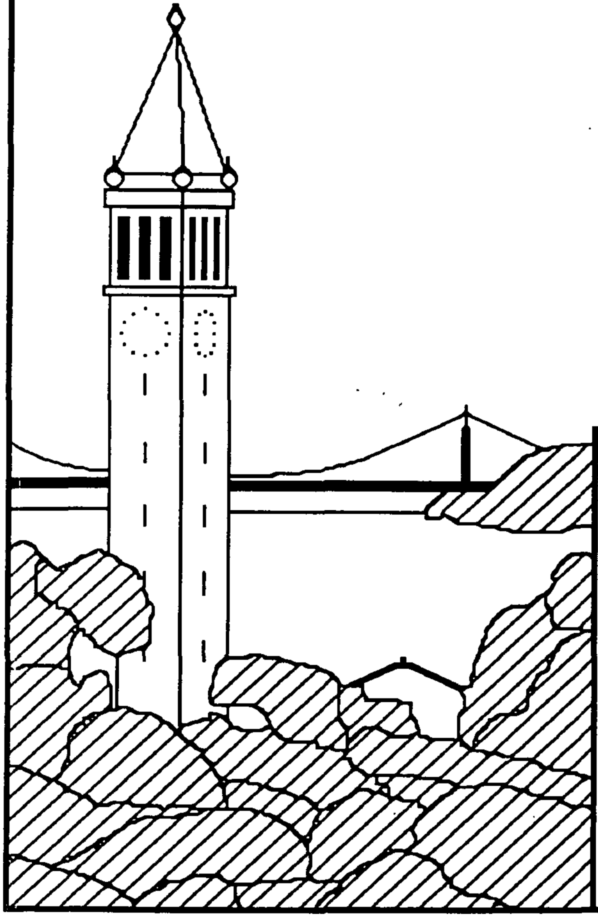
NAG 2-591
IN-60-CR
73849
P-267

Redundant Disk Arrays: Reliable, Parallel Secondary Storage

Garth Alan Gibson



Report No. UCB/CSD 91/613

December 1990

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

Redundant Disk Arrays:
Reliable, Parallel Secondary Storage

By

Garth Alan Gibson

B.Math.(Hons.) (University of Waterloo) 1983
M.S. (University of California) 1987

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

NAG-2591

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA at BERKELEY

Approved:

Chair: ....D. A. Patterson.................april 29, 1991
Date

....April 25, 1991.

........R. W. Wolff.............................4/24/91........

****

Redundant Disk Arrays:

Reliable, Parallel Secondary Storage

by

Garth Alan Gibson

Abstract

During the past decade, advances in processor and memory technology have given rise to increases in computational performance that far outstrip increases in the performance of secondary storage technology. Coupled with emerging small-disk technology, disk arrays provide the cost, volume, and capacity of current disk subsystems but, by leveraging parallelism, many times their performance. Unfortunately, arrays of small disks may have much higher failure rates than the single large disks they replace. Redundant Arrays of Inexpensive Disks (RAID) use simple redundancy schemes to provide high data reliability. This dissertation investigates the data encoding, performance, and reliability of redundant disk arrays.

Organizing redundant data into a disk array is treated as a coding problem in this dissertation. Among alternatives examined, codes as simple as parity are shown to effectively correct single, self-identifying disk failures.

D. A. Patterson

David A. Patterson

The performance advantages of striping data across multiple disks are reviewed in this dissertation. For large transfers this parallelism reduces response time. Striping data also automatically distributes independent, small accesses across disks to increase throughput. This dissertation evaluates the performance lost to the maintenance of redundant data. This loss is negligible for large transfers but can be significant for small writes because of increases in aggregate disk service time.

Because disk arrays include redundancy to protect against the high failure rates caused by large numbers of disk components, it is crucial that disk failures be characterized. This dissertation provides evidence that disk lifetimes can be modeled as exponential random variables.

Building on an exponential model for disk lifetimes, this dissertation presents analytic models for disk-array lifetime, evaluates these against event-driven simulation, and applies them to an example redundant disk array. These models incorporate the effects of independent and dependent disk failures (shared support hardware) as well as the effects of on-line spare disks. For the example redundant disk array, these models show that a 10% overhead for an N+1-parity encoding plus a 10% overhead for on-line spares can provide higher reliability than the 100% overhead of conventional mirrored disks.

Redundant Disk Arrays:

Reliable, Parallel Secondary Storage

by

Garth Alan Gibson

Dedicated to my grandparents

Ida Maud (Babcock) Stocks

You are my Mnemosyne, my goddess of memory.
Your reminiscences have given me a family mythology:
''candies down the pea belt,''
''Billy Bounce,'' and ''United Empire Loyalists.''
So much of my identity is based on your retrospections.

Herbert Samuel Albert Stocks

Although I never knew you,
I am forever influenced by your presence
in the warm waters of my favourite place,
the Madawaska.

Ruth Eliza (Warren) Gibson

You left me pearls in these words
from your poem, Time:
''There is nothing so precious as time
It is neither yours nor mine
To use it well, we have to learn
For yesterday's time will never return.''

Henry Walker Gibson

Always a smile, always a tease,
always a kind and wise word,
it was always so comfortable to be with you,
even if you did always win at horseshoes.

Acknowledgements

The time I have spent at Berkeley has been richly rewarding primarily because of the efforts of David Patterson. He has given me guidance, enthusiasm, and wisdom as well as countless hours of time. The challenges that he laid out for me first attracted me to Berkeley, his patient council brought me through the darkest days of the development of SPUR, and his keen intuition provoked my research into I/O systems. I have benefited immeasurably from his tutelage.

All three members of my committee, David Patterson, John Ousterhout, and Ronald Wolff, have contributed generously to the research reported in this dissertation. Although Dave propelled me through this research, all might have been in vain without John's insights into systems and Ron's patient coaching on modeling. Of course, all three deserve a special note of thanks for wading through drafts of this dissertation.

This dissertation also benefited substantially from the efforts of two others, Tom Philippi and Barbara Beach. Their careful proofreading, extensive editorial suggestions, and gentle humour made my task much more bearable.

Contributing to the contents of Chapter 3 in this dissertation have been many gracious colleagues. The basic ideas for arrays of small-diameter disks grew out of work I did with David Patterson and Randy Katz. The extensions to binary, double-erasure correcting codes is from collaborative work with Richard Karp, Lisa Hellerstein, David Patterson, and Randy Katz. David Patterson, Randy Katz, and I also contributed to the RAID measurement experiment carried out by Peter Chen with the assistance of Amdahl Corporation. Peter Chen and Ed Lee also contributed insight into their work on data striping-unit selection and parity placement strategies, respectively.

Thanks go to Thinking Machines Corporation for making available to me the disk failure data reported in Chapter 4. Rick Epstein, Sue Maxwell, and Raymond Mak were instrumental in making this possible. Thanks also go to Sue Maxwell and Joseph Yarmus for reading and improving a draft of this chapter.

The contents of Chapter 5 grew out of research from Martin Schulze's Masters report and his subsequent work with David Patterson, Randy Katz, and myself. In developing the models in this chapter, I relied heavily on the experience and insight of Ronald Wolff.

On a broader note, there have been many people who contributed to my understanding of the material in this dissertation. Early in my research I was fortunate to encounter the vision of Michael Stonebraker, Dieter Gawlick (then with Amdahl, now with DEC), and Jim Gray (then with Tandem, now with DEC). As you will see in Chapter 3, Jim has made a significant and provocative contribution to my understanding of disk array performance. I have also greatly benefited from the expert advice of Mike Mitoma, Jim Brady, and Jai Menon of IBM Almaden Research Center, Dave Gordon of Array Technology, Dave Anderson of Imprimis (now Seagate), and David Tweten of NASA/NAS at Ames.

Here at Berkeley, my research developed in parallel with the RAID project. Many of its members, notably Randy Katz, David Patterson, Ken Lutz, Peter Chen, Ed Lee, Ann Chervenak, Rich Drewes, Ethan Miller, and Martin Schulze, contributed to my understanding of disk arrays. Their efforts in the construction of a prototype, called RAID-I, were important to our collective education.

