as there were 4 failures within a 4-month span while there are only 3 more in the remaining 20 months.

Causes of Failure It is well known that heat and vibration are two external factors that often contributes to disk failures. Vibration doesn't seem to be a problem on our systems as the PC cases and external disk enclosures are both mounted on a well-built rack, but the PC cases do not have adequate ventilation in some areas, particularly around where internal disks are mounted. Therefore I believe heat is one of the reasons why so many IDE hard drives have failed. This theory has also been supported by the fact that in other machines that are not part of this cluster, I have seen several SCSI disks mounted inside the same PC cases fail, while identical disks in external enclosures have not exhibited any failures.

I checked the log of the machine room's air conditioner, but there wasn't anything that indicates an extended downtime anytime in 1998. There are two air conditioners in the room, so even if one of them went down, the room temperature will still be controlled.

A possible explanation of the clustering of the failures is that all IDE hard drives, which were shipped with the PCs to our site together, have been manufactured at around the same time and have "aged" together, causing the failures to cluster somewhat.

The sample size is too small to draw any conclusions, but it still might be a good idea to try purchasing disks from several sources, possibly from different manufacturers, when designing a system such as this one, in order to reduce the risk of having several machines fail during a short period of time.

In addition to the above, there have been two more disks that had several errors that didn't result in an immediate death. One of them had 14 failures in August 1998 (this disk was eventually replaced in October after suffering 6 more failures), and another had 5 failures in September 1998 and 2 more in November. The second disk is still functional to this day, with no more errors logged since then.

Replacing an IDE Drive It takes about one hour for an experienced operator to replace a failed IDE hard drive. About half of that is spent in the physical act of replacing the disk, and the other half in reinstalling the operating system. Here are the steps involved in replacing the disk.

1. Shut down the machine.
2. Disconnect all cables. The SCSI cables have two screws each that have to be loosened by hand. Sometimes the receptacles of these screws, which happen to be screws themselves, come off the SCSI adapters with the cables.
3. Remove the machine from the rack. The machine fits snugly in the shelves as two of the PC cases is exactly the width of the 19-inch rack. However, this also means pulling out the machines requires a fair amount of force.
4. Place the machine on a flat surface and open up two sides of plastic panels that can be easily snapped off.
5. Turn the machine upside down and remove plastic panels from the other two sides. This involves lots of practice and judicious use of two screwdrivers.
6. Remove four screws that are holding the disk drive in place.
7. Remove the IDE and power cables from the drive.
8. Replace the disk with a new one.
9. Reconnect the cables.
10. Reattach the four screws. Two of them are only accessible through holes roughly 2cm by 2cm on the metal case and have to be inserted into screw holes about 5cm away than the metal surface.
11. Put the plastic panels back. Be careful not to break them, as putting them back is almost as hard as taking them off and protruding parts have to be inserted in just the right order while sliding the panels around to hold them together.

12. Push the machine back into the rack.
13. Reconnect all cables.
14. Power up the machine.

From steps 5 through 11, it is obvious that these particular PC cases were not designed with installation and deinstallation of internal disks in mind. If there is any lesson learned from this experience, it is that in order to ease the system administrator's burden, it is important to actually open up the machines and try replacing some parts before purchasing them in quantities. Rack-mount PC cases are generally easier to handle in this respect.

The operating system installation is done by first making an identical copy of the system disk from another machine, and then editing a file to give it the right hostname. The copying takes about half an hour. By keeping spare system disks ready to be swapped in, this wait can be eliminated.

5.3.2 Network

The university has experienced a few network problems that disconnected the entire Berkeley campus from the outside world. Most recent of those occurred on July 1, 1999, when a broken water main caused the machine room holding the network hub to be flooded for a few hours. We have not had any problems between our cluster and the campus backbone. The following are are the problems we had inside our system.

Ethernet Cables

The Ethernet cables for the cluster were cut and installed on site by a contracted technician. Apparently this person had a bad tool and many of the connectors were not clamped properly. Out of the 42 cables, including spares, installed, about 20 were not working from the beginning.

After those were redone, the cables have been fairly reliable, with only three more failures the rest of the way. However, two of those happened when the machine room was reconfigured and the cables had to be unplugged temporarily. The implication of these recent problems is that the "working" cables might also be suffering from bad craftsmanship. From experience, I know that the three out of 24 figure is still too high for Ethernet cables.

Assuming there are enough spare cables installed on the racks, replacing a failed cable can be done just by unplugging and plugging both ends of the cable. In the long run, the unused bad cables will also need to be replaced. With ample supply of cables that the operator can use to replace, it will not take long to replace them.

Network Switch

One of our two 16-port network switches suffered a complete failure on November 1997. All hosts connected to that switch were unreachable until I noticed the problem and moved the cables over to an old network hub which was acting as a spare at the time.

The replacement switch arrived five days after the failure. The physical replacement of the switch was easy, as it involved removing just two screws in the front and pulling out the switch. The switch also had to be initialized by a special software from a portable computer connected to the serial port of the switch chassis. It took about 20 minutes in all.

Network Interface Card

The Ethernet cards on one of the machine suddenly stopped working on August 1997 and had to be replaced. The replacement took about five minutes. The steps are much simpler than those for IDE hard drives described in Section 5.3.1, as only the two easily removable panels have to be snapped off before the card can be removed by loosening two screws.

5.3.3 The SCSI Subsystem

The SCSI subsystem consists of various components. I have categorized them into three parts, the disks themselves, components that support the integrity of the SCSI bus, and the host bus adapters.

SCSI Disk

Our SCSI disks have two vastly different characteristics—reliable hardware and flaky firmware. This section analyzes those in detail.

Removing a SCSI disk involves shutting down the machine, powering off the enclosure and sliding the disk out. Installation of a new disk is just the opposite. New disks have to be mounted in canisters with four screws. It takes about 10 minutes in all.

On a partially related note, in April of 1999, one of the new departmental file servers had problems triggered by a disk failure. Several drives failed in a chain reaction and some filesystems were lost even though the system had RAID-5 protection. The system administrators have provided a detailed write-up on the problem for circulation in the department. According the report, the vendor contended that the problem was caused by disk firmware revision mismatches. However, I suspect there were bugs in the RAID driver or operating system that were triggered by the firmware bugs that compounded the problem to the point that the server had to be taken off-line for a few days and lots of data lost.

Hardware There have been relatively few SCSI disk failures. Only 8 of the 364 disks in the system had to be replaced. These failures have happened in August, September, December of 1997, two in February of 1998, and one each in May, August, and October of 1998. From this data, there does not appear to be any clustering, although we haven't seen any failures for 8 months. Load doesn't seem to have any effect on SCSI disk failures, as only 1 of the 8 failed disks were on the 8 (out of 24) machines with artificial loads.

Figure 5.2 summarizes the disk errors I found in the logs. Three of the disk replacements predate the logs so I couldn't determine whether there were any error messages before the failures for them. Including the total failures, there have been 13 disks with read errors and 10 with write errors. None of them had more than one sector that was unreadable or unwritable. All of those were still unreadable after retries, resulting the kernel returning an error to the application requesting the data. Two disks had both read and write errors, and 9 more could be attributed to bad enclosures. Subtracting those and the ones that were replaced, there are 8 disks that had an unrecoverable error that can't be attributed to anything else, but are still working fine to this day.

There have also been 1,958 recovered read errors on 26 disks and 977 recovered write errors on 2 disks. Most of these were on disks that eventually failed.

Disk ID	Unrecovered errors		Recovered errors		Replaced?	
(machine:disk)	Read	Write	Read	Write	Disk	Enclosure
m0:5			431		Y	
m0:73			1			
m0:76			1			
m1:41	1					
m1:44			5			
m2:4			8			Y
m2:8	1					Y
m2:9			10			Y
m2:10			16			Y
m2:15	1	1				Y
m2:53		1				Y
m2:56			1			Y
m3:13			4			
m4:0	1					
m4:31		1				
m5:0			1313		Y	Y
m5:1			2			Y
m5:4	1					Y
m5:8	1					Y
m5:9	1					Y
m5:32	1					Y
m5:33	1					Y
m5:45		1				Y
m6:1		1				
m7:9	1	1	140	14	Y	
m9:40			2			
m10:41			1			
m10:5			2			
m11:2			1			
m11:9		1	7	963	Y	
m14:1		1	3		Y	
m14:4			1			
m14:42			1			
m14:45			1			
m15:0			1			
m15:1			2			
m16:0	1					
m16:2	1					
m16:43		1				
m17:33		1				
m17:41			1			
m18:5			3			

Figure 5.2: SCSI disk read/write errors

Firmware There appears to be a bug in the firmware of our SCSI disks [GM99]. In short, when there is heavy traffic on the SCSI bus, a disk can lose track of its state and stop responding to commands. There has to be several disks, at least four or five, on the same SCSI bus, all accessed heavily at the same time for this to happen. The only way to get the disk to start responding again is by sending a SCSI bus reset or a power cycle to reinitialize its finite state machine. We had the SCSI driver modified by the FreeBSD SCSI team to try to detect a hanging disk and restart it with a bus reset, which enabled us to fix most of the potential problems, but there are still some instances which couldn't be corrected automatically. As shown in Figure 5.1, these appear in the log as SCSI timeouts.

Almost two thirds of the 364 SCSI disks (235) in the system has suffered from this temporary amnesia. There are 6,540 timeout messages recorded in the logs. Some of these are benign, but others are more serious. The driver gave up on 23 of these cases, causing the kernel to stop sending commands to those disks. The only way to access the disk again was by rebooting the machine. Many others caused access to the disks hanging indefinitely, sometimes hanging or crashing the operating system in the process.

Since this particular problem is quite specific to our system, and can be avoided by thoroughly testing the disks before selecting a particular brand for purchase or upgrading the firmware on disks as necessary, I have chosen to ignore them for the purpose of my simulation.

By the way, in order to upgrade the disk firmware, the machines have to be booted into MS-DOS to run a program supplied by the vendor. Since most of our machines don't have keyboards or monitors connected, this involves moving many disks around and with the SCSI enclosure problems we were having, we were afraid this could cause more problems than it will solve. This awkwardness is the reason why we have not upgraded the firmware on our disks despite occasional crashes it causes.

SCSI Bus

There are three components that can cause SCSI bus integrity problems—cables, enclosures and termination. Termination, either in the form of external terminators or internal to the SCSI host bus adapters, has never failed on our system.

SCSI Enclosures

The SCSI enclosures has been the biggest disappointment of our system. As shipped, nine of the enclosures had SCSI bus integrity problems. During the operation of the system, three more developed similar problems, one in May 1998 and two in November, and had to be sent back to the manufacturer for repairs. There is at least one more enclosure in the cluster that sometimes develops problems when a disk is unplugged and plugged back in.

Replacing SCSI enclosures take at least two people to lift them. It is ideal to have three as the enclosures are a little wider than the racks and thus the racks have to be pushed aside by screwdrivers while inserting the enclosures. In addition, the disks have to be moved from the old enclosure to the replacement, as well as two SCSI cables and four power cables reinstalled.

Three of the disk enclosure power supplies died during operation. Six more were bad as shipped, or died shortly after installation. Since our enclosures have hot-swappable dual power supplies, none of these affected operation of the system. Replacing the power supplies are very simple, involving just one screw and reinstalling the power cable.

SCSI Cables

The external SCSI cables have been fairly reliable, with only two having to be replaced in two years. They can be replaced very easily.

SCSI Host Bus Adapters

Only one of the SCSI host bus adapters failed on our system. One of the chips on the adapter literally burned out. SCSI adapter cards are as easy as the network cards to replace.

5.4 Software Errors

Compared to hardware, the software on our systems has been remarkably reliable.

5.4.1 Operating System

As far as I can tell, there have been no crashes due to a bug in the operating system during the period. Even though I don't know the exact cause of all the crashes and hangs, as many of those didn't leave a log or crash dump that I can analyze, I believe all the OS crashes can be attributed to disk firmware problems for the following reason.

One of the machines, `stampede`, is an NFS server for the entire Tertiary Disk cluster as well as some other machines in the department. In addition to being an NFS server, `stampede` runs an HTTP proxy, a small anonymous FTP server and mailing list server. `Stampede` has been used much more heavily than of the machines in the cluster, was crashing almost daily when it had the same disks as others. `Stampede` has not had a single crash in the two years since I replaced the disks with those from another manufacturer.

Also, I have been watching the console messages of some machines on their serial ports as they either hung or crashed. In every one of the dozen or so cases I have observed, there was a message indicating a disk firmware problem just before the machine hung or crashed.

5.4.2 Application

The HTTP server has died 11 times during the 30-month period. 10 of those were bus errors and 1 was an abort signal. All of the bus errors happened when the operating system was upgraded while the HTTP servers were running. Such crashes usually result when a shared library is changed while an application is running—the in-memory version of the shared library can be out of sync with the version on disk. Since the machines are subsequently rebooted, these errors were not repeated.

As the frontends were instructed not to forward requests to machines that were being upgraded, none of these were visible to the users. This leaves only one crash, the one caused by the abort signal, as a legitimate crash of the HTTP server caused by a software bug.

This result is in stark contrast to Jim Gray's 1985 study on failures, in which he observed that 25% of system failures are caused by software [Gra85]. This difference is probably due to both the operating system and application on our site being relatively stable versions. If we counted the number of application crashes during the period when it was still under heavy development, the system would have had a much higher software failure rate.

5.5 Summary

We have been observing our system since we constructed it. Records of replacements, which, with system logs of the operating system, were analyzed to determine the failure rate of each component. Although we have lost some of the logs, I believe we were able to determine the failure rates with good accuracy.

The SCSI disks' failure rates were consistent with what were advertised by the manufacturer; those of IDE disks and external disk enclosures were not. There didn't appear to be any correlation between separate failures. The operating system and applications were surprisingly reliable.

The data collected in this chapter is used in the next chapter as default parameters for the simulator to determine the availability of our system and variations of it over a long period of time.

Chapter 6

Validation

I ran experiments to validate the feasibility of our approach. The objective was to prove that the system will actually run continuously and flawlessly in the existence of a variety of failures. A simulator, in conjunction with an event generator, was used to calculate the expected downtime and time between failures depending on several parameters such as component failure rates and repair interval. By running a simulation over a long period of time, I was able to obtain a good estimate of availability and times before failures.

The simulator models the system as a directed acyclic graph (DAG) of nodes and arcs. A *node* represents either a hardware component, such as a disk, or an abstract concept, such as “data” or “users”. The system is functional if all the “data” are connected to the “users”. Using this method, I reduced the availability question into a simple graph connectivity problem of the DAG. I believe this novel approach is an advance over a more detailed model of the system.

Figure 6.1 shows a simple example. Data is mirrored on two disks on two different machines. Users are on the same local Ethernet segment as the servers. Power supply and cable failures are ignored. There are two paths from Data to Users. If either one of Route A or B is complete, the system is functional. However, if both Route A and B are disconnected—for instance, if PC A and Disk B are broken at the same time—then the system has suffered an outage.