Together with the RAID group, Berkeley's POSTGRES and Sprite research groups formed the XPRS research environment which combined database, operating systems, and I/O systems research into a wide learning experience for me. Michael Stonebraker, Margo Seltzer, and Mark Sullivan, members of POSTGRES, had a great influence on my thinking.

I have been involved and enriched by the Sprite project since it began. Sprite, as an experimental operating system, has provided me with an powerful simulation engine though its transparent process migration and its large number of high-performance workstations. Sprite is the result of the efforts of John Ousterhout, Brent Welch, Mike Nelson, Fred Douglas, Andrew Cherenson, Mendel Rosenblum, Mary Baker, John Hartman, Ken Shirriff, Mike Kupfer, Bob Bruce, and Adam de Boor. Fred and Adam, in particular, are responsible for the massive amount of simulation Sprite transparently provided for me.

Although this research does not include any material directly pertaining to my earlier involvement with the SPUR multiprocessor workstation project, this project and its members were instrumental to my approach to systems research. Thanks to David Patterson, Randy Katz, David Hodges, George Taylor, Mark Hill, David Wood, Susan Eggers, Deog-Kyoon Jeong, Shing Kong, Corinna Lee, Joan Pendleton, Scott Ritchie, Walter Beach, Daebum Lee, Douglas Johnson, and Ken Lutz.

I think that an important, and often overlooked source of education in graduate school is the interaction between junior and senior students. For me, Mark Hill, Tony DeRose, George Taylor, Susan Eggers, David Wood, Gregg Foster, and Prabhakar Ragde were notable peer mentors. As I have tried to pass on this fine tradition to Peter Chen, Ed Lee, and Ann Chervenak, I have learned anew how educating collaboration with bright new minds can be.

Special thanks go to those who have helped me enjoy life during the preparation of this dissertation. Henry Moreton and the Mystery! gang gave me a weekly excuse for diversion, and Ania and Tina Upstill and their parents Steve and Krys were my family away from home. Although distant, my family at home are the root of the strength and conviction that gets me through trying times. Their love for me and confidence in me shield me from all dangers.

My final thanks go to Barbara Beach. Barbara was the one who, many years ago, suggested that I pursue a PhD degree. It was she, more than most, who suffered as I followed through on this suggestion. And it is she, more than any other, that I seek to please and impress with each goal I attain.

My next goal, especially intended to begin to repay all that I owe her, is to be as supportive of the dissertation that she is now writing as she has been of this one.