There are two types of nodes: *Data* nodes allow data to flow through them; *power* nodes are power supplies that don’t carry data but instead control functionality of other nodes. Nodes associated with hardware components will sometimes break and affect the availability of the system. Table 6.1 classifies various types of simulator nodes. Various abstract concepts and their implementation in the simulator are explained in Section 6.1.2.

Type	Represents	Examples
Data	Hardware	SCSI disk, SCSI adapter, SCSI cable, Ethernet cable, Ethernet switch
	Abstract	data, users, striped set, dataset
Power	Hardware	UPS unit

Table 6.1: Simulator nodes

In my simulator, an *arc* generally denotes a connection where data can flow from one

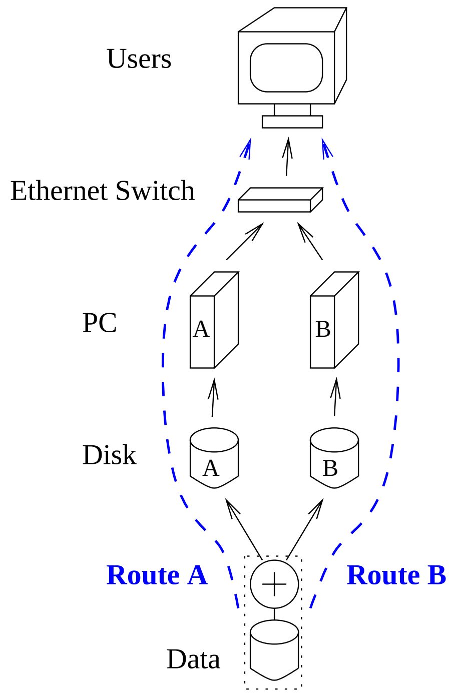


Figure 6.1: A simple connectivity graph

node to another. Some arcs convey other special information, such as "electric power". Arcs never break in my simulator. Note that "cables" in our system are represented as nodes, not arcs; for instance, Ethernet cables connecting Ethernet adapter cards on individual PCs to the central switch are implemented as in Figure 6.2. By using nodes to represent cables, it became possible to simulate cables with multiple endpoints, such as double-ended SCSI buses.

The failure data mentioned in Chapter 5 was used to design the input to the simulator. In addition to running the simulator on a configuration similar to ours, I have also tried different configurations, such as with or without double-ending, different number of disks per PCs, and so on. Some simulations were done to answer "questions" in the form of English sentences—for instance, "what will it take to build a system with 99.999% availability?" They are all described later in this chapter.

6.1 Simulation Setup

I wrote an event-driven simulator to use the data we collected to experiment with various design parameters. The parameters include: different repair intervals, systems with and without disk striping, systems with and without fast reboot optimizations, different disk failure rates, and so on. The simulator also allowed us to investigate radically different designs, such as having significantly more disks per machine or not having double-ending.

The simulation setup consists of two parts: the event *generator* and the *simulator*. The generator generates random events based on parameters given to it. The simulator reads the output

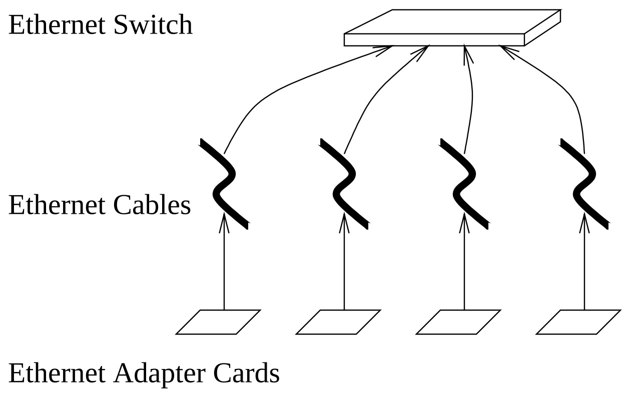


Figure 6.2: Ethernet cables

of the generator and emulates the system's behavior. The simulator is designed to analyze the flow of data in the system as a graph connectivity problem and detect possible outages.

6.1.1 The Generator

The design goal of the generator was to be able to generate events in a realistic manner according to the reliability observations we have made on our system over the course of the project. The events follow the exponential distribution and are not correlated to each other.

Generator Parameters

Table 6.2 shows general command-line options to the generator. The random seed is used as an argument to the `srandom()` function call to set the initial seed to the pseudo-random number generator. By setting this to a specific value, we can repeat the simulation with identical set of events. By default, the pseudo-random number generator is initialized by the process ID of the generator process, making the output different every time when it is run repeatedly. The `-size` option will dramatically expand the size of the system; it is used in Section 6.3.11.

Table 6.3 shows the options to specify the number of components and their average lifetime. Note that the average lifetimes are for individual components; for example, with 512 disks, there will be approximately five failures per year if each disk has a 90-year average lifetime with exponential distribution. It is also possible to specify the number of each component. Numbers of some components are related; for instance, the number of Ethernet adapters are equal to the number of PCs, and the number of SCSI cables are twice of the number of disk enclosures because of double-ending. The related components are denoted by "$\times n$" in the table, and their numbers can be changed together by specifying only one option. I rounded up the default number of components to the closest power of 2 from those of our system to make it easier to test different configurations.

As mentioned above, most of the default failure rates are taken from data described in Chapter 5. The exception to this is the Ethernet switch, which had one failure out of two components, or a 50% failure rate in three years. It is obvious that the sample size is too small to draw any meaningful conclusions about them. I assigned a default failure rate of zero to Ethernet switches,

Option	Description	Default
`-d`	Duration of simulation	100,000 years
`-s`	Random seed	process ID
`-ri`	Repair interval (visit frequency)	1 week
`-rt`	Reboot time	60 seconds
`-size`	Multiply size of entire system by integer	1

Table 6.2: Generator general options

	Number		Average lifetime	
Component	Option	Default	Option	Default
Disk	`-nd`	512	`-da`	90 years
PC	`-npc`	32	`-pa`	7 years
Operating System	`-npc`	32	`-osa`	∞
UPS	`-nups`	16	`-ua`	∞
SCSI adapters	`-nenc` $\times 2$	64	`-saa`	90 years
SCSI cables	`-nenc` $\times 2$	64	`-sca`	40 years
Disk enclosures (SCSI)	`-nenc`	32	`-esa`	30 years
Disk enclosures (power)	`-nenc` $\times 2$	64	`-epa`	60 years
Ethernet adapters	`-npc`	32	`-eaa`	40 years
Ethernet cables	`-npc`	32	`-eca`	16 years
Ethernet switches	`-nes`	2	`-ewa`	∞

Table 6.3: Generator component options

```
B/R name     number  time
---------------------------
B etheradapter  11  2546387
R etheradapter  11  3024000
B PC             0  4345488
R PC             0  4838400
B ethercable    16  5459964
R ethercable    16  6048000
B disk         153  9387963
B encpower      20  9436074
R disk         153  9676800
R encpower      20  9676800
B encscsi       26 10826452
R encscsi       26 10886400
```

Figure 6.3: Sample generator output

along with UPS units, which did not fail during our project's duration, as I did not want to assign an arbitrary number for something we did not know. The effect of these parameters, as well as others, are explained in Section 6.3. The failure rates of Ethernet switches and UPS units did not have much effect on the the overall performance of the system.

We did not observe any operating system crashes, but by specifying the `-osa` option, it is possible to simulate randomly crashing operating systems. The `-rt` option in Table 6.2 can be used to specify the amount of time it takes for the operating system to recover from a crash with a reboot.

All failed components except the operating system are assumed to be repaired when the scheduled operator visit occurs—this is not an unreasonable assumption since all that are needed are enough spare components. The repair interval is specified by the `-ri` option.

Generator Output

Figure 6.3 shows some lines from a sample run of the generator. The fields are, from left to right,

```
break/repair component-name component-number time
```

The first field is "B" when a component breaks and "R" when it is repaired. The next field is the type of the component and and integer identification number. The last field is the number of seconds since the simulation has started. For instance, the first line of Figure 6.3 means "Ethernet adapter card number 11 broke 2,546,387 seconds after the simulation was started."

6.1.2 The Simulator

The other program, the simulator, reads the output generated by the generator and determines the system's behavior. The simulator keeps track of the failures and notes the fact when some

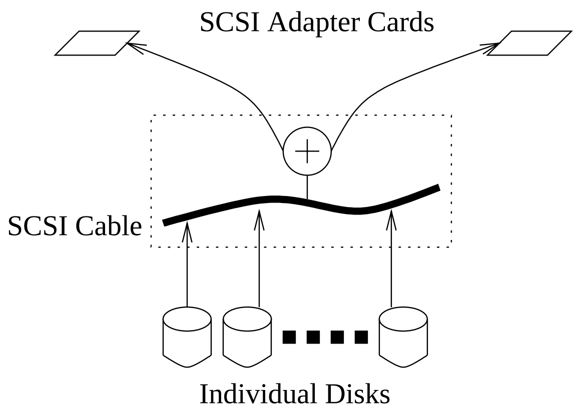


Figure 6.4: OR-node

of the data becomes unreachable. At the end, it will print out a summary of the system's availability.

Simulator Architecture

The simulator treats the system as a DAG. A node of the graph is either a hardware component, such as a disk, or an abstract concept such as "data" or "users". Some hardware components, such as power supplies, have more than one corresponding node in order to model the behavior better. An arc generally denotes a connection where data can flow from one node to another. Some arcs convey other special information, such as "electric power". As mentioned above, the system is functional if all the "data" are connected to the "users".

Nodes There are two types of nodes in the system. Most of the nodes are *data nodes*, which denote places through which data can flow. There is also a special type of node for uninterruptible power supplies, called *power nodes*. Power nodes don't handle data; they supply power to data nodes.

Data Nodes Data nodes carry data. There are two types of data nodes, OR-nodes and AND-nodes, depending on how failures of a subset of its descendants are handled.

OR-Nodes The double-ended SCSI cable in Figure 6.4 is an OR-node. From an OR-node, the data can flow in either direction. If the data eventually gets to the destination from any of the direct descendants, the OR-node is said to be connected to the destination. I use the "plus-in-circle" symbol to denote an OR-node in the figures.

AND-Nodes The striped virtual disk in Figure 6.5 is an AND-node. An AND-node is connected to the destination only if all of the immediate descendants of the node are connected to the destination node. In particular, if any of the immediate descendant of an AND-node fails, the AND-node itself will be disconnected. I use the "times-in-circle" symbol to denote an AND-node in the figures.

Most data nodes of the Tertiary Disk system graph are OR-nodes. The only components that are pure AND-nodes in our configuration are data and striped sets. Note that if a node has only

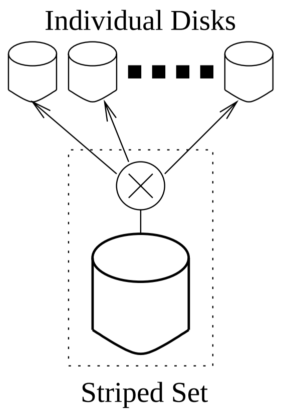


Figure 6.5: AND-node

one descendant, it can be called either an OR- or AND-node and have the exact same behavior. In such cases, I have omitted the circle symbols.

Power Nodes Power nodes represent uninterruptible power supplies. They also have arcs that connect them to other components but these arcs don't represent flow of data—they are power cables. If a power node goes down, it will take down all the components connected to it.

Simulator Details

As mentioned in the previous section, the simulator builds a DAG and analyzes its connectivity to determine the system's functionality. However, the DAG does not simply reflect the connectivity of the hardware of the system. Some of the nodes, such as *data* and *users*, are abstract concepts that don't have corresponding hardware; some others, such as redundant power supplies, require special handling to model their behavior with a simple DAG. This section explains how some parts of the system are represented in our simulator.

Data The *data* is divided into *datasets* and spread around the entire cluster. By default, datasets are mirrored and reside on striped sets by default. If any of the datasets are unavailable, the system is said to have suffered an outage.

To model this, I use a single AND-node called "data" that has all the datasets as immediate descendants. The connectivity from the datasets depend on the configuration of the system. Figure 6.6 shows how the data is connected to the striped sets in the default configuration using mirroring. Note that the data is an AND-node while the datasets are OR-nodes.

Striped Sets As shown in Figure 6.5, a striped set of multiple disks can be represented as an AND-node with all the disks in the set as its descendants. A concatenation set, where the individual disks are serially concatenated instead of having data striped across them, is similar—the only difference with a striped set is the actual data layout on the physical disks, which has no bearing upon the connectivity of the system.

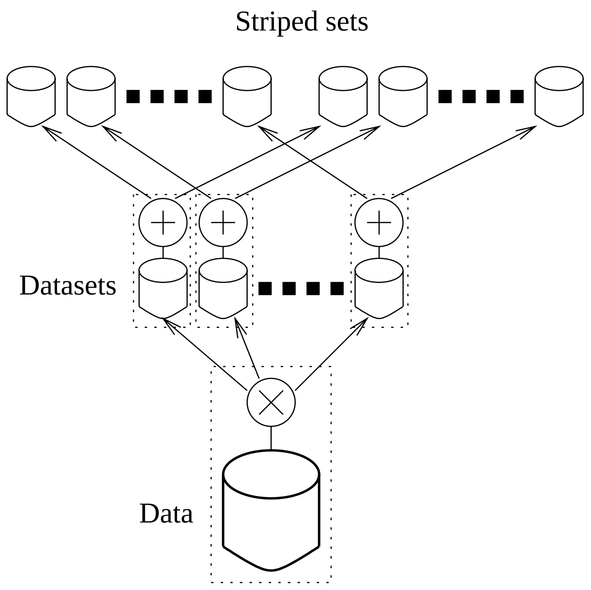


Figure 6.6: Data, datasets and striped sets

It is worth pointing out that, as an extension, a parity-based RAID-5 array can be represented as a variation of an AND-node where *all but one* of the descendants have to be connected for the node to be functional. Another way to look at this is to count the number of descendants that have to be connected for a certain node to be connected. Let's call this number the *connectivity factor*. For a node with N descendants, an AND-node has a connectivity factor of N, an OR-node has 1 and a RAID-5 node has $N - 1$. By making this an arbitrary number, we can simulate other designs such as P+Q RAID, which has a connectivity factor of $N - 2$. See Table 6.4 for a summary.

Name	Connectivity factor	Comment
Mirror	1	OR-node
Striped set	N	AND-node
Concatenated set	N	AND-node
RAID-5	$N - 1$	
P+Q RAID	$N - 2$	

Table 6.4: Connectivity factors

Power Failures A power supply is represented by adding an extra node behind an existing one as in Figure 6.7. This extra node is added even if we don't simulate failures for this particular type of power supply to make the graph easy to construct and understand. Power supply nodes are connected to the UPS (power) nodes.