Table of Contents

CHAPTER 1: Introduction 1

1.1: Overview of the Dissertation 3

1.2: A Personal Note 5

CHAPTER 2: The Importance of Input/Output 9

2.1: Revisiting an Old Problem 10

2.1.1: Memory Hierarchies 10

2.1.2: The Access Gap Problem 12

2.1.3: Overlapping I/O with Computation 14

2.1.4: Lowering I/O Response Time 15

2.1.5: Lowering Memory Miss Ratios 20

2.2: Rapidly Growing Processor Performance 22

2.3: New Opportunities in Secondary Storage Technologies 24

2.4: Replacing Magnetic Disks with DRAM: How Soon? 26

2.5: Summary 29

CHAPTER 3: Redundant Disk Arrays 31

3.1: The Emergence of Disk Arrays 32

3.2: Disk Arrays As a Threat To Data Reliability 39

3.2.1: Avoiding Catastrophic Failures with Prediction 41

3.2.2: Protecting Data Reliability with Off-Line Redundancy 43

3.3: On-Line Redundancy: Encoding Data into Disk Arrays 44

3.3.1: Single-Erasure-Correcting Encodings 47

3.3.1.1: Mirroring 47

3.3.1.2: N+1 Parity 49

3.3.1.2.1: Byte-Interleaved Striping 50

3.3.1.2.2: Block-Interleaved Striping 51

3.3.1.2.3: Non-Striped 53

3.3.2: Multiple-Erasure-Correcting Encodings 54

3.3.2.1: Binary Symbol Codes 56

3.3.2.1.1: Examples: Hamming and Two-Dimensional Parity Codes 56

3.3.2.1.2: Optimal, Double-Erasure-Correcting Binary Codes 58

3.3.2.2: Non-Binary Symbol Codes 60

3.3.3: Encoding Summary 64

3.4: Performance of Redundant Disk Arrays 66

3.4.1: Benefits of Data Striping 67

3.4.1.1: Striping for Parallel Transfer 68

3.4.1.2: Striping for Disk Load Balancing 69

3.4.1.3: Extra Positioning Operations 70

3.4.1.3.1: Synchronizing Disk Spindles 72

3.4.1.3.2: Shortening Positioning Distances 73

3.4.1.4: Interaction Between Striping Unit and Workload 73

3.4.1.5: Selecting a Best Striping Unit 76

3.4.2: Performance Lost to Maintaining Redundancy 78

3.4.2.1: Fundamental Differences For Single-Erasure Correction 79

3.4.2.2: Experimental Validation 80

3.4.2.3: Optimizations for Small Write Accesses 82

3.4.2.3.1: Caching and Hints 82

3.4.2.3.2: Floating Parity 83

3.4.2.3.3: Log-Structured File System 84

3.4.2.4: Double-Erasure-Correcting Codes 85

3.5: Summary 86

CHAPTER 4: Characterizing Disk Lifetimes 88

4.1: Characterizing Catastrophic Disk Failures 89

4.2: In Search of Public Data on Lifetime Distributions 93

4.3: Improving Disk Mean Lifetimes 96

4.4: A Sample of Lifetime Data 98

4.4.1: Empirical Reliability 102

4.4.2: Exponential Model 104

4.4.3: Weibull Model 106

4.4.4: Sensitivity 110

4.5: Summary 112

CHAPTER 5: Reliability Modeling 115

5.1: Reliability Metric 118

5.2: Related Work 121

5.3: Tools and Methods 124

5.4: Independent Disk Failures 126

5.4.1: Markov Model for a Single Parity Group 126

5.4.1.1: Approximations and Simplifications 128

5.4.2: Multiple Parity Groups 130

5.4.3: Calibrating Consistency with Simulation 131

5.4.4: Implications for the Design of Disk Arrays 133

5.5: Dependent Disk Failures 136

5.5.1: Markov Model for Orthogonal Disk Array 139

5.5.2: Estimating Mean Array Lifetime 142

5.5.3: Separating Disk Repair into Replacement Delivery and Recovery 148

5.5.4: Implications for the Design of Disk Arrays 150

5.6: Independent Disk Failures with On-line Spares 152

5.6.1: Estimating Mean Array Lifetime 152

5.6.2: Implications for the Design of Disk Arrays 158

5.7: Dependent Disk Failures with On-line Spares 161

5.7.1: Modeling Mean Array Lifetime 164

5.7.2: Infinite Spares Bound 167

5.7.3: One String of On-Line Spare Disks 169

5.7.4: Two Strings of On-Line Spare Disks 174

5.7.5: Implications for the Design of Disk Arrays 176

5.7.5.1: Higher Overhead for Redundancy May Not Improve Reliability 177

5.7.5.2: Higher Reliability Through Faster Disk Recovery 179

5.7.5.3: Higher Array Reliability Through Higher String Reliability 181

5.7.5.4: Partially Populated Spare Strings and Low Reorder Thresholds 182

5.8: Summary and Conclusions 186

5.8.1: Summary of Reliability Models for Redundant Disk Arrays 186

5.8.2: Design of A Strawman Redundant Disk Array 190

5.8.3: Conclusions 192

CHAPTER 6: Conclusions 194

Bibliography 200

APPENDIX A: Reliability Simulation 0

A.1: Introducing RELI 0

A.2: Parity Group Organization in a Set of Strings 219

A.3: Modeling Repair, Recovery, and Spare Pool Replenishing 220

A.4: Verifying RELI 221

A.5: Terminating RELI 222

APPENDIX B: Effect on Reliability of Error in MTTDL 223

APPENDIX C: Goodness-of-Fit Testing 223

APPENDIX D: Comparison Data for Chapter 5 227

Table of Figures

1.1: "RAID the First" Prototype 8

2.1: Extrapolations for the Ratio of DRAM to Disk Price per Bit 26

3.1: Disk Array Configuration Flexibility 37

3.2: Striped Disks with N+1 Parity 48

3.3: Striped Disks with Block-Interleaving and Distributed Parity 52

3.4: Non-Striped Disks with Distributed Parity 53

3.5: Larger Redundant Disk Arrays Have Lower Reliability 55

3.6: Double-Erasure-Correcting Code Examples 57

3.7: Contrasting Binary and Non-Binary Codes by Example 60

3.8: Disk Array Performance with 1 KB Striping Unit 74

3.9: Disk Array Performance with 32 KB Striping Unit 76

3.10: Fundamental Redundant Disk Array Performance Comparison 79

3.11: N+1-Parity Performance Measurements 81

4.1: The Bathtub Lifetime Distribution 90

4.2: Examples of Exponential, Weibull, and Gamma Lifetime Distributions 91

4.3: Product-Limit Estimates for Reliability 103

4.4: Graphical Estimation of Exponential Reliability 105

4.5: Graphical Estimation of Weibull Reliability 107

5.1: Exponential Reliability versus Ratio of Time to Mean Lifetime 119

5.2: Exponential 1-, 3-, and 10-Year Reliabilities versus Mean Lifetime 120

5.3: Data Loss Model for Independent Disk Failures in a Single Parity Group 126

5.4: Single Parity Group Reliability versus Approximations 129

5.5: Reliability Example for Multiple Parity Groups 130

5.6: Error in Estimated MTTDL Relative to Simulation 132

5.7: Estimated versus Simulated Reliability 132

5.8: Reliability versus Redundancy Overhead 134

5.9: Strawman Reliability versus Repair Time 134

5.10: Example of Support Hardware Shared by Multiple Disks 136

5.11: Orthogonal Organization of Parity and Support Hardware Groups 138

5.12: Markov Model for an Orthogonal Disk Array 138

5.13: Reliability for Strawman with an Orthogonal Organization 140

5.14: Exponential Estimate versus Markov Solution for Reliability 141

5.15: Submodels for Data Loss in Orthogonal Disk Arrays 142

5.16: Orthogonal Estimated MTTDL versus Markov Solution 144

5.17: Simplified Orthogonal MTTDL Estimate versus Markov Solution 145

5.18: Simulated Orthogonal MTTDL versus Markov Solution 147

5.19: Simulated 1-, 3-, and 10-year Reliability 147

5.20: Partitioning Disk Repair into Replacement and Recovery 149

5.21: Repair Durations in Orthogonal Strawman 150

5.22: Example of Spare-Pool Depletion and Data Loss 153

5.23: Simulatation versus Estimatation with Spares 157

5.24: Evaluating Benefits of a Small Pool of Spare Disks 158

5.25: Pool Size, Reorder Threshold, versus Disk Delivery Time 159

5.26: Two Failed and Spared Disks Example 161

5.27: String Failed and Spared Example 162

5.28: Simulated versus Exponential 1-, 3-, and 10-year Reliabilities 163

5.29: Effect of Spares on Mean Lifetime 165

5.30: Generic Model for MTTDL 166

5.31: Submodels for Orthogonal Disk Arrays with Infinite Spares 167

5.32: Submodel for Orthogonal Array with One String of Spares 169

5.33: Detailed Submodel for Orthogonal Array with One String of Spares 171

5.34: MTTDL with One String of Spares 173

5.35: Submodel for Orthogonal Array with Two Strings of Spares 174

5.36: MTTDL with Two Strings of Spares 175

5.37: N+1 Parity versus Mirrored Reliability 178

5.38: Disk Recovery Time versus On-line Spares 180

5.39: Diminishing Effect of More Reliable Strings on Array Reliability 182

5.40: Partially Populated Spare Strings and Low Reorder Thresholds 183

B.1: Effect of 10% Error in Estimated MTTDL on Reliability 223

Table of Tables

1.1: Comparison of a Strawman Redundant Disk Array to an IBM 3390 2

3.1: 1987 Magnetic Disk Technology Comparison 33

3.2: 1990 Magnetic Disk Technology Comparison 34

3.3: 1990 Magnetic Disk Cost and Price 35

3.4: Check-Disk Overhead Limits for a Non-Binary Hamming Code 62

4.1: Lifetime Data Sample One 99

4.2: Lifetime Data Sample Two 100

4.3: Exponential Lifetime Distribution Maximum Likelihood Estimates 105

4.4: Weibull Lifetime Distribution Maximum Likelihood Estimates 109

4.5: Parameter Estimates for Alternative Data Extensions 111

5.1: Comparison of Strawman Disk Array to IBM 3390 117

5.2: Summary of Reliability Estimates for Strawman Disk Array 190

C.1: Goodness-of-Fit Tests on Each Set of Simulated Lifetimes 225

C.2: Goodness-of-Fit Tests on All Simulated Lifetimes in Each Section 227

D.1: Raw Data for Figures 5.6 and 5.6 in Section 5.4 231

D.2: Raw Data for Figures 5.18 and 5.19 in Section 5.5 233

D.3: Raw Data for Figure 5.23 in Section 5.6 235

D.4: Raw Data for Figures 5.34 and 5.36 in Section 5.7 243

D.5: Raw Data for Figure 5.40 in Section 5.7 244

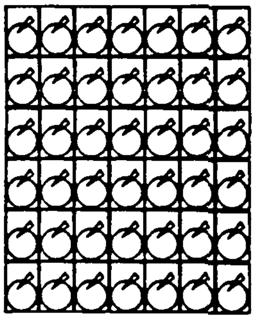
Table of Equations

2.1 12
2.2 13

3.1 59
3.2 63
3.3 70
3.4 77
3.5 77
3.6 78

4.1 90
4.2 91
4.3 92
4.4 92
4.5 103
4.6 104
4.7 104
4.8 106
4.9 107
4.10 108
4.11 108
4.12 108
4.13 108

5.1 118
5.2 118
5.3 119
5.4 121
5.5 127
5.6 127
5.7 127
5.8 129
5.9 130
5.10 131
5.11 131
5.12 131
5.13 137
5.14 143
5.15 143
5.16 143
5.17 144
5.18 145
5.19 150
5.20 153
5.21 154
5.22 154
5.23 155
5.24 155
5.25 155
5.26 156
5.27 156
5.28 156
5.29 164
5.30 168
5.31 170
5.32 170
5.33 172
5.34 174
5.35 175
5.36 184



CHAPTER 1

Introduction

> *"[Memory is] the treasury and guardian of all things."*
> – Cicero

In the electronic world of digital information, Cicero's insight is quite literally true; the memory of a computer system is responsible for containing and protecting all information. Among the many things this responsibility implies is that computer memory must be readily accessible and solidly dependable. This dissertation is concerned with the accessibility and dependability of the outermost recesses of on-line memory: secondary storage. This is the bulwark upon which memory's performance and reliability ultimately depends. Although today's secondary storage systems are the results of years of extraordinary research efforts, society's accelerating dependence on increasely high-performance and cost-effective computers precludes a single optimal design. It is my thesis that the burgeoning demand for reliable, high-performance secondary storage can and will be met by redundant disk arrays.

A *disk array* is a collection of physically small magnetic disks that is packaged as a single unit but operates in parallel. Disk arrays capitalize on the availability of small-diameter disks from a price-competitive market to provide the cost, volume, and capacity of current disk systems but many times their performance. Unfortunately, relative to current disk systems, the larger number of components in disk arrays leads to higher rates of failure. To tolerate failures,

Metric	IBM 3390	Redundant Disk Array
Disk Units	1	70+7+7
Formatted User Data Capacity (MB)	22,700	22,400
Number of Useful Actuators	12	77
Avg. Access Time (msec)	19.7	19.8
Max. Read I/Os/sec/Box	609	3,889
Max. Write I/Os/sec/Box	609	≥ 972
Max. Transfer Rate (MB/sec)	15	130
Disk Power Consumption (W)	2,900	1,000
Volume for Disks (cubic feet)	97	11
Mean Time To Data Loss (1,000 hours)	50-250	6,600
Component Disk Costs ($1,000)	?	67
Customer Price ($1,000)	156-260	?

Table 1.1: Comparison of a Strawman Redundant Disk Array to an IBM 3390. *A "strawman" redundant disk array constructed with 84 IBM 0661 model 370 (3½-inch) disks has many advantages over IBM's top-end disk product, the IBM 3390. This disk array configuration is developed in Chapter 5 as a running example. It has the user capacity of 70 disks; its overhead is 7 disks (10%) for redundant data and 7 disks (10%) for on-line spares. Because parity data is distributed among 77 of the disks and because user data is not stored on spare disks, only 77 disks contribute to its performance. For the maximum I/O accesses per second calculation, the transfer unit is a single sector. For the maximum transfer rate calculation, the transfer unit is a track from every disk that contains user data (77 disks). Most metrics apply to disk components only and may be degraded when controller and host effects are included. The IBM 3390 mean time to failure is not publicly known but can be expected to be better than IBM's previous top-end product, which is reported to have had a mean time to failure of 53,000 hours [Balanson88]. To compare costs, I show the price a disk array manufacturer would pay for comparable 3½-inch disks from Seagate and the price range that IBM's best customers pay for a maximally configured IBM 3390 and half of an IBM 3990 (disk controller). This table is constructed from the data and results in Tables 3.2, 3.3, 5.1, and 5.2 and Figure 3.10.*

redundant disk arrays devote a fraction of their capacity to an encoding of their information. This redundant information enables the contents of a failed disk to be recovered from the contents of non-failed disks. This dissertation will highlight the simplest and least expensive encoding for this redundancy, known as *N+1 parity*. In addition to compensating for the higher failure rates of disk arrays, redundancy allows highly reliable secondary storage systems to be built much more cost-effectively than is now achieved in conventional duplicated disks.

Disk arrays that combine redundancy with the parallelism of many small-diameter disks are often called Redundant Arrays of Inexpensive Disks (RAID) [Patterson88]. This combination promises improvements to both the performance and the reliability of secondary storage.

For example, Table 1.1 compares IBM's premier disk product, the IBM 3390, to a redundant disk array constructed of 84 IBM 0661 3½-inch disks. The redundant disk array has comparable or superior values for each of the metrics given in Table 1.1 and appears likely to cost less. I present this table here to demonstrate the potential advantages of redundant disk arrays; I will explain performance and reliability in detail in the rest of this dissertation.

To support my thesis, I make three principal contributions in this dissertation.

(1) I demonstrate that secondary storage systems must inevitably rely on redundant arrays of small-diameter disks.

(2) I provide a broad understanding of alternative, redundant data encodings and of the relationship between parallelism, redundancy, and performance in disk arrays.

(3) I make possible the three-way optimization of cost, performance, and reliability by developing analytic estimates for the reliability of redundant disk arrays.

1.1. Overview of the Dissertation

Collectively, the four principal chapters in this dissertation demonstrate the performance and reliability advantages of redundant disk arrays. Individually, they explain my motivation for research into Input/Output architectures, introduce and explore coding and performance in redundant disk arrays, present a self-contained analysis of disk-failure data, and model the reliability of a redundant disk array with an N+1-parity encoding.

In Chapter 2 I review secondary storage and place emphasis on techniques and technologies important to the performance of computer systems. Beginning with secondary storage's role in the memory hierarchy and its access gap problem, I enumerate conventional approaches to improving performance. Secondary storage's problems are growing, as I next show, because processor performance advances at a much faster rate than disk performance. Fortunately, disk technology is advancing in ways that can be exploited, as the following section shows. Finally, I look to the time when secondary storage will be entirely solid-state, and magnetic disks will be obsolete. Because this time appears to be quite distant, however, disk arrays will be needed for the foreseeable future.

In Chapter 3 I explain redundant disk arrays, their encodings, and their performance. I begin with a look at the basic arguments for disk arrays, then turn to the threat to reliability posed by large numbers of small disks. To cope with this threat, I examine prediction-based and off-line techniques for improving reliability. Finding these insufficient, I focus on on-line redundancy – a coding problem. Of primary interest here are the inexpensive, single-erasure-correcting codes based on parity, but I also examine binary and non-binary double-erasure-correcting codes that may be needed for very large or very highly reliable disk arrays.

Disk array performance is the topic of the rest of Chapter 3. I begin this topic with a discussion of the performance of a non-redundant disk array, because it is in this context that basic performance expectations for all disk arrays can be established. The key to exploiting the inherent parallelism of a non-redundant disk array is data striping, and critical to data striping is the size of the interleaving unit. After reviewing this important issue, I turn to the performance consequences involved in maintaining redundant data. Except for random writes of small amounts of data, these redundancy-maintenance penalties are not large. For small, random writes, however, 50% to 75% of the total disk array bandwidth can be spent on redundancy maintenance. Fortunately, related research in file systems offers the possibility of eliminating small, random writes altogether.

In Chapter 4 I break from disk arrays to examine the disk failures that redundancy is intended to tolerate. After describing common models for the distribution of disk lifetimes, I review the meager sources of public data on disk lifetimes I was able to find. Fortunately, I am next able to present data about a sample of 1350 disks observed over a period of 18 months. I use this data to explore the fit of exponential and Weibull distributions for disk lifetimes. An exponential distribution turns out to be plausible, and mean lifetimes vary between 80,000 hours (nine years) and over 300,000 hours (34 years).

In Chapter 5 I examine the reliability of data in disk arrays with detailed models for the time until data is lost. After characterizing the behavior of the reliability metric, reviewing related work, and discussing the tools and methods I use in that chapter, I present four models for single-erasure-correcting redundant disk arrays. The first of these models treats all failures as independent disk failures, and this simplest of models has a well-known solution. In addition to presenting this solution, I introduce the approximate modeling techniques used in later sections. The next model recognizes that the subsystem hardware supporting an array's disks – organized into strings – suffers failures affecting multiple disks simultaneously. Redundancy groups must be organized carefully to avoid data loss when these failures cause multiple disks to fail dependently. I then construct a model for disk-array lifetime in the presence of these dependent disk failures. The next two sections are devoted to including on-line spare disks into the prior two models. With on-line spares when there are no dependent failure modes, I am able to construct a model describing all possible contingencies for spare-pool size and replacement-disk order policy. With on-line spares and dependent failure modes, I provide a more restrictive model that estimates lifetimes for arrays with zero, one, two, and infinitely many extra strings containing on-line spare disks and with a replacement policy that orders new disks immediately after every failure. Throughout this chapter, I illustrate these models by applying them to a "strawman" disk array. The result of this ongoing analysis of a particular redundant disk array is shown above in Table 1.1.

1.2. A Personal Note

Before studying disk arrays I participated in the design and implementation of a shared-memory multiprocessor workstation [Hill86, Taylor86, Gibson87, Hill87, Jeong90, Wood90a]. This research project, carried out at the University of California at Berkeley, was called Symbolic Processing Using RISC (SPUR), and it involved the development of three VLSI chips, a processor board containing about 300 MSI chips, and a memory bus protocol. I was responsible for the design and implementation of the memory bus [Gibson89a] as part of the cache controller VLSI chip design [Wood87]. Happily, our five-processor prototype correctly implemented a snooping bus protocol [Eggers89] while co-existing with memory boards and ethernet ports that did not.

My experience with SPUR taught me many things, not the least of which was the great increase in complexity induced when requirements call for a complete, functional system rather than a paper design or even a marginally functional prototype. However, one thing in particular that SPUR made clear was that a high-performance system based on multiple microprocessors could be constructed inexpensively, but such a system could not be provided inexpensively with comparable increases in network and I/O bandwidths. Fresh from my efforts with the part of a memory system between the processor cache and main memory, I turned to the question of high-performance, cost-effective secondary storage.