Redundant Power Supplies Redundant power supplies such as those found in our SCSI enclo-

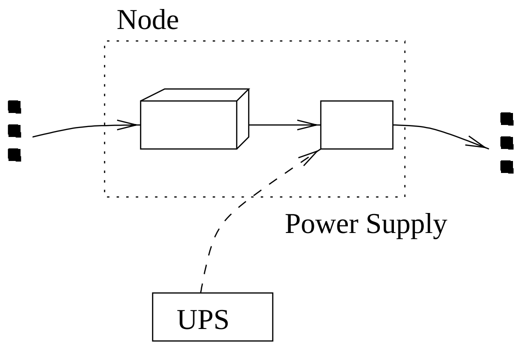


Figure 6.7: Power supplies

sures can be emulated by dividing the nodes into sub-nodes as shown in Figure 6.8. The enclosure itself is duplicated in two places—the in-node (frontend) and the out-node (backend). Data flows into the in-node and out of the out-node. Between the in-node and the out-node are the power supplies. The dotted line that surrounds the in/out-nodes and power supplies is the enclosure as seen from the outside.

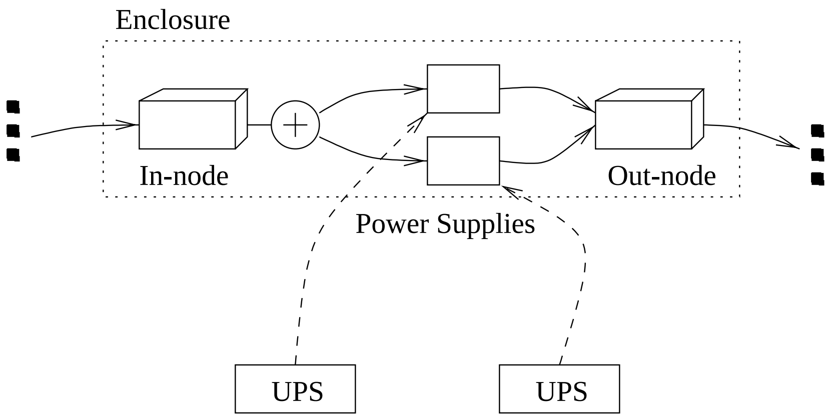


Figure 6.8: Redundant power supplies

The “real” enclosure can be thought as being either the in-node or the out-node, but it is probably easiest to consider the entire contents of the dotted lines to be the enclosure. The important characteristic of this combined node is that even if either of the power supply or UPS fails, data still flows in and out of the enclosure, while if both power supplies lose power, the enclosure will be dysfunctional.

In my simulator, an enclosure failure is treated as an in-node failure.

Simulator Parameters

By default, the simulator will construct a system with 32 PCs and 512 disks in a double-ended configuration. For simplicity, 16 disks are put in one enclosure, although a real wide-SCSI bus can only address 16 devices, and thus cannot support 16 disks in addition to the two SCSI adapters required for double-ending on one bus. Double-ending and mirroring can be turned off by command-line options. Note that the `-mirror` option actually specifies the number of replicated copies of data: `-mirror 1` will turn mirroring *off;* `-mirror 4` will specify a 4-way mirror.

Table 6.5 shows all the parameters to the simulator. By default, the simulator runs until the input from the generator is exhausted, but it can be stopped before the end of input by specifying the `-d` option. Verbose descriptions of each outage can be generated with the `-v` option. It is also possible to specify numbers of each individual components. As was the case with the generator, some options change the number of more than one component; for instance, if you specify `-nenc 16` with double-ending, there will be 16 enclosures, 32 enclosure power supplies, 32 SCSI cables and 32 SCSI adapters. The `-size` option will make the entire system larger by simply multiplying the number of all components.

Option	Description	Default
`-d`	Duration of simulation	until end of input
`-nde`	Turn off double-ending	false (enable double-ending)
`-v`	Turn on verbose output	false
`-mirror`	Mirroring factor	2
`-nd`	Number of SCSI disks	512
`-nccd`	Number of disk stripe sets	32
`-nenc`	Number of disk enclosures	32
`-npc`	Number of PCs	32
`-nups`	Number of UPS units	16
`-size`	Multiply size of entire system	1

Table 6.5: Simulator parameters

Simulator Output

The simulator, without the `-v` option, outputs just one line of summary information such as the following:

```
185 breaks (9678 h) ave 52 h (0.97 h/y), avail:  99.98895%, MTBF 54.05 y (473514 h)█
```

This line means the simulator detected 185 outages totaling 9,678 hours over 10,000 years for an average of 52 hours per outage and 0.97 hours of outage per year. The availability of the system was 99.98895%, and the mean time between failures (MTBF) was 54.05 years, or 473,514 hours.

The most interesting numbers here are the MTBF and outage hours per year. One goal of our system design was to create a large-scale storage system with an MTBF figure comparable to a single disk. With manufacturers' quoted MTBF figures for high-end SCSI disks being a few hundred thousand to a million hours (about 30 to 110 years), a number with six to seven digit hours here means our system has attained the goal.

On the other hand, the outage hours per year is more intuitive and is easier to understand and compare. We cannot compare this number with disks—once disks fail, they generally stay failed, so disk drive manufacturers don't quote outage hours per years for disks—but we can compare it with high-availability systems. For instance, Jim Gray mentions in a paper that, in 1989, Tandem systems had system MTBF of 21 years and a MTTR of 10 hours, which translates to about 0.5 hours of outage per year [Gra90]. Also, some manufacturers claim 99.999% availability for their server systems. That translates to about 0.09 hours, or 5 minutes, of outage per year.

6.2 Alternatives to Simulation

Although the highly redundant design makes our system hard to approach analytically, there are alternatives to event-driven simulation. One of them is to use a Bayesian network. Although Bayesian networks are primarily intended for static probability analysis, it is possible to employ them in a time-based problem like ours. However, for a system with a few hundred nodes, the computational complexity of estimating the availability using Bayesian networks is very high.

There are some additional problems with using Bayesian networks to evaluate our system. One is that the operator visiting to repair failed companents means we can no longer calculate the probability of two components failing at the same time as a product of the probability of each component failing; the failure intervals of components are no longer independent. Also, with an event-driven simulator, it is trivial to explore additional repair policies—for instance, the operator visits only when there are *two or more* failed components—but such additional dependencies will make the transition matrix of a Bayesian network much more complicated. Therefore, we decided to use an event-driven simulator to evaluate the system.

6.3 Simulation Results and Exploration

The simulation was run under several conditions. First, the behavior of the base system, with hardware configuration and failure parameters similar to the system we have, is presented and analyzed. Subsequently, variations to the systems are introduced.

The simulator was used to answer several questions to further understanding about our design and choice of architecture. Here are the questions with their corresponding section numbers.

6.3.1 What are the availability characteristics of our default system?

6.3.2 What are the effects of individual component failure rates?

6.3.3 How much time does the operator actually spend conducting repairs? Is there some way to schedule repairs more efficiently?

6.3.4 How much will the repair interval affect reliability?

6.3.5 What is the effect of disk striping?

6.3.6 How valuable is mirroring?

6.3.7 How about RAID-5?

6.3.8 How valuable is double-ending of SCSI disks?

6.3.9 Will operating system crashes affect availability? How much will reboot time after a crash affect availability?

6.3.10 What is the effect of decreasing or increasing the number of disks per PC?

6.3.11 How will a system 10 times as big as the current one behave? How can I make it as reliable as the current one?

6.3.12 What will it take to build a system with 99.999% availability given the current component failure rates? What about 99.9999%?

The following sections will explore these questions in detail.

6.3.1 Overview

"What are the availability characteristics of our default system?"

The default system had a MTBF of 630,000 hours (70 years) and an availability of 99.990%, or 0.8 hours of downtime per year. The average downtime per failure was 57 hours. In other words, on average, the system will go down once every seven decades, and when it's down, it takes about two and a half days on average to recover. Note that the operator comes in to repair any failed component every seven days—the time to recover is obviously quite strongly affected by the repair interval, as can be seen below.

6.3.2 Failure Rates

"What are the effects of individual component failure rates?"

The failure rates of each component has differing effects on the behavior of the entire system. Let's take a look at how they affect the overall system reliability.

SCSI Drives

By far the largest portion of failures come from SCSI drives. Figures 6.9 and 6.10 show the relationship between SCSI drive failure rates and availability. As mentioned above, with the disk failure rates we observed, one failure every 90 years, the outage per year is just under 1 hour, and the MTBF is 70 years. Manufacturers often quote 1 million hours MTBF for their high-end SCSI drives; that translates to about 114 years, so our whole system MTBF is close to that.

As can be seen from the graphs, if the disks become more reliable in the future, the MTBF of the system could go up to over 1,000 years and outage down to less than 0.1 hours per year before other factors start dominating the characteristics of the system. On the other hand, with unreliable disks, the system reliability could go down very quickly. For instance, if the disks have only one tenth average lifetime than what the manufacturers claim, the outages will be 44 hours per year (99.5% availability) and the MTBF will be about 1.3 years.

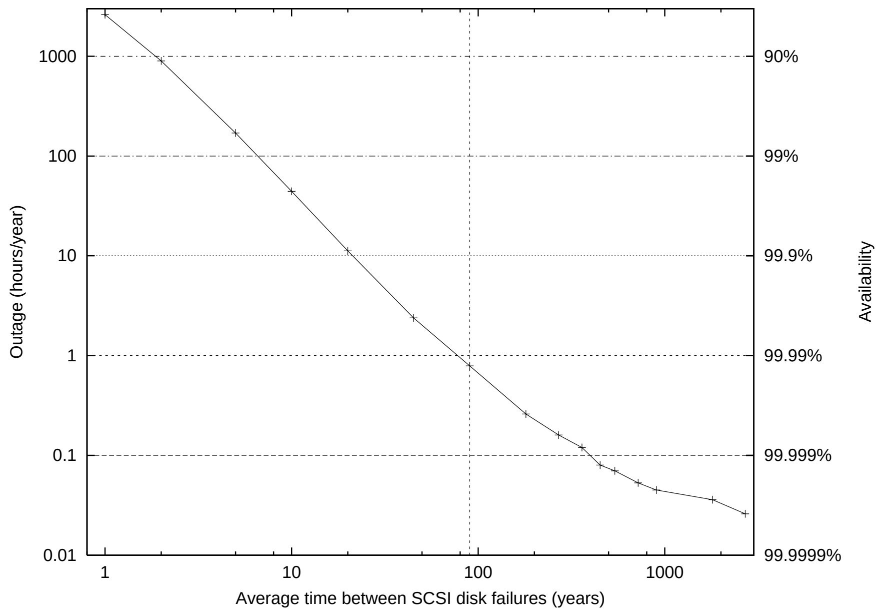


Figure 6.9: SCSI drive failure rates and availability
(Dotted line indicates observed value)

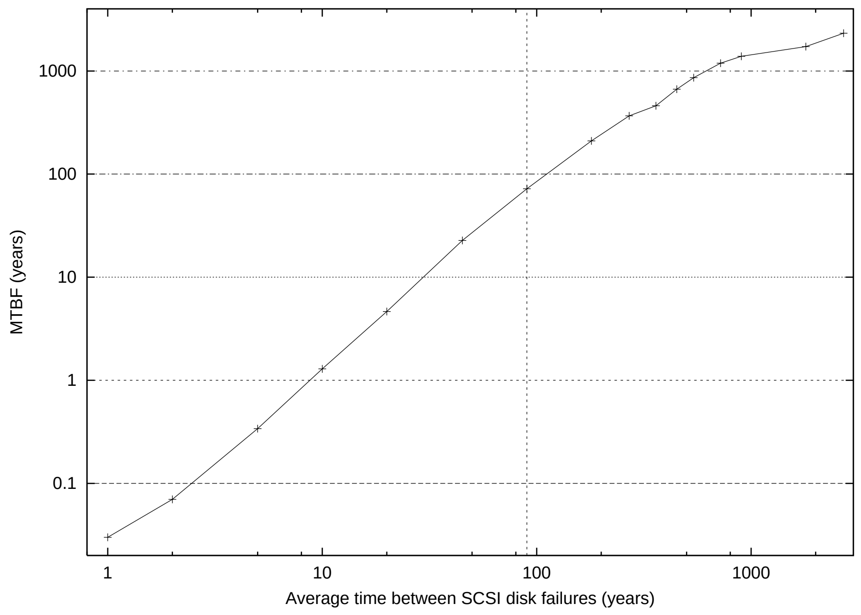


Figure 6.10: SCSI drive failure rates and MTBF
(Dotted line indicates observed value)

IDE Drives

Figure 6.11 shows the relationship between the PC's internal IDE drive failure rates and the outages. The disk failure rates we observed was one failure every 7 years. Manufacturers usually quote about 300 thousand hours as IDE drive MTBF, which is about 34 years. Since we are already in a fairly flat part of the graph, our higher failure rate didn't affect the availability much. However, without double-ending, the IDE drive failures have a profound impact on system availability; see Section 6.3.8 for details.

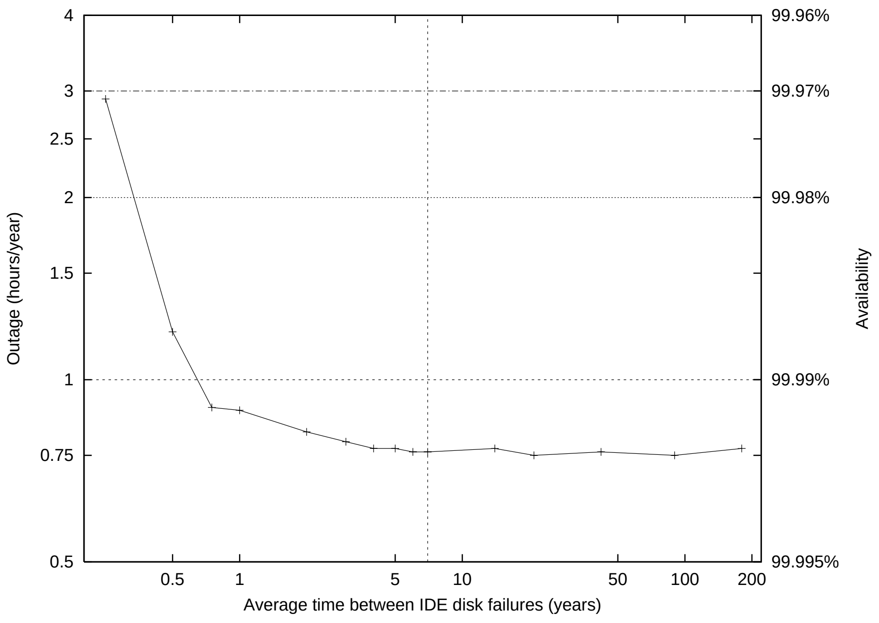


Figure 6.11: IDE drive failure rates
(Dotted line indicates observed value)

UPS

Figure 6.12 shows the relationship between the UPS failure rates and availability. We didn't observe any UPS failures so the range of failure rates is totally arbitrary. It is apparent from the graph that UPS failures does not have a large impact on our system, as long as MTBF is greater than 1 year.

Ethernet Switches

Figure 6.13 shows the relationship between the Ethernet switch failure rates and availability. We did observe one failure out of two switches in three years, with an average component MTBF of 6 years. It is apparent from the graph that Ethernet switch failures does not have a large impact on our system, which is a testimony to our double-ended configuration.

Others

As can be seen in Figure 6.14, the failure rate of SCSI adapters seem to have a very little effect on overall MTBF. Our observed rate of once per 90 years is already in a very flat position of

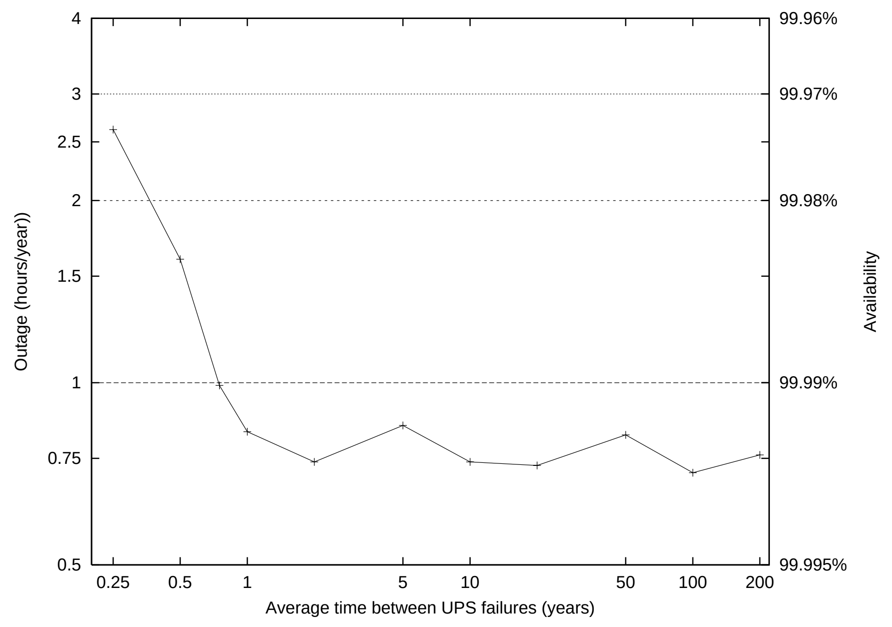


Figure 6.12: UPS failure rates

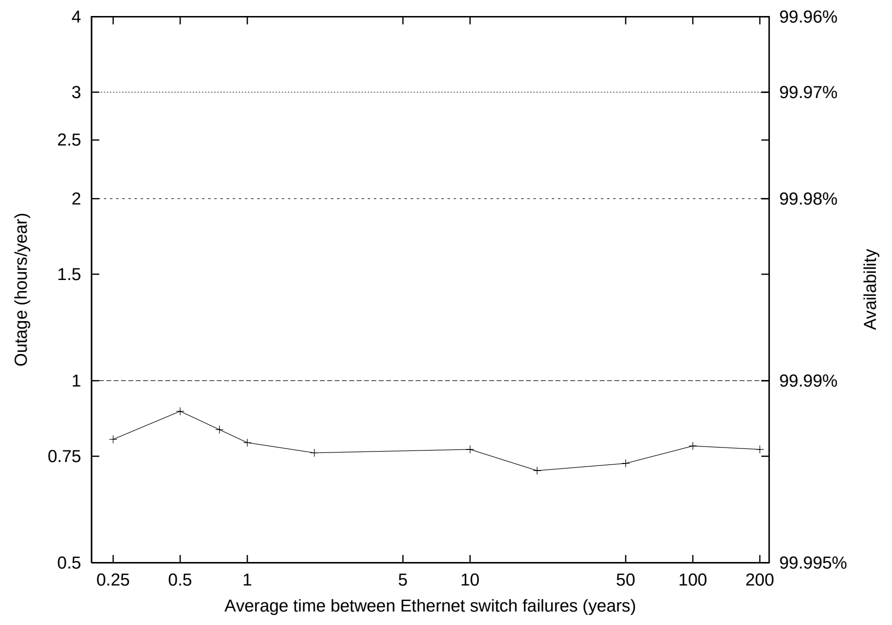


Figure 6.13: Ethernet switch failure rates

the curve. The same can be said for all other components, namely SCSI cables, disk enclosure power supplies, disk enclosure SCSI bus integrity, Ethernet adapters and Ethernet cables.

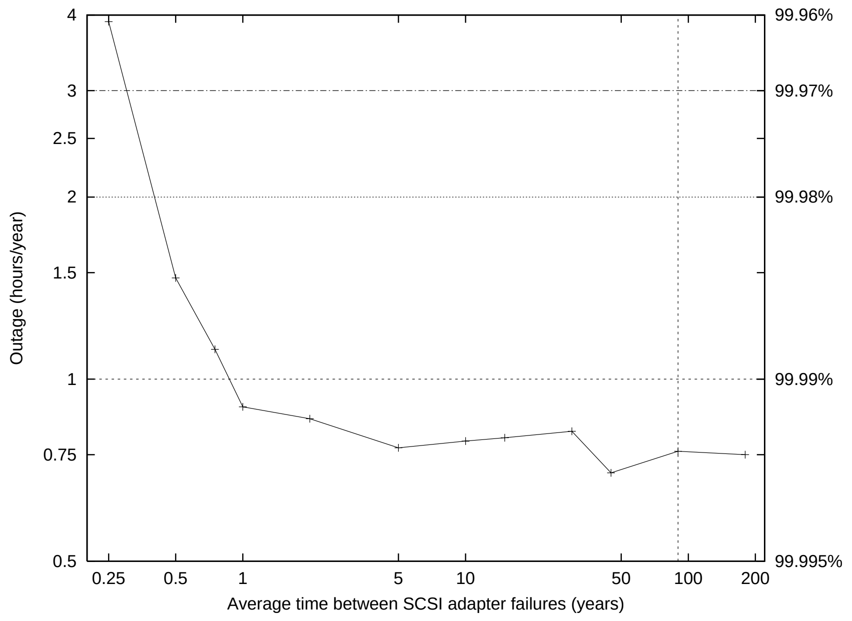


Figure 6.14: SCSI adapter failure rates
(Dotted line indicates observed value)

Figure 6.15 shows the effect of SCSI drives, IDE drives and SCSI adapters. Since each component has different observed lifetimes, I normalized the horizontal axis to align their average lifetime along our observed values. Thus, the dotted line marked "1" is our observation. It can be seen from this graph that any difference that reliability of IDE drives or SCSI adapters make are insignificant compared to SCSI disks.

To summarize the effect of component failure rates on system reliability:

- SCSI disks have by far the largest effect on system reliability, with an almost linear relationship.
- We couldn't get enough data on UPS units and Ethernet switches, but judging from the shapes of curves, they don't seem to matter much.
- No other component has a significant effect on the system reliability either.

6.3.3 Operator Cost

"How much time does the operator actually spend conducting repairs? Is there some way to schedule repairs more efficiently?"

Figure 6.16 shows how long it takes for an operator to do the job. The two lines are minutes per visit that the operator spends conducting repairs. One of them is divided by the total

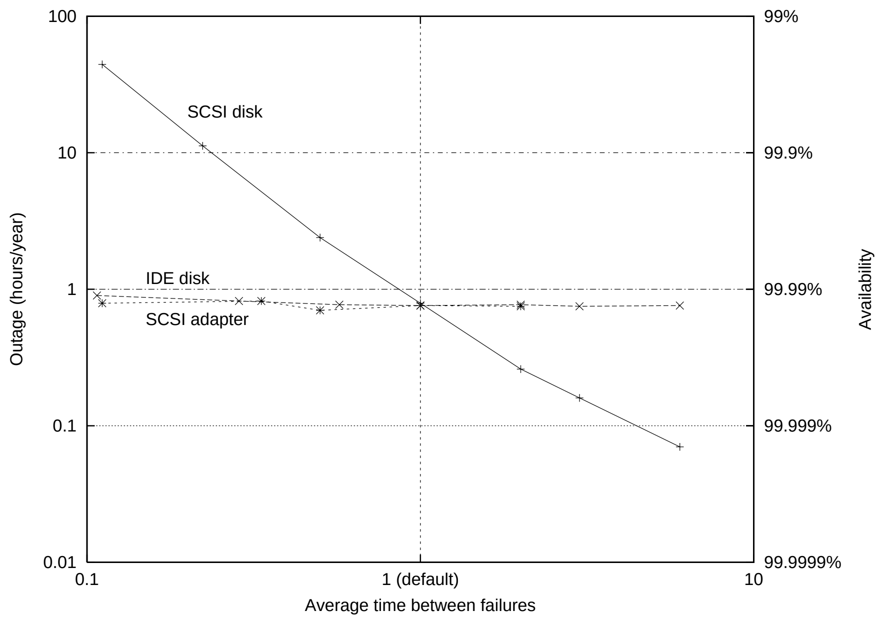


Figure 6.15: Effect of component failure rates

number of repair intervals; the other one is divided by the number of times that the operator actually has something to repair. As can be seen from the upper line, the latter number tails off at about half an hour per visit.

Figure 6.17 shows the ratio of repair intervals during which a failure has occurred. With a one-week repair interval, the operator has to conduct a repair only 28% of the time. The time between failures on our system is about 21 days. In other words, a repair is necessary once every three weeks, and as the repair interval decreases, the ratio of visits that are actually necessary also decreases quickly.

See the next section for discussion on how to efficiently schedule repairs to reduce the cost while maintaining high availability.

6.3.4 Repair Interval

"How much will the repair interval affect reliability?"

Figures 6.18 and 6.19 show the effect of varying the repair interval. As mentioned before, with the standard 1-week interval between operator visits, the MTBF is about 70 years. By more frequent visits, we can increase this to a much greater number and reduce the downtime. For instance, with a once-per-day visit, the downtime per year will be 0.03 hours, or 2 minutes, per year, and the MTBF will be 260 years; availability is 99.9997%. On the other hand, with a once-per-month visit, the outage per year will rise to 15 hours. That is more than a fifteen-fold increase compared to systems with weekly visits. Also, the MTBF will go down to 15 years. The MTBF might be still acceptable in some cases, however, the outages, averaging just under 10 days, could last as long as a full month.

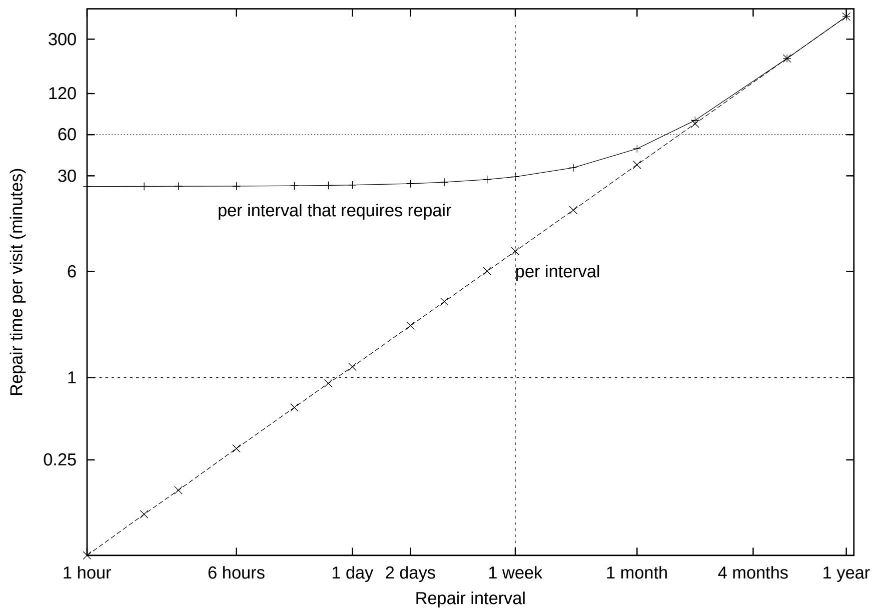


Figure 6.16: Operator time

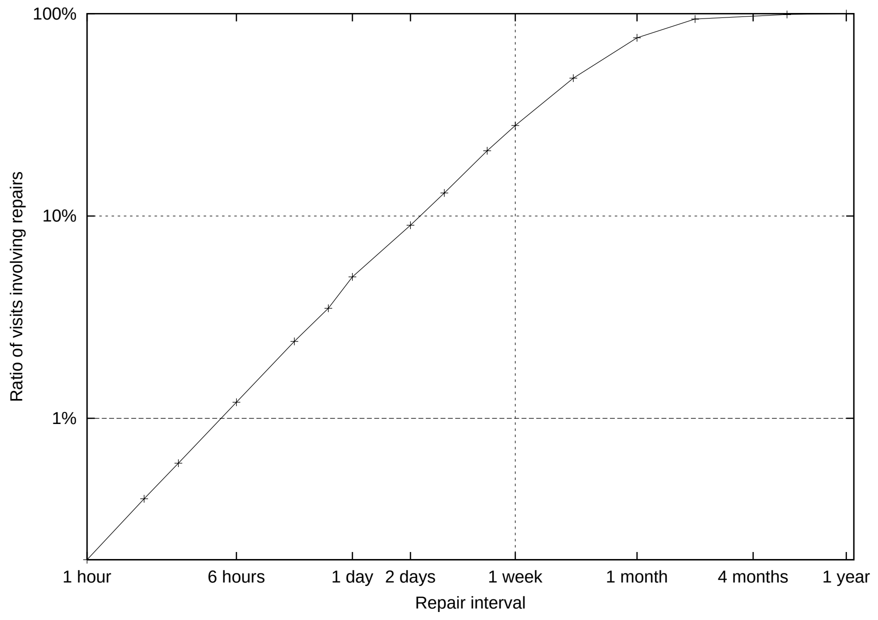


Figure 6.17: Visits that result in repair

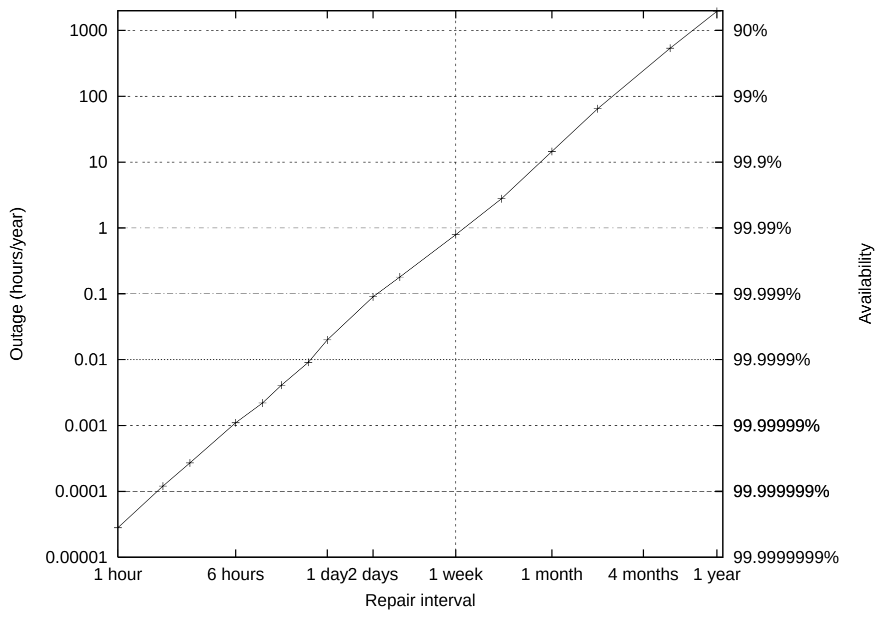


Figure 6.18: Repair interval and availability
(Dotted line indicates default)