My research into secondary storage, in collaboration with David Patterson and Randy Katz, led to our 1987 paper that named, described, and compared different organizations of Redundant Arrays of Inexpensive Disks (RAID) [Patterson87]. This launched a broader research project [Katz89a] whose initial goal was a disk-array prototype that was constructed from off-the-shelf disks, controllers, and a host processor. Imprimis (now Seagate) donated 32 5¼-inch disks and Sun Microsystems donated a Sun 4/280 file server. This prototype, known as "RAID the First" or RAID-I, is now complete [Lee90, Chervenak90, Chervenak91] and operates as a file server in Berkeley's networked environment. Figure 1.1 shows a picture of RAID-I while it was being tested.

While RAID the First was under development, the RAID research project blossomed [Gibson89b, Gibson89c, Katz89b, Ousterhout89, Schulze89, Chen90a, Chen90b, Katz90, Rosenblum90, Seltzer90a, Seltzer90b, Stonebraker90b, Lee91, Miller91], and was incorporated into a multi-dimensional project at Berkeley called XPRS that studied shared-memory multiprocessor databases operating on top of a general purpose, network-based operating system that exploits high-performance I/O systems and networks [Stonebraker88]. These other two component projects are, respectively, POSTGRES [Stonebraker90a], a descendent of INGRES, and Sprite [Ousterhout88] which originated with the SPUR project.

The RAID project is now designing a second prototype. Known as "RAID the Second" or RAID-II, this prototype includes the design and implementation of a custom disk-array controller. RAID-II's controller is connected to an UltraNet 100 MB/sec token-ring network by HIPPI interface boards that were donated by Thinking Machines Corporations. The controller contains 64 to 256 MB of buffer memory donated by IBM and accesses up to 120 3½-inch disks, also donated by IBM, that are controlled by up to four Array Technology RAID+ SCSI host bus adapters. RAID-II is controlled by the Sprite operating system executing on an attached (multi-) processing system. The central idea in RAID-II is to make supercomputer file servers with a higher level of performance by providing a high bandwidth path (220 MB/sec) between disks, network, and file cache.

April 24, 1991.

The research was funded by NSF grant MIP-8715235, NASA/DARPA grant NAG 2-591, a Computer Measurement Group fellowship, and an IBM predoctoral fellowship. Additional funding and/or equipment came from IBM, Imprimis, Digital Equipment Corp., Sun Microsystems, Thinking Machines Inc., Array Technologies, Ultranet Corp., Hewlett-Packard, Intel Scientific, Kodak, NCR, Control Data Corp., and StorageTek.

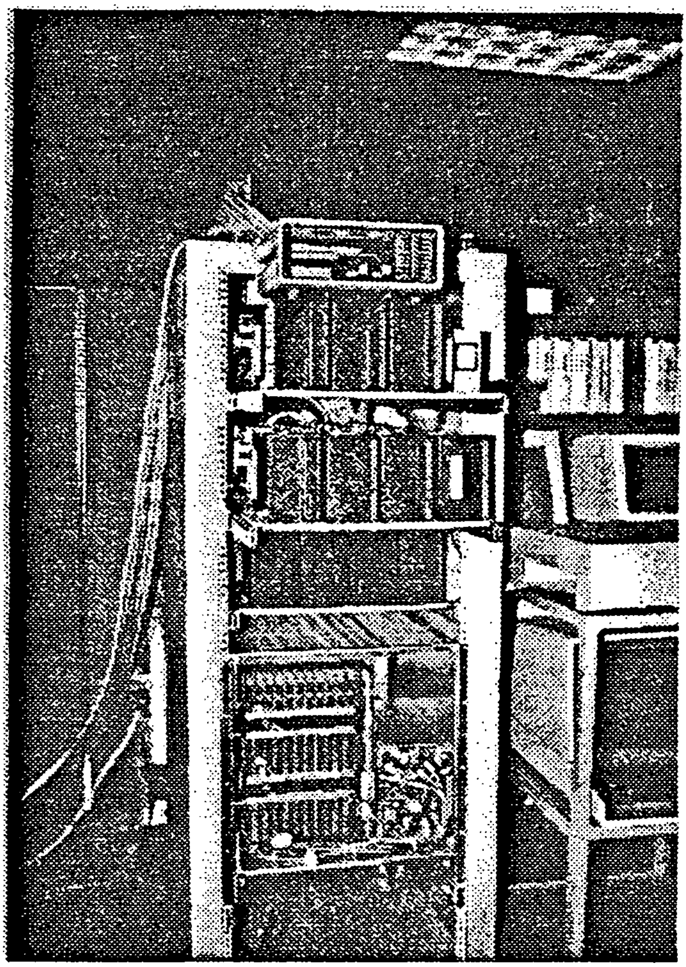


Figure 1.1: "RAID the First" Prototype. *The first prototype constructed by Berkeley's RAID project is composed of a Sun 4/280 host processor donated by Sun Microsystems, 28 5¼-inch, WREN IV disks donated by Imprimis, and four Jaguar disk controllers (employing the SCSI interface protocol). In this picture, the top two (of three) shelves of disks are pulled out from their chassis allowing an ANCOT SCSI analyzer (resting on top of the top shelf) access to their cabling. Each shelf has two sets of four disks, mounted with their platters spinning perpendicular to the plane of the floor. One set faces the front of the chassis and one set faces its back. The last four disks sit on top of the chassis (invisible in the darkness of this image). Notice that the rightmost disk on the top shelf is a 3½-inch, IBM 0661 Lightning. RAID-I's Sprite operating system includes a special "RAID driver" module that implements N+1-parity redundancy. This machine is currently being used at Berkeley as a file server in a cluster of Sprite workstations.*

ORIGINAL PAGE IS
OF POOR QUALITY

CHAPTER 2

The Importance of Input/Output

Why should I study Input/Output (I/O)? The answer to this question is that I/O is an important and neglected portion of computer design. As Patterson and Hennessy write: ''Input/Output has been the orphan of computer architecture'' [Patterson90 pp 499]. With the research presented in this dissertation, I hope to encourage more computer architects to ''adopt'' I/O systems into their portfolio of systems deserving design creativity.

The basic problem is that when a computer designer sets out to build a faster computer, faster is often taken to mean more instructions per second at peak performance. Faster should, however, mean more instructions per second for my tasks, and, if possible, for your tasks too. Where these tasks involve peripheral devices, overall performance will be influenced by I/O performance; designers should be engaged in improving I/O performance in parallel with their efforts to improve processor performance.

I have chosen to study I/O because

(1) the rates at which processor performance and main memory capacity increase far outpaces the rate at which I/O performance increases,

(2) a large class of applications continues to depend on I/O performance, and

(3) technological trends suggest that there is a window of opportunity for architectural innovations for I/O.

I will concentrate on one particular I/O technology in this dissertation, the magnetic disk, because on-line file data is primarily stored on magnetic disks and because file access is the I/O event most likely to be critical to performance.

This chapter begins with a review of the memory hierarchy, the longstanding *access gap* between main memory performance and magnetic disk performance, and common techniques for compensating for this access gap. Then I examine the rapid growth in processing power that aggravates the access gap problem and a similar problem I call the *transfer gap*. Next I describe the evolution toward small-diameter disk technologies that offer new approaches to overcome these access and transfer gaps. Finally, because replacing magnetic disks with solid state memory would be a simple solution to both gaps, I estimate when this might become cost effective.

2.1. Revisiting an Old Problem

The performance problems associated with delays for the mechanical positioning of a magnetic disk's read/write heads are not new. This section reviews the problem and existing approaches to reducing it.

2.1.1. Memory Hierarchies

It has always been extremely expensive to construct a memory system that satisfies all memory references without stalling its processor. This observation was first made by computer pioneers [Burks46], and it led directly to the use of memory hierarchies. *Memory hierarchies* work by satisfying most memory references in fast levels of the hierarchy, whereas slower levels of the hierarchy provide more storage capacity at a much lower cost per bit. It might seem that memory hierarchies should not work; if memory accesses are randomly distributed throughout memory and only a small fraction of memory is supported by fast levels of the hierarchy, then the average access time should be little shorter than the access time of the slower levels. But memory accesses are not randomly distributed throughout memory. They exhibit both *temporal locality* and *spatial locality* [Denning70]. Temporal locality means that recently accessed locations are likely to be rereferenced and spatial locality means that locations near recently accessed locations are likely targets for new references. Locality is exploited by maintaining "recently-used data" in faster levels of the memory hierarchy. This way faster levels of the hierarchy satisfy a much larger fraction of new references than their relative size indicates, and average access time is much closer to the time to access the fastest level.