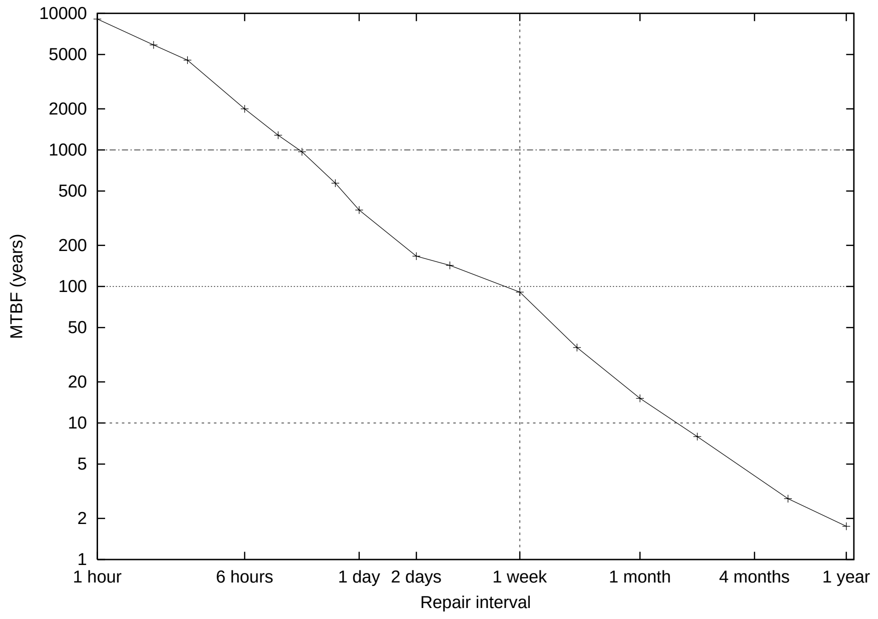


Figure 6.19: Repair interval and MTBF
(Dotted line indicates default)

With a once-per-year visit, the system will have a MTBF of less than 2 years. That means the chance that the system is operational when the operator shows up is less than 50%. Also, the outage per year is close to 2,000 hours—almost 3 months.

According to these numbers, by picking one week at the beginning of the project, it appears we have made an extremely lucky guess for what an repair interval would be for our system that keeps administration costs down while having excellent availability. If we wanted the highest availability, we would hire people to maintain the system 24 hours a day. Such vigilance would probably cut MTTR to 2 hours, yielding an availability of 99.999999%. When computer manufacturers claim 99.999% availability, they usually assume constant attention to the system.

One interesting observation is that, combined with the results from the preceding section, it should be able to effectively reduce the repair interval without actually having the operator visit the site every time. For instance, according to Figure 6.18, an 18-hour repair interval will increase the availability of the system to 99.9999%. However, Figure 6.17 show that repairs are necessary only 3.5% of the time with the 18-hour interval. Note that the "18-hour repair interval" here means the operator has to come in within 18 hours of the failure, and not necessarily every 18 hours if there is nothing to fix.

With this in mind, consider the following arrangement with the operator that we call *Pager Arrangement*, as opposed to the traditional *Fixed Interval* method:

- The operator carries a pager.
- If the system pages the operator between 7 AM and 3 PM, the operator should come in and conduct the repair before 5 PM on the same day.
- If the system pages the operator between 3 PM and 7AM in the next morning, the operator should come in and conduct the repair at 9 AM.

According to the Pager Arrangement, the operator will only have to come in once every 21 days, or three weeks, to conduct necessary repairs. Figure 6.16 also shows that the time it takes for the operator to do the job is roughly constant at half an hour for intervals less than a week.

Under this kind of arrangement, the availability will be slightly better than the Fixed Interval 18-hour repair interval, since this is in effect a combination of 8-hour and 18-hour repair intervals. See Figure 6.20 for a comparison between the Fixed Interval and Pager Arrangements. In the Fixed Interval Arrangement, any failure that happened during an 18-hour interval will be fixed at the end of that interval; with the Pager Arrangement, any failure in the 8-hour period between 7AM and 3PM gets fixed at 5PM with a maximum delay of 10 hours, and any failure in the 16-hour period between 3PM and 7AM will be repaired at 9AM with a maximum delay of 18 hours.

As a variation of the Pager Arrangement, the operator can remotely check the status of the system at 3PM and 7AM, and come in at 5PM and 9AM, respectively, if there is a need for repair. In this case, the operator needs a network connection instead of a pager. The availability of this system will be identical with one with the Pager Arrangement.

6.3.5 Disk Striping

"What is the effect of disk striping?"

As mentioned in Section 3.2.2, we use disk striping to combine multiple disks on one PC into one large virtual disk. This merging is a tradeoff between operator cost and reliability: it is

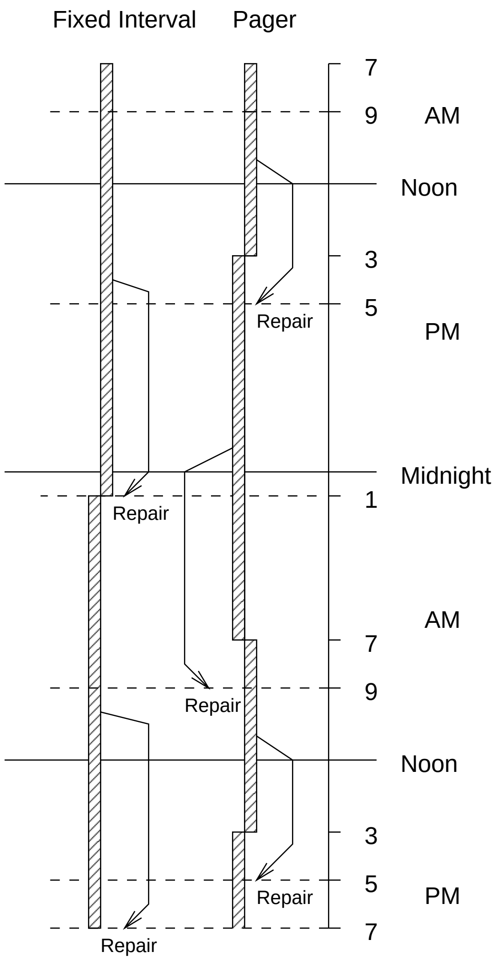


Figure 6.20: Fixed Interval and Pager Arrangements

easier for the operator to handle data layout issues, especially filesystem full conditions, when there are fewer and larger “disks”. It is also easier to tune the performance of the system, as striping will inherently balance the load evenly among the disks in the stripe set. On the other hand, reliability will suffer due to a disk failure making the whole dataset vulnerable.

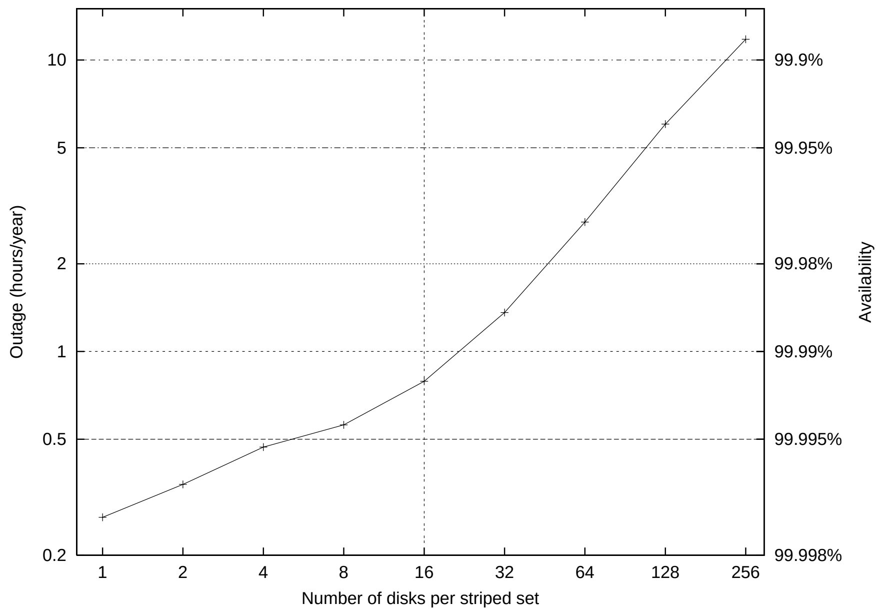


Figure 6.21: Stripe set sizes
(Dotted line indicates default)

Figure 6.21 shows how the number of disks on a striped set affects availability. Our default is 16, meaning the 32 disks on a double-ended pair are combined into two striped sets, one per PC. With our current software, the actual maximum is 32, since we do not have a network filesystem or some other network striping software. Therefore, the figures for size 64 and larger are purely hypothetical. On the other end of the graph, a size 1 stripe set is actually a single disk.

As can be seen from the figure, as the striped set size increases, the availability gradually goes down. However, even at the maximum, when the entire collection of disks are divided into only two stripe sets (for mirroring), the outage per year is only a little over 10 hours. With no striping, the outage can be reduced to about one third of our default configuration, but with a significant penalty in operator labor.

6.3.6 Mirroring

“How valuable is mirroring?”

Figure 6.22 shows how mirroring will affect availability. I changed the SCSI disk failure rates and plotted the outage per year for no mirroring, 2-way mirroring, 3-way mirroring and 4-way mirroring systems. If a particular plot ends in the middle of the graph, it means the outage has been less than 0.01 hours per years to the right of the last point.

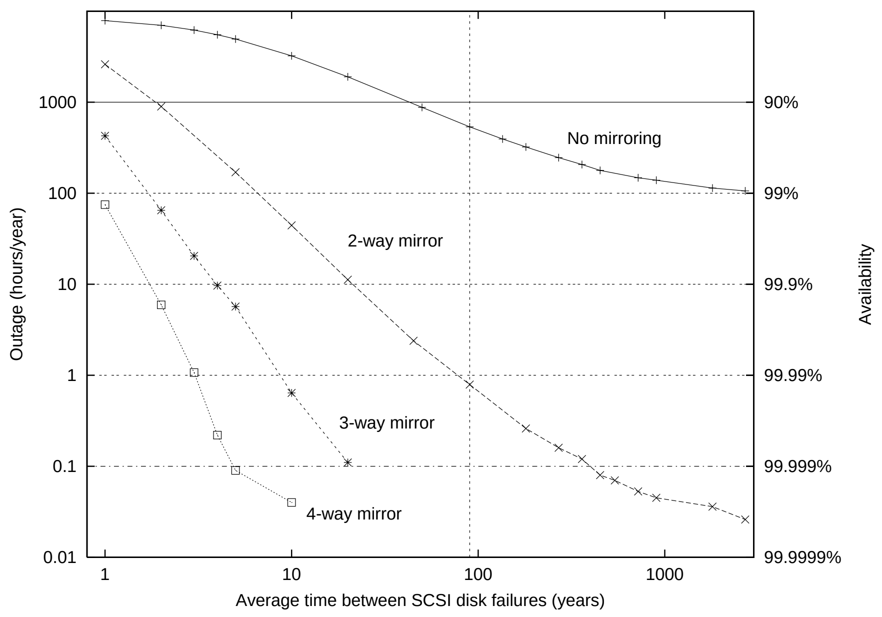


Figure 6.22: Disk failure rates and mirroring
(Dotted line indicates observed value)

Without mirroring of data, the MTBF goes down to 1,300 hours (0.16 years) and availability to 93.9% (540 hours of downtime per year) with an average downtime of 85 hours. The dominating factor is disk failures, as any disk going down will cause a failure.

With extremely reliable disks, the outage can be reduced to about 100 hours per year, but this is still unacceptably high. With less reliable disks, the numbers are positively dismal without mirroring. On the other hand, with 3-way or 4-way mirrors, the reliability goes up significantly.

6.3.7 RAID-5

"How about RAID-5?"

Can we substitute mirroring between machines with a parity-based protection scheme within the same machine and still have the same kind of availability? The answer is no.

According to Figure 6.22, with no mirroring, the availability can only improve to about 99% even with infinitely reliable disks. A parity-based protection scheme within an enclosure or a SCSI adapter will be limited by this number as an upper bound. In other words, a system without inter-machine data protection simply cannot be made reliable enough because of other single points of failure such as SCSI buses.

6.3.8 Double-Ending

"How valuable is double-ending of SCSI disks?"

With no double-ending, i.e., with each enclosure only connected to one PC, the availability goes down to about 99.95%, or 4 hours of outage per year. This level is about five times what the current system offers. The MTBF goes down to about 14 years, or 125,000 hours, which is one quarter of the current system.

Figures 6.23 and 6.24 show how double-ending affects availability. The horizontal axis are average time between SCSI drive failures and IDE drive failures, respectively.

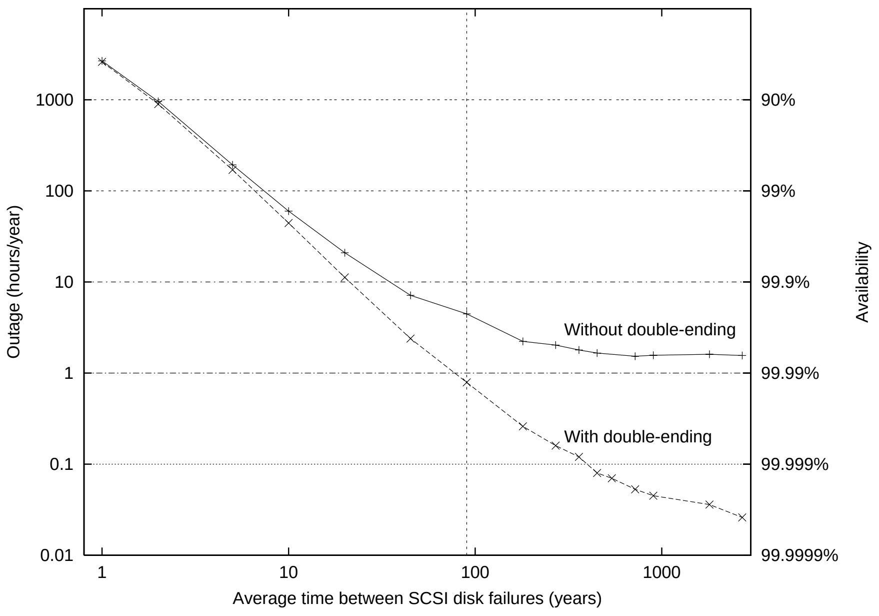


Figure 6.23: SCSI drive failure rates and double-ending
(Dotted line indicates observed value)

We can see from Figure 6.23 that SCSI disk failure rates affect configurations with and without double-ending. When the disk failure rates are very high, the two graphs coincide; when the main reason for outages are data disk failures, adding connectivity to the system by double-ending doesn't improve the overall availability. When the disk failure rates are better, the overall system availability will increase, but without double-ending, the improvement is limited, as the line without double-ending tails off at about 2 hours per year, when other factors start dominating the availability.

Figure 6.24 provides a partial answer to that limitation: the IDE drive failures are almost masked out when the SCSI drive are double-ended, but without double-ending, it greatly affects the system reliability, as a single PC failure causes a few dozen disks to be disconnected temporarily.

Figure 6.25 shows how decreasing the number of disks per striped set will affect availability. Even without any striping, the outage per year can only be reduced to about 3 hours without double-ending. The reason of this limitation is also because SCSI disk failures are no longer the main cause of unreliability in the case without double-ending.

It is not really possible to create a system without double-ending with similar reliability characteristics as our current system given our failure rates by just adjusting number of components. For instance, adding more PCs will reduce the number of disconnected disks when one of the PCs go down, but increases the chance that one of the PCs will be down at any given moment. The only ways to significantly reduce the failures without double-ending are to reduce the repair interval or

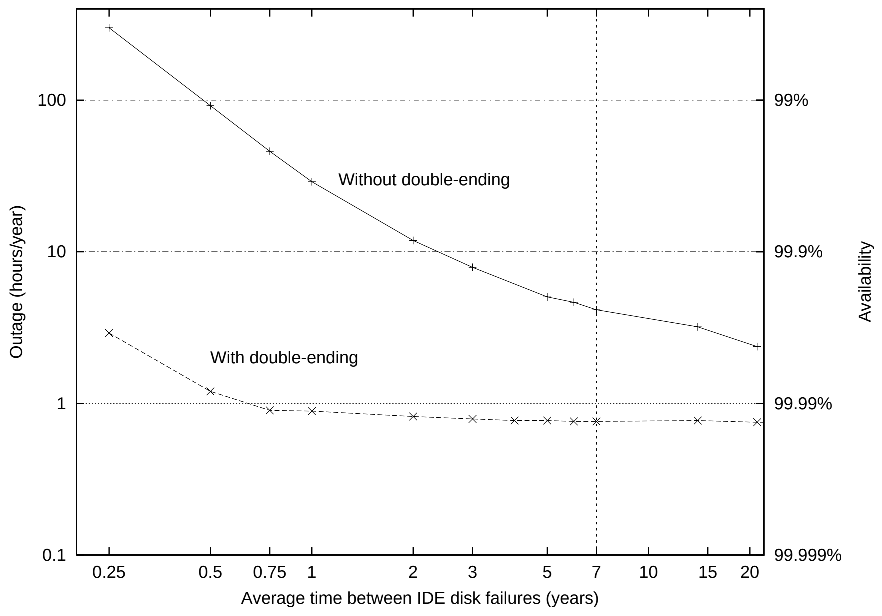


Figure 6.24: IDE drive failure rates and double-ending
(Dotted line indicates observed value)

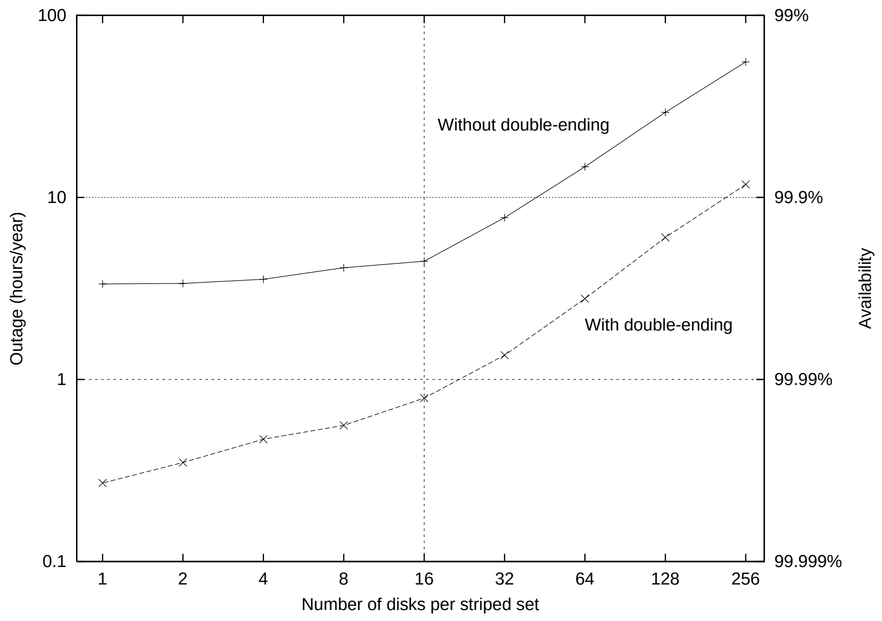


Figure 6.25: Size of striped sets and double-ending
(Dotted line indicates default)

increase the number of mirrors of data.

Figure 6.26 shows the effect of repair interval on failures on systems with or without double-ending. Repair interval of 3 days will reduce the outage hours per year to about 0.8 on a system without double-ending, which is comparable to our double-ending system with a repair interval of 7 days. As the two lines are roughly parallel, for the Tertiary Disk system it appears that double-ending allows you to double the repair interval and yield the same availability.

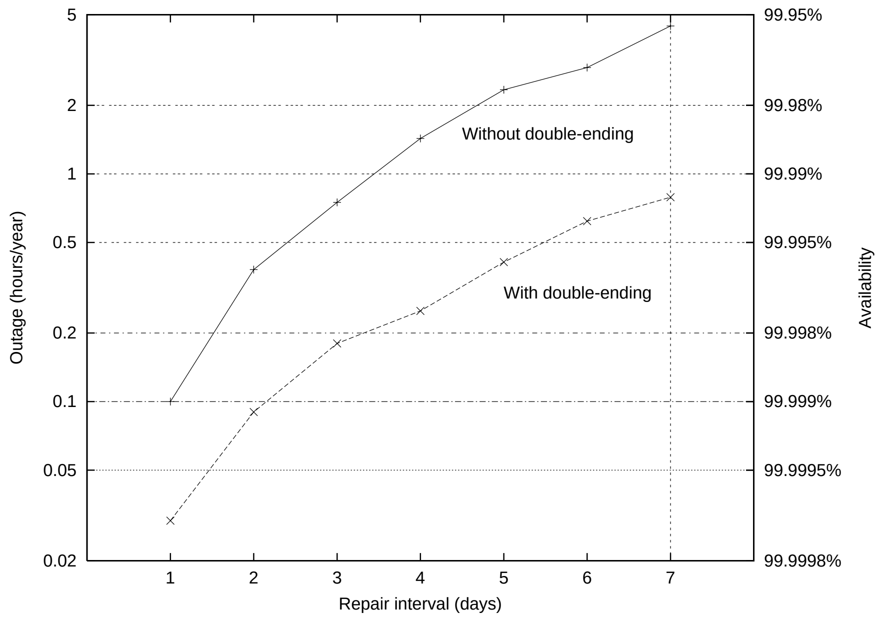


Figure 6.26: Repair interval and double-ending
(Dotted line indicates default)

Figure 6.27 shows how increasing the number of mirrors affect the downtime. With three copies of data, a system without double-ending can be made more available than a double-ended system with the normal two-way mirror.

6.3.9 Operating System Crashes

"Will operating system crashes affect availability? How much will reboot time after a crash affect availability?"

Since we haven't experienced any machine crashes caused by operating system bugs during our project, the generator doesn't generate OS-related reboot events by default. However, as mentioned before, it can be made to generate those events by command-line switches.

Figures 6.28 and 6.29 show the effect of operating system crashes with the original reboot time and fast reboot improvements. As mentioned in Section 3.3.2, the original reboot time was 22 minutes; with the fast reboot modifications, it was reduced to 1 minute.

As can be seen from Figure 6.28, even with very frequent crashes, there was virtually no effect on outage hours per year. The reason why operating system crashes have very little effect on

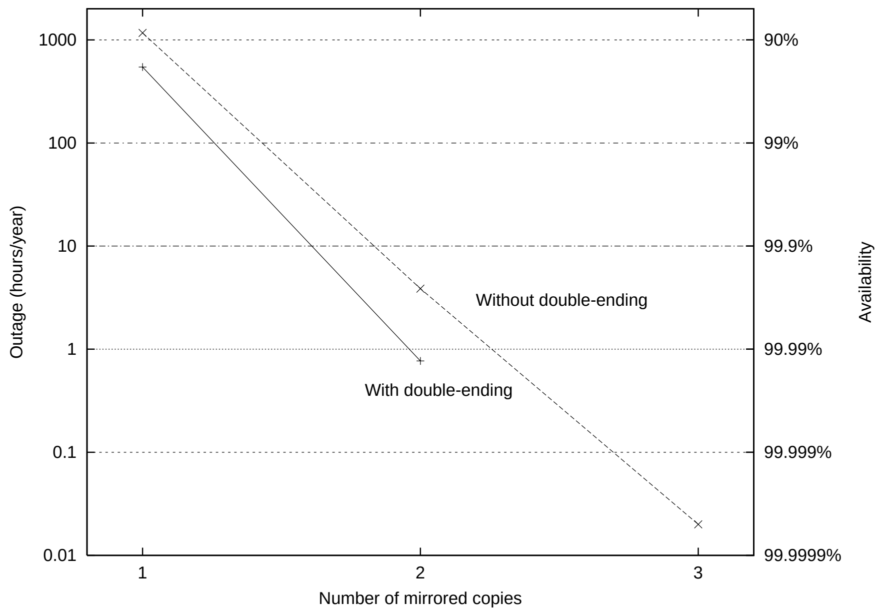


Figure 6.27: Mirroring and double-ending

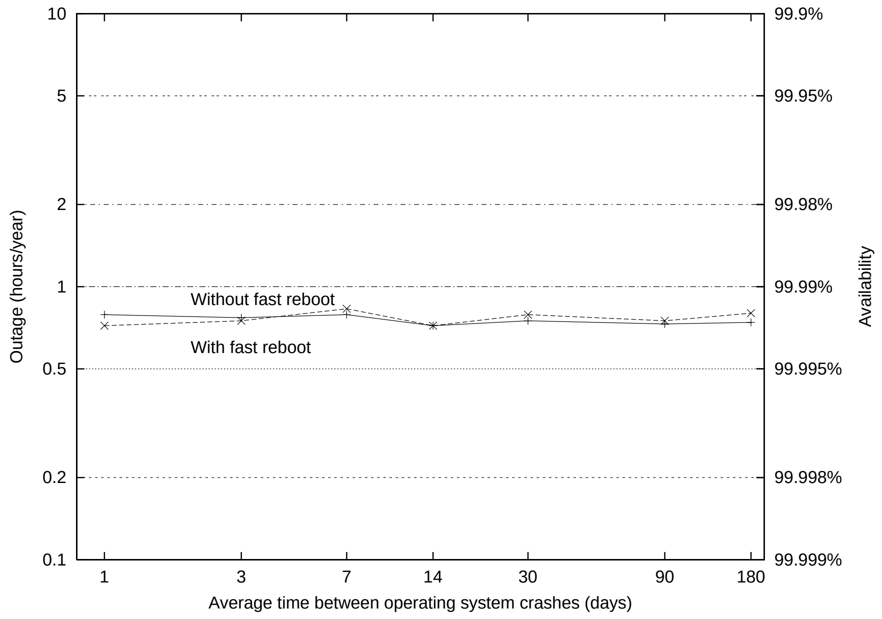


Figure 6.28: Operating system crashes and availability

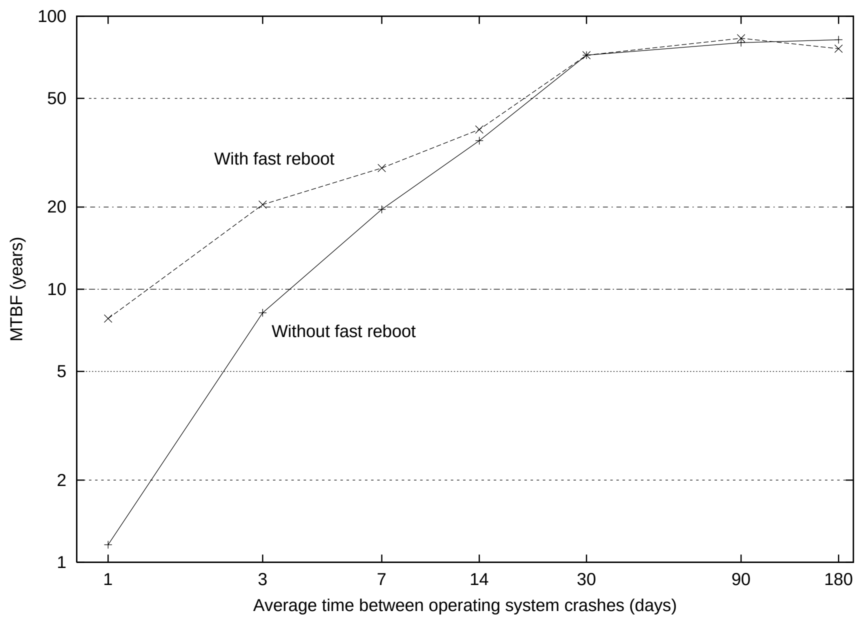


Figure 6.29: Operating system crashes and MTBF

outage hours is because the recovery time from such crashes is very small compared to hardware failure repairs; thus, even if the system suffers an outage, it will recover very quickly, resulting in little extra outage time. The difference between systems with and without fast reboot are within the error range.

On the other hand, Figure 6.29 shows that the MTBF of the system will be affected with frequent crashes. With operating systems that crash daily, the MTBF will be down to about 1 year with normal reboots, and about 8 years with the fast reboot improvements.

6.3.10 Number of PCs per Disk

"What is the effect of decreasing or increasing the number of disks per PC?"

Intuition seems to suggest that, if you add more PCs to the system, the availability will improve since there will be less things to be disconnected when a PC goes down. However, this is actually not the case: with the increase in numbers of PCs comes more PC failures.

Figure 6.30 shows how little adding more PCs to the system helps availability. Furthermore, even with only two PCs, connected to all the disks on the system via double-ending, the system availability suffers only a very small hit.

6.3.11 Total Size

"How will a system 10 times as big as the current one behave? How can I make it as reliable as the current one?"

If we were to build a system that has more components, what will its characteristics be?