Traditionally, the memory hierarchy is divided into *primary, secondary,* and *tertiary* levels based on their increasing access times and decreasing cost per bit. The slowest level, tertiary memory, is the least expensive. Usually constructed of magnetic tape, tertiary memory is removable and does not lose information when its power is turned off *(non-volatile).* Secondary memory, composed of mainly fixed-head magnetic disks, is the fastest media that is non-volatile. Because a magnetic disk cannot be removed from its recording mechanism, this *on-line* storage medium is not susceptible to human mistreatment or misplacment. Primary memory, once composed of magnetic cores and now almost universally Dynamic Random Access semiconductor Memory (DRAM), was traditionally the level of storage first accessed by every memory reference made by a program. Today, because processor speeds have grown so quickly relative to DRAM speeds, primary memory has been further divided into on-chip processor caches, first and second level off-chip processor caches, and main, or DRAM, memory.

In this study I will use the term I/O to refer mainly to secondary memory references because these have the largest impact on computer performance. However, an I/O event is really a reference to either non-primary memory or any peripheral device. This concept has been blurred because portions of secondary memory are now used to expand a program's virtual image of primary memory [Denning70] and portions of primary memory are now used to buffer peripheral data and files [Powell77]. Intercomputer network traffic is a form of peripheral I/O that is becoming increasingly important to computer performance, particularly for systems that operate as multiprocessors. Although networking is also the object of substantial research efforts [Arnould89, Computer90], it will not be studied here.

2.1.2. The Access Gap Problem

The *access gap* has been defined as the ratio between the average access time of magnetic disk and the average access time of main memory [Pugh71]. The value of this ratio is frequently between 2,500 and 70,000. The magnitude of this ratio gives only part of the reason that magnetic disk performance is important to computer system performance. For example, if all accesses were satisfied in main memory, the access gap would not matter to overall performance. To best assess the importance of magnetic disk performance, I must examine its contribution to the average time it takes to access memory.

Average memory access time is the best metric for the success of a memory hierarchy architecture [Liptay68]:

$$\textit{Average Memory Access Time} = \sum_{l=1}^{L} RefFrac(l) \times Hit(l) \times HitTime(l) \qquad (2.1)$$

$$\text{where } RefFrac(l) = \prod_{i=1}^{l-1}(1 - Hit(i)) \text{ and } RefFrac(1) = 1 .$$

In this expression, the slowest and cheapest level of the memory hierarchy is level L, and the fastest is level 1. $RefFrac(l)$ is the fraction of processor references that are not satisfied before memory level l, $Hit(l)$ is the fraction of references to level l that are satisfied by level l, and $HitTime(l)$ is the duration of a reference satisfied at level l. If magnetic disk is at level d, then the contribution of magnetic disk performance to average access time is

$RefFrac(d) \times Hit(d) \times HitTime(d)$.

To see how the access gap is related to overall performance, I should express magnetic disk's contribution relative to main memory's contribution (where main memory is at level $m = d-1$):

$$\frac{[\prod_{i=1}^{d-1}(1-Hit(i))] \times Hit(d) \times HitTime(d)}{[\prod_{i=1}^{m-1}(1-Hit(i))] \times Hit(m) \times HitTime(m)} = \frac{(1-Hit(m))}{Hit(m)} \times Hit(d) \times \frac{HitTime(d)}{HitTime(m)} . (2.2)$$

which is the product of the access gap, $HitTime(d)/HitTime(m)$, the success of main memory at satisfying references to it, $(1-Hit(m))/Hit(m)$, and the success of magnetic disk at satisfying references to it, $Hit(d)$.

Because this derivation applies to any pair of levels in the hierarchy, there really are a number of access gaps, $HitTime(l+1)/HitTime(l)$, between successive levels, l and $l+1$. Access gaps at different levels of the memory hierarchy vary widely. For example, between first level caches, which usually employ Static Random Access Memory (SRAM) technology, and main memory, which usually employs DRAM technology, access times increase by a factor of up to 20. As I mentioned above, the access gap between DRAM main memory and magnetic disk is often between 2,500 and 70,000. Finally, in those cases where tertiary storage is used as an on-line archival repository, the access gap between disk and tape is usually 500 to 10,000.

With memory-disk access gaps of four orders of magnitude, the contribution to overall latency from the disk in the memory hierarchy could be quite large unless the fraction of processor references not satisfied by main memory is very small. More specifically, if a disk's contribution is to be comparable to that of main memory, then main memory's hit ratio needs to be between 99.96% and 99.999%. This shows how a large access gap puts substantial pressure on main memory's hit ratio.

2.1.3. Overlapping I/O with Computation

The pressure on main memory's hit ratios lessens when I/O operations overlap with useful computation. When many programs compete for processing resources, a processor idled by an I/O operation can save its current state and resume execution of a different program [Codd60]. This *multiprogramming* approach generally increases the response time of every program, but because processor resources are idle less frequently, it completes a large group of programs in less total time. Multiprogramming's more efficient use of traditionally scarce processor resources was an important reason for its adoption in almost all general purpose computing environments.

Unfortunately, scientific supercomputing or single-user workstation workloads frequently have only one task awaiting execution. In these cases, the only option is to overlap I/O with the same task's computation. Large scientific supercomputer programs explicitly overlap computation with asynchronous I/O requests; that is, applications pre-fetch data and deposit them into privately maintained buffer pools [Cray88]. On-line transaction processing systems have also made extensive use of asynchronous I/O and application buffer pools [Effelsberg84].

There are two ways to automatically overlap a single task's I/O and computation: *read-ahead* and *write-behind* [Feiertag71]. Read-ahead succeeds if the correct data can be prefetched. Fortunately, many applications process file data sequentially, giving read-ahead wide applicability. A simple way to achieve moderate levels of read-ahead is to double or quadruple the size of blocks in a file system [McKusick83]. In contrast, write-behind allows a task to continue without waiting for the completion of each write operation. Since there is no prediction involved, write-behind has even wider applicability. Unfortunately, because it also reduces the likelihood that data is secure on a stable storage media, many applications, notably databases, require mechanisms to bypass write-behind or to force immediate writes [Stonebraker81]. Both read-ahead and write-behind combine profitably with the file or disk caching techniques described in Section 2.1.5.

2.1.4. Lowering I/O Response Time

The most direct way to battle the problem of access gap is to reduce magnetic disk's average access time. In the context of magnetic disks, access time is usually referred to as the time until the disk responds to an access, or its *response time*. Each transfer's response time is the sum of time spent during operating system overhead, disk contention, seek positioning, rotational positioning, and data transfer. In the last three decades, a substantial amount of research has been devoted to reducing most of these delays. This section reviews the most important reductions proposed by these efforts.

When accesses contend for a disk, devastating response time penalties result. As disk utilization approaches 100%, the average number of requests waiting for service and the average waiting time that results grow without bound. This is the reason that some systems with high I/O loads purchase more disks than they need for a given storage capacity [Gray90]. It is also for this reason that systems with high I/O loads periodically redistribute data to balance these loads across all disks [Geist82].

The problem of disk contention is aggravated by I/O architectures with shared disk interconnects and no buffering at each disk. The problem is that the interconnect must be available when the disk reaches the requested data or the transfer must be retried. IBM's I/O architecture, for example, has this problem. Because they overlap one disk's positioning delay with another's transfer on the shared interconnect, IBM disks signal the interconnect managing device, called a storage director, shortly before the disk's read head reaches the requested data [Ahearn72]. This early warning capability is called *Rotational Position Sensing* (RPS). If the storage director does not become free soon after it receives a RPS warning signal, the signaling disk's reconnection will fail, and it must try again after another rotation. As each disk's utilization increases, the probability of contention over shared interconnect rises, and the delays resulting from each retry after an RPS reconnect miss seriously degrade latency [Buzen86, Buzen87]. A rule of thumb used for IBM systems in the 1970s was to distribute files or purchase new disks until shared interconnect was utilized less than 30% of the time and disks were utilized less than 40% of the time [Beretvas78]. Managers of large IBM systems still follow these rules [Peters87].