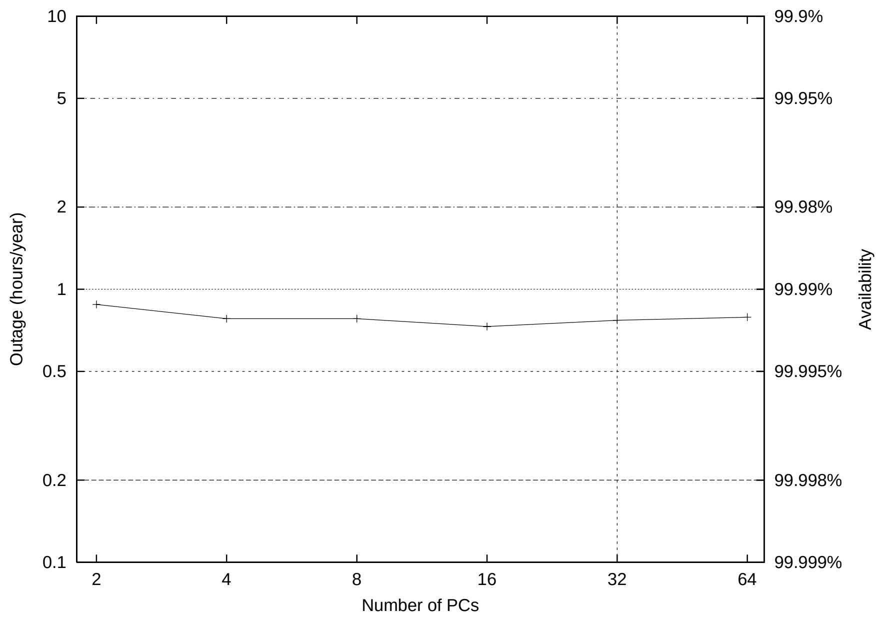


Figure 6.30: Changing number of PCs

For instance, if we were to fill up the entire machine room with ten times as many disks and PCs, how will it behave?

Figure 6.31 shows how availability is affected by the total system size. The simulation indicates that degradation in availability is proportional to size. For instance, the outage average is 7.67 hours per year, about ten times that of the current system, for a tenfold system.

What will it take to bring such a system under control? From previous sections, we know that increasing the number of mirrors, decreasing the stripe set size, and reducing the repair interval are the only options.

I found that, with 3-way mirroring, the system's outage will be down to less than 0.1 hours per year. Note that replicating data in more places will reduce the effective capacity of a set of disks, so the cost penalty is rather severe. For instance, a 3-way mirror will cost 50% more than a 2-way mirror to construct given the same net storage system size.

Figure 6.32 shows the effect of reducing the repair interval on a system ten times the size of our current system. To bring the outage down to a comparable level, the repair interval has to be 2 days—the operator comes in three times as often.

However, with an arrangement similar to the one proposed in Section 6.3.4, we can reduce the operator cost without compromising reliability. For instance, with one-day intervals on a tenfold system, 38% of the days have one or more failure. If the operator carries a pager and only comes in when the system had a failure in the previous 24 hours, this person only needs to actually make a visit 2.7 days per week.

Figure 6.33 shows how reducing the number of disks per striped set will help. By not using any striping, the outage can be reduced to about twice that of the current system. However, the burden of maintaining 5,120 one-disk filesystems would be great.

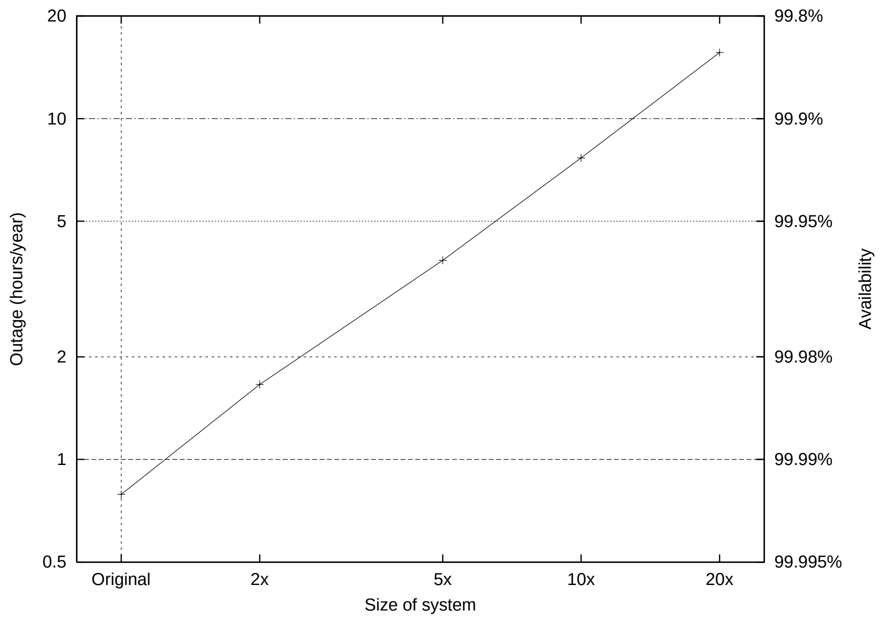


Figure 6.31: Increasing the total system size

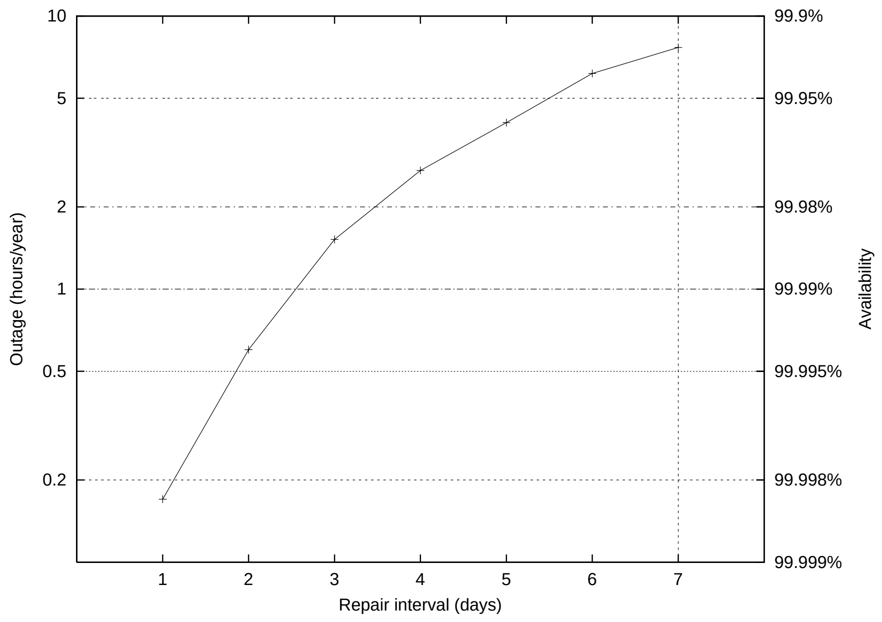


Figure 6.32: Reducing the repair interval
(Dotted line indicates default)

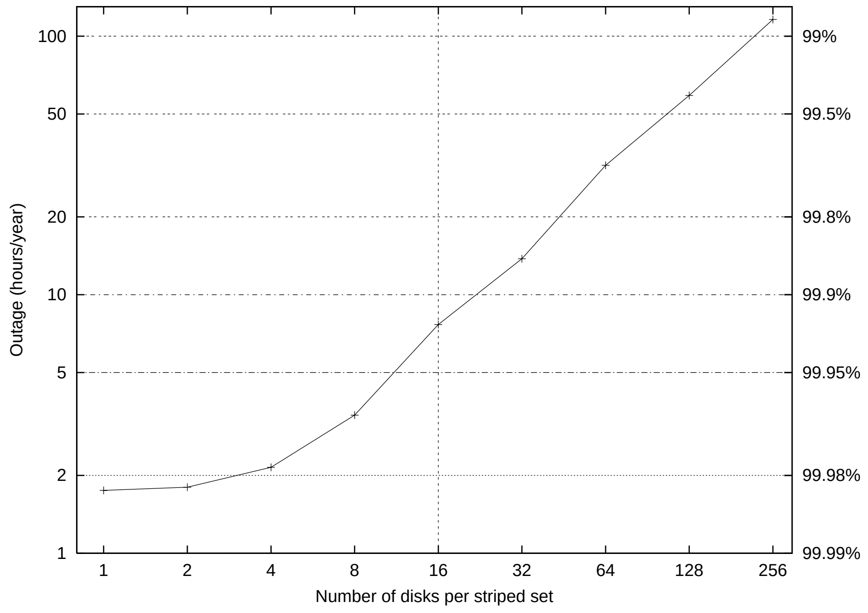


Figure 6.33: Reducing the number of disks per striped set
(Dotted line indicates default)

6.3.12 99.999% or 99.9999% Availability

"What will it take to build a system with 99.999% availability given the current component failure rates? What about 99.9999%"

Some commercial systems boast "five nines", i.e., 99.999% availability. What will it take to build such a system using only commodity components? What about "six nines", i.e., 99.9999% availability?

99.999% availability only allows 0.09 hours, or 5 minutes, of downtime per year, and 99.9999% allows 0.009 hours, or 30 seconds. We need to reduce our downtime by a factor of 10 or 100 to obtain this. From previous discussions, it is clear that the only options available are to increase the number of mirrored copies of data or decreasing the repair interval.

With current failure rates and configurations, a 3-way mirror will accomplish the task quite easily. In fact, with a 3-way mirror, my simulator detected only 66 hours of outage in 100,000 years, or 2.4 seconds of downtime per year, which translates to 99.999992% availability. A 3-way mirror incurs a 50% capacity penalty.

The other option, reducing the repair interval, can get us to 99.999% with a two-day repair interval. As for 99.9999%, we can obtain that goal with an 18-hour repair interval as mentioned in Section 6.3.4.

6.4 Summary

I have implemented a event-driven simulator which constructs a DAG to represent the system and simulates its behavior efficiently. Using this simulator, and a random event generator based on the data I collected in Chapter 5, I evaluated the availability of our system and explored the design space.

These are the key discoveries presented in this chapter.

- Our system appears to work very well under the observed failure rates. To put things in perspective, we had three campus-wide power failures or network outages in the past 12 months, resulting in more than twenty hours of disconnection from the outside world. Compared to these, the 1 hour per year outage rate of our system is miniscule.
- Of all component reliabilities, the SCSI disk failure rate has the greatest effect on system reliability, with system reliability being a near linear function of SCSI disk reliability. This result is positive, as we want the system availability to be a function of the storage itself, and not some less visible component.
- SCSI disk reliability dominates because double-ending effectively masks failures of everything else by providing extra connectivity. Without double-ending, the system's reliability suffers and the system is much more sensitive to component failure rates.
- Without mirroring data, the system will be very unreliable. On the other hand, adding the third mirrored copy of all data will make it virtually unbreakable.
- Adding more PCs, and thus more connectivity, to the system, does not affect the overall system reliability much. The reason is because while more PCs will cause less disks to be affected if one of them is disconnected, the extra PCs will increase the chance that one or more of them is down at any given moment.
- It is possible to construct a system ten times the size of ours using the same equipment and with similar reliability measures by increasing the number of mirrors of data, reducing the numbers of disks per striped set, or reducing the repair interval.
- It is possible to increase the system availability to 99.999% by increasing the number of mirrors of data or reducing the repair interval. To improve it to 99.9999%, the only option is to increase the number of mirrors of data or further reducing the repair interval. However, as mentioned in Section 6.3.4, it could be possible to reduce the repair interval without increasing the operator cost too much.

Chapter 7

Conclusion

7.1 Results

In this dissertation, I presented the following:

- The design of a large storage system built using commodity components only
- An application and modifications to the operating system we made to make it self-maintaining and run almost unattended
- An analysis of various methods to reduce system administration cost of large storage systems
- A catalog of components failures we observed over two years the prototype has been in operation
- An evaluation of the reliability of our design by simulation and exploration of the design space

7.2 Contributions

In summary, these are the contributions of this research:

- Demonstration of construction of self-maintaining Internet service—both via experiment on our constructed system and via simulation—with much lower system administration cost, and much lower hardware costs
- Showing mirroring level, disk reliability, and repair time are the three key parameters in constructing highly available large scale storage servers with enough internal connectivity
- Observations on how to architect a system so that it can be highly available yet be largely insensitive to reliability of hidden infrastructure components
- Use of DAG to represent system to simplify simulation of availability
- Recording failures and repairs of a large system over two years to provide parameters to an availability simulator

7.3 What We Would Have Done Differently

There were several design choices we made when we drafted the initial design of our system. Some of them were forced by circumstances, as other alternatives were not readily available at that time, while some were our conscious decision. This section will summarize those and explore other possibilities to evaluate if we would build a system differently if we were to design it again today.

7.3.1 Disk Interface

At the beginning of the project, we had only one choice for the disk interface, single-ended SCSI. Newer and better standards, such as differential SCSI, SSA and FC-AL were released but components supporting those standards had not yet reached "commodity" level.

In three years, the landscape has changed significantly. SSA merged into FC-AL, but the joined standard has not yet reached commodity status. IDE, which was once the low-end of disk drives, has made leaps and bounds in terms of reliability and especially performance. However, IDE still has a two-disk-per-cable restriction. Unless we are going to a design with significantly fewer disk drives per PC, IDE is not yet a viable solution.

As for SCSI, LVD ("low-voltage differential") disks have become widespread in the last couple of years. LVD combines the benefits of differential SCSI and a lower signal voltage to achieve an even higher bus speed of up to 80MHz, which can transfer up to 160MB/s on a 16-bit cable, with a bus that is much more electrically stable than the original single-ended SCSI bus running at 20MHz.

Given that we had a fair number of problems with SCSI enclosure and cable integrity, LVD-SCSI equipment, the current standard on the market, would be a better choice today.

7.3.2 Disk Purchases

During the course of the project, our department suffered two massive failures on departmental fileservers despite them having RAID-5 protection. One of them was caused by disk firmware bugs. Apparently the particular revision of firmware on the disks had a problem that will manifest itself while doing a RAID-5 reconstruction. Thus, when one of the disk failed and the automatic recovery process started, the entire array was lost.

Although we have not had such catastrophic failures, we had our share of disk firmware problems as mentioned in Section 5.3.3. I have also personally experienced disk problems related to firmware on SCSI disks I own—I had to upgrade some of them to make the disks work reliably.

The biggest issue with disk firmware problems is that they often will not show up immediately—sometimes they will not appear until the most inopportune moment, such as RAID-5 reconstruction on our departmental fileserver. Unfortunately, those are often also times of emergency, with the system's vulnerability is already high.

In order to safeguard against such problems, with a system like ours, it might be a good idea to use disks of different brands. For instance, if we purchase disks from two manufacturers, use the same disks to construct a striped set and make sure the two mirrored copies of data reside on disks from different manufacturers, we will be able to survive a catastrophic failure caused by disk firmware problems.