One way to reduce the unnecessary revolutions induced by a busy interconnect is to provide a small buffer between the disk and the interconnect. As long as the interconnect becomes available before a disk's buffer overflows, that disk is not required to retry its transfer. Because disk-to-host interconnect can easily be made faster than disk transfer rates, buffering at each disk allows greater use of disks [Houtekamer85]. Although non-IBM systems have incorporated buffering at the disk [Massiglia86 pp 242], IBM's policy of backward compatibility makes per-disk buffering difficult in IBM systems [Cormier83, Houtekamer85]. Until IBM can move its application programs to a higher level of I/O abstraction – for example, their current *System Managed Storage* (SMS) interface [Gelb89] – they must rely on disk load balancing and disk caching (discussed in the next section) [Goldstein87] to avoid RPS reconnect-miss delays.

Another way to reduce disk access times is to reorder outstanding disk accesses [Denning67, Toerey72]. This method can be highly advantageous when disk queues are deep, because disk positioning times are a major contributor to disk access times. Deep queues also mean long contention delays, however, and in systems without buffering at the disk, severe delays caused by RPS reconnect misses result. Capacity planners use rules of thumb, such as those given above, specificly to reduce such delays. Because these rules often result in shorter disk queues [Lynch72], request reordering has little benefit. Request reordering is still pursued [Bates89, Geist87, Seltzer90a] because under particular circumstances, such as the write-behind of many files from a large file cache, it can be effective.

One approach to reducing positioning time that works well when queues are shallow relies on multiple copies of the data. By selecting the copy that is "closer," both seek [Bitton88] and rotational [Scheffler73] delays can be reduced. Although most users find the cost of multiple copies prohibitive, systems that already employ duplication for high reliability can easily exploit these benefits.

When transfer units are about the size of a disk track, rotational positioning delays can be nearly eliminated by reading data in the order it is observed by the disk head and then reconstructing it in memory [Salem86]. This approach, called *zero latency reads*, is particularly advantageous because per-disk buffers are usually large enough to store at least one track. For access units that are a fraction, f, of the size of a track, the average rotational latency plus transfer time is reduced by $f^2/2$ revolutions[1] by zero latency reads. This reduces rotational latency plus transfer time by at much as 33% when the access unit is a full track.

Since many user requests involve data that has been dispersed to a variety of locations across the disk, overall latency can be improved by merging the many accesses needed to retrieve this data. The benefit of this merging operation, also called *chaining* [Buzen75], is the result of avoiding the software overhead involved in returning to the user task and issuing the next request. A more powerful method for merging requests is not to split data at all. If disk capacity allocation is the reason data is split, a variety of allocation schemes exist that will encourage contiguity: file system blocks can be enlarged [McKusick83], users can be required to preallocate space [Bohl81], or file systems can employ allocation-clustering algorithms [Koch87, McKusick83, Powell77]. Unfortunately, these mechanisms suffer from wasted capacity because of fragmentation.

Even if requested data is contiguous on disk, allocation schemes can shorten positioning time. For example, by locating more frequently accessed data in the center of a disk's surface it can be favoured with typically shorter seeks [Staelin90]. Similarily, average seek times may be reduced if a disk's heads are returned to the center of the disk surface whenever the disk has no

[1] Without zero latency reads, the average rotational latency plus transfer time is $1/2+f$ revolutions because, with probability $(1-f)$, a transfer rotates an average of $(1-f)/2$ revolutions and, with probability f, it rotates an average of $(1-f/2)$ revolutions, to reach the data, and then rotates f revolutions to transfer it. With zero latency reads, the average rotational latency plus transfer time is $1/2+f-f^2/2$ because, with probability $(1-f)$, a transfer rotates an average of $(1-f)/2$ revolutions and, with probability f, it rotates exactly 1 revolution because it collects the end of the data first.

accesses to perform[2] [King87].

Another technique for reducing I/O response time is to reduce the operating system's overhead. The basic metric for this overhead is the average number of instructions executed by the operating system on behalf of a user's I/O request, known as the *I/O path length*. Unfortunately, many other techniques for reducing I/O response times, such as file caching and request reordering, usually increase I/O path lengths.

One cost on which supercomputer applications have focussed is the conversion between internal (usually binary) and external (usually ASCII) formats [Abu-Sufah86, White84]. Significant savings are possible in this area because about 25% of a Fortran program's execution time is spent formatting [Ditzel80, Knuth71], and unformatted I/O can be as much as 130 times faster than formatted I/O in Cray Fortran programs [Cray88].

Offloading format conversion from the central processor is one example of the potential advantages of *peripheral*, or *I/O Processors* (IOP). IOPs can be as simple as *Direct Memory Access* (DMA) devices that copy data from peripheral devices to memory by "stealing" processor cycles [Astrahan57, Serell62], or they can be as complex as processing units with separate instruction sets and memory [Brown72, Thornton64]. Although many functions offloaded are best executed on IOPs because of their simplicity and frequency of occurrence, the decision is not as clear with more complex functions. Peripheral processors that have simple or inaccessible debugging features are also typically less powerful; code migrated to an IOP is more difficult to debug and frequently runs more slowly than on the central processor.

Extreme examples of offloading function from the central processor to improve a user's overall response time can be found in various processor-per-disk-track [Slotnik70] and processor-per-disk-head [Kannan78] database machines. Although these designs promise substantial parallelism to decrease overall execution time, they can also generate more work. For

[2] This technique works best when there is little locality in the disk workload.

example, using the *search* function in IBM disk controllers to select the disk sector with a matching key can be slower than a main memory search [Buzen75] because the search function requires each record be stored in a separate sector, wasting the data capacity of a track with intersector gaps. A main memory search, in contrast, allows each track to be packed with a few multiple-record sectors, thus allowing more data to be searched with each disk revolution. Perhaps most important, however, is the cost of these designs; special-purpose disks are more expensive and system costs are dominated by disk costs. The success of parallelism in database machines really depends of the availability of a vast I/O bandwidth [Boral83].

Disk designers have also introduced products with architectural modifications intended to reduce average access time. Disk transfer time can be sharply decreased by transfering from all magnetic surfaces in parallel if read/write circuitry, normally one module in a disk, is replicated for each surface [Bucher80, Kryder89]. To reduce average seek times, some disks have two heads on each surface positioned over different tracks [Massiglia86]. This configuration either restricts the number of tracks each head manages or provides disk scheduling software a choice of the closer head. Such disk-based architectural changes are expensive because they have generally been restricted to high-performance markets where the number of units sold is low and cost is high. They are also narrowly focussed changes; if a disk has eight parallel heads, it can transfer eight times faster, but not more or less.

A more flexible method for increasing architectural parallelism in secondary storage systems is the disk array [Jilke86]. Disk arrays have two powerful advantages: they increase the ratio of disk heads to user data so that disk contention is reduced [Patterson88], and they stripe files across multiple disks so that large accesses are serviced in parallel [Kim86, Livny87, Salem86]. Chapter 3 discusses in-depth the advantages of disk arrays and expectations for their performance.

2.1.5. Lowering Memory Miss Ratios

A powerful technique for overcoming the access gap problem by exploiting locality is file or disk caching [Ousterhout85, Powell77, Smith85]. Caching is a well-known and well-used technique for increasing hit ratios based on the principles of temporal and spatial locality. If the right, small subset of memory hierarchy level $l+1$ is kept in memory hierarchy level l, then the fraction of processor references satisfied at level l can be increased. This technique is so successful that most levels of the memory hierarchy now function as caches for slower and larger levels.