7.3.3 Disk Enclosures

By far the hardest part of setting up the system initially was dealing with disk enclosures. There were numerous problems with them, including SCSI backplane integrity problems, missing cables, wrong type of mounting rails, and so on.

After sending almost half of them back to the manufacturer for repair, the enclosures have been working reasonably well, with only three additional failures in two years. However, my experience shows that moving the enclosures will sometimes cause them to stop working, and we have been unable to move the Tertiary Disk prototype to another location because of that reason.

However, I am not sure if we could have done anything differently with regards to disk enclosures. We have tested several of the enclosures before ordering them in full—in fact, the ones we initially tested are still working fine to this day on `stampede`, our file server, without any SCSI bus integrity related problems. The manufacturer apparently changed something with their design between those and the ones we got in large quantity, and that caused the problem. As disk enclosures are relatively low-volume products, these change of designs are fairly frequent and there is little a purchaser can do about it, except checking references of the manufacturer.

In another note, as mentioned in Section 1.3.3, many newer disk enclosures have environmental monitoring systems called SCSI Environmental Services (SES) which can be accessed through the SCSI bus. To build a self-maintaining storage system, SES will be very useful.

7.3.4 PC Cases

We used ordinary computer cases for our PCs. In fact, these are “ordinary” PCs in the truest form of the word—they were part of a large donation to the department and most of the others went into graduate students’ offices. There are two problems with them.

The first problem is that they are not very space-efficient. For our system, we did not need many PCs so it wasn’t a big problem, but if the PC to disk ratio is higher, the space inefficiency will be a serious issue.

The second problem is that they are not very easy to handle, as these PCs are designed to be used in ordinary offices, and not to be installed in machine-room racks. The operator’s job was made much harder when there was a repair that involves changing a component inside one of the PCs.

Both problems can be solved by using rack-mount PC cases. Although these cases are more expensive than ordinary PC cases—rack-mount PC cases cost about $250 while ordinary cases cost $50 to $100—the reduction in space and operator burden should more than make up for the cost increase.

7.4 Future Work

There are some issues that have not been fully resolved in our system. This section will list two of those and suggest future directions for potentially interesting research.

7.4.1 Completion

Some of the improvements listed in this dissertation, such as bypassing the BIOS during booting and a PC automatically taking over for its double-ended companion, are not fully implemented. Some of these should be completed before the system can be considered truly self-maintaining.

7.4.2 Design Parameters

The experimentation with different design parameters are truly hypothetical at this point. For instance, we have argued that one part-time operator can maintain a 5,000-disk system without much difficulty. However, that analysis was based on extrapolation only, assuming the difficulty of tasks will increase proportionally with the system size. Such assumptions are not always true, as there are other factors, such as psychological effects the system size has on the operator, the likelihood of the operator getting confused about which part of the system needs repair, that can not be measured without extensive research in those areas.

Bibliography

[ACPtNT95] Thomas E. Anderson, David Culler, David Patterson, and the NOW Team. A case for Networks of Workstations. *IEEE Micro*, pages 54–64, February 1995.

[ADN$^+$95] Tom Anderson, Michael Dahlin, Jeanna Neefe, David Patterson, Drew Roselli, and Randy Wang. Serverless network file systems. In *Proceedings of the 15th Symposium on Operating Systems Principles*, 1995.

[AH85] Robert G. Angus and Ronald J. Heslegrave. Effects of sleep loss on sustained cognitive performance during a command and control simulation. *Behavior Research Methods, Instruments & Computers*, 17(1):55–67, 1985.

[And97] Eric Anderson. *System Administration: Monitoring, Diagnosing, and Repairing*. Ph.D. Qualifying exam Proposal, available from `http://www.cs.berkeley.edu/~eanders/`, April 1997.

[AP97] Eric Anderson and Dave Patterson. Extensible, scalable monitoring for clusters of computers. In *Proceedings of the 11th Systems Administration Conference (LISA '97)*, pages 9–16, October 1997.

[Asa99] Satoshi Asami. *GridPix: the Interactive Tile-based Image Viewer*. `http://now.cs.berkeley.edu/Td/GridPix/`, 1999.

[ATP99] Satoshi Asami, Nisha Talagala, and David Patterson. Designing a self-maintaining storage system. In *Proceedings of the Sixteenth IEEE Symposium on Mass Storage Systems*, pages 222–233, March 1999.

[Bak93] Mary Baker. Fast crash recovery in distributed file systems (Ph.D. thesis). Technical Report UCB/CSD-94-787, UC Berkeley, 1993.

[Bar81] Joel F. Bartlett. A NonStop kernel. In *Proceedings of the 1st Symposium on Operating Systems Principles*, 1981.

[BBC$^+$90] Joel Bartlett, Wendy Bartlett, Richard Carr, Dave Garcia, Jim Gray, Robert Horst, Robert Jardine, Dan Lenoski, and Dix McGuire. Fault tolerance in Tandem computer systems. Technical Report 90.5, Part Number 40666, Tandem Computers, Inc., 1990.

[BBD$^+$96] William J. Bolosky, Joseph S. Barrera, III, Richard P. Draves, Robert P. Fitzgerald, Garth A. Gibson, Michael B. Jones, Steven P. Levi, Nathan P. Myhrvold, and

Richard F. Rashid. The Tiger video fileserver. In *Proceedings of the 6th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 96)*, April 1996.

[BFD97] William J. Bolosky, Robert P. Fitzgerald, and John R. Douceur. Distributed schedule management in the Tiger video fileserver. In *Proceedings of the 16th Symposium on Operating Systems Principles*, pages 212–223, 1997.

[BGM$^+$96] Elizabeth Borowsky, Richard Golding, Arif Merchant, Elizabeth Shriver, Mirjana Spasojevic, and John Wilkes. Eliminating storage headaches through self-management. In *Proceedings of the 1996 OSDI*, October 1996.

[BHK$^+$91] Mary Baker, John Hartman, Mike Kupfer, Ken Shirriff, and John Ousterhout. Measurements of a distributed file system. In *Proceedings of the 13th Symposium on Operating System Principles*, pages 198–212, October 1991.

[Bia98] Andrzej Bialecki. *PicoBSD: fitting FreeBSD on a single floppy*. `http://www.freebsd.org/~picobsd/`, 1998.

[BL98] Eric Brewer and Ken Lutz. *Private communication*. Inktomi Corp., 1998.

[BSD94] Disklabel - read and write disk pack label. In *4.4 BSD System Manager's Manual*. O'Reilly & Associates, 1994.

[CGP$^+$88] Peter Ming-Chien Chen, Garth A. Gibson, David A. Patterson, Randy H. Katz, and Martin E. Schulze. Two papers on RAIDs. Technical report, University of California at Berkeley, 1988.

[CNC$^+$96] Peter M. Chen, Wee Teck Ng, Subhachandra Chandra, Christopher Aycock, Gurushankar Rajamani, and David Lowell. The Rio file cache: Surviving operating system crashes. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 74–83, October 1996.

[DK93] Ann L. Drapeau and Randy H. Katz. Striped tape arrays. Technical report, UC Berkeley, 1993.

[dSG93] James da Silva and Ólafur Guǒmundsson. The amanda network backup manager. In *Proceedings of LISA*, pages 171–182, November 1993.

[FAM96] *The Art ImageBase*. Fine Arts Museums of San Francisco, `http://www.thinker.org/imagebase/`, October 1996.

[FB93] The FreeBSD Project, `http://www.FreeBSD.org/`, 1993.

[FG99] Stephen H. Fairclough and Robert Graham. Impairment of driving performance caused by sleep deprivation or alcohol: A comparative study. *Human Factors*, 41(1):118–128, March 1999.

[Fin97] Raphael A. Finkel. *Pulsar: An extensible tool for monitoring large Unix sites*. IEEE, 1997.

[Gib92] Garth Gibson. *Redundant Disk Arrays: Reliable Parallel Secondary Storage*. The MIT Press, 1992.

[GM99] Justin Gibbs and Kenneth Merry. *Private communication*. The FreeBSD Project, 1996–1999.

[GNA$^+$96] Garth Gibson, David F. Nagle, Khalil Amiri, Fay W. Chang, Eugene Feinberg, Howard Gobioff, Chen Lee, Berend Ozceri, Erik Riedel, and David Rochberg. A case for network-attached secure disks. Technical Report CMU-CS-96-142, Carnegie Mellon University, 1996.

[GNA$^+$98] Garth Gibson, David F. Nagle, Khalil Amiri, Jeff Butler, Fay W. Chang, Howard Gobioff, Charles Hardin, Erik Riedel, David Rochberg, and Jim 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)*, 1998.

[Gra85] Jim Gray. Why do computers stop and what can be done about it? Technical report, Tandem Computers, Inc., 1985.

[Gra90] Jim Gray. A census of tandem system availability between 1985 and 1990. *IEEE Transactions on Reliability*, 39(4):409–418, October 1990.

[GS91] Jim Gray and Daniel P. Siewiorek. High-availability computer systems. *IEEE Computer*, 24(9), September 1991.

[HA93] Stephen E. Hansen and E. Todd Atkins. Automated system monitoring and notification with swatch. In *Proceedings of LISA*, pages 145–155, November 1993.

[HD89] Hui-I Hsiao and David J. DeWitt. Chained declustering: A new availability strategy for multiprocessor database machines. Technical report, University of Wisconsin, June 1989.

[Her97] Gary Herbst. IBM's drive temperature indicator processor (Drive-TIP) helps ensure high drive reliability. Technical report, IBM Corp., `http://www.storage.ibm.com/hardsoft/diskdrdl/technolo/drivetemp/drivete%mp.pdf`, October 1997. white paper.

[HP96a] John L. Hennesy and David A. Patterson. *Computer Architecture: A Quantitative Approach*, pages 18–28. Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.

[HP96b] John L. Hennesy and David A. Patterson. *Computer Architecture: A Quantitative Approach*, pages 29–34. Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.

[HWS98] G. Robert J. Hockey, David G. Wastell, and Jürgen Sauer. Effects of sleep deprivation and user interface on complex performance: A multilevel analysis of compensatory control. *Human Factors*, 40(2):233–253, June 1998.

[IBM98] *Self-Monitoring Analysis and Reporting Technology (S.M.A.R.T.)*. IBM Corp., `http://www.pc.ibm.com/us/infobrf/ibsmart.html`, 1998.

[Jir99] Jirobase management platform specification. Technical report, Sun Microsystems, October 1999. white paper, available from `http://www.jiro.com/`.

[KMHPP97] Eastman Kodak, Microsoft, Hewlett-Packard, and Live Pictures. *FlashPix Standard*. July 1997.

[KMR] T. Kwan, R. McGrath, and D. Reed. User access patters to NCSA's world wide web server. Technical report.

[LC97] David E. Lowell and Peter M. Chen. Free transactions with Rio Vista. In *Proceedings of the 17th Symposium on Operating System Principles*, October 1997.

[LT96] Edward K. Lee and Chandramohan 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, 1996.

[McK84] Marshall Kirk McKusick. A fast file system for UNIX. *ACM Transactions on Computer Systems*, pages 181–197, August 1984.

[McK85] Marshall Kirk McKusick. *Fsck - The UNIX File System Check Program*. 4.2BSD, Computer Systems Research Group, UC Berkeley, 1985.

[McK98] Marshall Kirk McKusick. *Private communication*. 1998.

[MFM95] Dan Mosedale, William Foss, and Rob McCool. Administering very high volume internet services. In *Proceedings of LISA IX*, pages 95–102, September 1995.

[MKF83] Daniel J. Mullaney, Daniel F. Kripke, and Paul A. Fleck. Sleep loss and nap effects on sustained continuous performance. *Psychophysiology*, 20(6):643–651, 1983.

[ML98] *Live Linux*. Media Lab., Inc., `http://www.mlb.co.jp/live/`, June 1998.

[NA92] Network Appliance, Inc., `http://www.netapp.com/`, 1992.

[OD89] John Ousterhout and Fred Douglis. Beating the I/O bottleneck: A case for log-structured file systems. *Operating Systems Review*, 23(1):11–27, January 1989.

[Ous94] John Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[Pau89] Paul B. Paulus, editor. *Psychology of Group Influence*, pages 15–51. Lawrence Erlbaum Associates, Inc., 2nd edition, 1989.

[RD98] Yansong (Jennifer) Ren and Joanne Bechta Dugan. Design of reliable systems using static & dynamic fault trees. *IEEE Transactions on Reliability*, 47(3):234–244, September 1998.

[SBMS85] Margo Seltzer, Keith Bostic, Marshall M. McKusick, and Carl Staelin. An implementation of a log-structured file system for unix. In *Proceedings of the 1993 Winter USENIX*, pages 119–130, June 1985.

[Sch95] Jürgen Schönwälder. *Scotty—Tcl Extensions for Network Management Applications*. `http://wwwhome.cs.utwente.nl/~schoenw/scotty/`, 1995.

[Sha95] Tom Shanley. *PCI System Architecture*. Addison-Wesley, Reading, Mass., 3rd edition, 1995.

[SMH95] Michael Shaddock, Michael Mitchell, and Helen Harrison. How to upgrade 1500 workstations on saturday, and still have time to mow the yard on sunday. In *Proceedings of the Ninth USENIX Conference on System Administration*, pages 59–65, 1995.

[SR93] *Network Buyer's Guide*. Strategic Research Corp., `http://www.networkbuyersguide.com/`, 1993.

[TAAP98] Nisha Talagala, Satoshi Asami, Tom Anderson, and David Patterson. Large scale distributed storage. Technical Report UCB/CSD-98-989, UC Berkeley, January 1998.

[Tal99] Nisha Talagala. Parameterizing disk drives (Ph.D. thesis). Technical report, UC Berkeley, 1999.

[Tan97a] NonStop availability for NonStop Himalaya and Windows NT server systems. Technical report, Tandem Computers, 1997. white paper.

[Tan97b] Tandem maintenance and diagnostic system (TMDS). Technical report, Tandem Computers, 1997. white paper.

[TAP+98] Nisha Talagala, Satoshi Asami, David Patterson, Bob Futernick, and Dakin Hart. The Berkeley-San Francisco Fine Arts Database. In *Proceedings of the Fifteenth IEEE Symposium on Mass Storage Systems*, pages 163–167, March 1998.

[TML97] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A scalable distributed file system. In *ACM Symposium on Operating Systems Principles*, pages 224–237, 1997.

[Wal98] Jim Waldo. Jini architecture overview. Technical report, Sun Microsystems, 1998. white paper.

[WID97] *DHCP WIDE-Implementation*. WIDE Project, `http://www.wide.ad.jp/software/README.dhcp.txt`, April 1997.

[Wil98] John Wilkes. *Private communication*. 1998.

[X3T] X3T9.2 Committee. *SCSI-2 Standard*.

[Zaj66] Robert B. Zajonc. *Social Psychology: An Experimental Approach*, pages 10–15. Wadsworth Publishing Company, Inc., 1966.