Main memory was first used in earnest as a form of cache for disk storage when program working space became larger than the available main memory. Although at first some contested that only a programmer could optimize the moving of program overlays between disk and main memory, today most of main memory is devoted to caching portions of each task's *virtual memory* automatically [Denning70]. Although virtual memory is a technique that allows programs to escape the confines of physical memory rather than a technique for caching disk resident objects, it is an important contributor to I/O traffic. Substantial research has been devoted to the algorithms for selecting which portions of what tasks should be kept in main memory [Denning80]. While it is unlikely that new methods for reducing virtual-memory disk traffic will be forthcoming, increasing DRAM density and decreasing DRAM cost-per-bit is leading to larger main memories. Unfortunately, the size of the average program is growing at 50% to 100% per year [Patterson90 pp 16]. Since this rate of growth is comparable to that of DRAM density, any lowering of page-fault rates is most likely to be the result of decreasing numbers of users per system.

As the size of main memory increases, designers of operating systems have begun to use portions of it to cache file data normally stored on disk. Files that are frequently used for research and engineering workloads tend to be small and short lived [Ousterhout85, Powell77]. For these files a relatively small portion of main memory can be quite effective for reducing disk traffic, especially when disks are on the other side of a local area network [Lazowska86, Nelson88]. Disk caching serves the same purpose as file caching but maintains the cache at the disk controller. Although this is a less efficient use of DRAM and can degrade performance by increasing controller utilization [Buzen82], it is easier to integrate with existing operating systems, it increases user confidence that recently written file data will not be corrupted by software crashes, and it avoids the problem of consistency that arises if multiple processors cache data in main memories they do not share [Grossman85, Smith85].

Disk or file caching in larger and cheaper main memories, unfortunately, will not compensate for the memory-disk access gap experienced by all applications programs. Database systems, especially on-line transaction processing systems, frequently access customer data from large databases in essentially random patterns [Garcia-Molina84]. For example, because a customer's bank balance is not accessed frequently, each transaction requires at least one disk access [AnonEtAl85]. In applications like these, response time is directly sensitive to disk performance and contention.

Scientific computing applications can also defeat caching because data objects are often significantly larger than main memory [Kim87a]. In such cases, Kung has shown that to match an increase in processor speed by a factor of α requires main memory size to increase by approximately a factor of α^2 for Gaussian elimination, and by a power of α for Fast Fourier Transforms (FFT) [Kung86]. Since main memory costs are a major portion of a system's total cost, it is prohibitively expensive to reduce I/O traffic this way. Additionally, as systems acquire more and faster processors, many scientific applications simply increase the problem size [Gustafson89]. This increase can lead to huge data objects spread across many processors. The Illiac IV was used in such a way, and it spent as much as 40% of its time loading and unloading data [Feierbach79].

Even in general engineering workloads where caching satisfies most read requests, there is still a need for high bandwidth for disk writes. For reasons of security, write-behind caches periodically write all recently written data because software crashes are too frequent to trust copies in main memory. Recently, effort has gone into designing a new file system, called the *log-structured file system*, that optimizes write bandwidth by structuring the file system as an append-mainly log [Ousterhout89, Rosenblum90]. This approach has the advantage that all writes should be sequential and large.

2.2. Rapidly Growing Processor Performance

Not only is the access gap problem between memory and disk far from solved, it is getting worse. Very Large Scale Integration (VLSI) technologies and multiprocessing are providing vast increases in processing capabilities per dollar [Bell85, Burger84]. The microprocessor developers at Intel [Gelsinger89, Moore75, Moore79, Myers86] have increased performance at a rate of 37% per year while holding chip costs nearly constant since they introduced their 8080 chip in 1974. They believe that "every concept proven useful in mainframe or minicomputers has migrated or will migrate onto the microprocessor." They also expect that multiprocessing, which has improved minicomputer and mainframe performance by 20% per year, will soon be used internally on a microprocessor, and they envision 2000 VAX MIPS on a chip by the year 2000. To achieve this goal, microprocessor performance must grow at about 65% per year during the 1990s. And rapid growth in processing power is not confined to microprocessors; Bell [Bell89] projects that multiprocessor supercomputer performance will grow at about 175% per year in the early 1990s.

As processor performance grows rapidly, comparable speedups in the execution of programs should be expected. Alas, if programs spend part of their execution time waiting for disk accesses to complete, overall speedup can fall far short of expectations. Amdahl [Amdahl67] noticed this phenomenum in the context of overall speedup in a parallel system when a portion of execution was serial. His observation, known as Amdahl's law, has been generalized to state that, "the performance improvement to be gained from using some faster mode of execution is limited by the fraction of time the faster mode can be used" [Patterson90 pp 8]. In the context of I/O, this means that improving only the processing speed of a system will not proportionally improve the execution rate of programs that perform I/O.

In contrast to processor performance, the access time for a magnetic disk, which is controlled mechanically, has been decreasing much more slowly. For example, in the last two decades, IBM increased the seek speed of their main disks by a factor of three, increased their rotational speed by 20%, and increased their data transfer rate by a factor of five [Harker81, IBM3380, IBM3390]. In other words, the time to seek from a random cylinder to a random cylinder, rotate half a revolution, and transfer four kilobyte decreased by 60% in 18 years – only 5% per year! Even if processor performance grows only at 37% per year for 10 years, processing speed should increase by more than a factor of 20, but disk speeds will not even double.

It is interesting to note that the growth of DRAM performance, about 7% per year [Patterson90 pp 426], is only slightly faster than the growth of magnetic disk performance, so the memory-disk access gap is not widening quickly, and the increasing gap between processor and disk speeds is actually widening the relatively small gap between cache and memory performance much more quickly. System designers have been compensating for this widening cache-memory access gap with larger caches to further exploit locality, with larger transfer units to amortize initial access overhead, and with increased memory parallelism to overlap multiple accesses. The success of these techniques passes the pressure of processor performance improvements onto I/O performance. Consequently, the same techniques need to be applied to I/O systems. The previous section discusses the application of the first of these techniques – the exploitation of locality with disk and file caching – and Section 2.1.4 discusses approaches for increasing transfer unit size using read-ahead and write-behind. The final approach, parallelism internal to the disk system, is one of the goals of disk array research. The success of this last technique is discussed in Section 3.4.

As the performance of the interconnection between processors and main memories is expanded, it widens the difference between processor-memory bandwidth and memory-disk bandwidth. I call this difference the memory-disk *transfer gap*. The transfer gap adversely affects the performance of applications that issue large I/O requests, and it hinders the performance of optimized designs such as the log-structured file system mentioned in the previous section.

2.3. New Opportunities in Secondary Storage Technologies

Although disk technology is not improving its performance as quickly as is processor technology, it has not been stagnant either. The industry of mass data storage is growing at a rate of 30% per year and currently accounts for 20% of computer revenues [Bortz90]. During the past 30 years the storage density of a disk's surface has increased at an average rate of 22% per year, and in the last 20 years the rate has increased to 26% per year [Frank87]. There is no reason to assume that this growth will not continue:

> The fundamental limit for information density in magnetic recording technology, for example, is at least four orders of magnitude beyond what has been achieved in the commercial devices, and laboratory demonstration devices indicate that there are no overwhelming obstacles to continued improvement at high rates for the remainder of the twentieth century [Bortz90].

In fact, IBM recently demonstrated a laboratory disk system with a storage density of over one Gbit per square inch, 18 times greater than that of their newly introduced top-end 3390 [Wood90b]. At a density growth rate of 26% per year, this technology will be "on time" even if it takes 12 years to reach the market. Because one Gbit per square inch is the projected norm by the year 2000 [Kryder89], the growth rate of surface density must increase about 30% per year during the 1990s.

Until recently, magnetic disk technology was driven by the high-performance computer market and produced expensive, large-diameter (14-inch) disk systems. The consumer market for personal computer storage products has changed the industry, however. For now and at least for the near future, a large consumer demand will continue to create a competitive market for high-density, small-volume magnetic storage products.

As the market for small-diameter disks developed, at least three trends became clear. First, small volume is much more important in small-diameter disks than in large-diameter disks, so aggregate small disks can achieve greater capacity in similar sized boxes [Jilke86]. Second, because small-diameter disks have shorter seek distances and lighter moving parts, "tolerance demands on electro-mechanical systems are more easily achieved in small form factors" [Bortz90]. Finally, sensitivity to price in the burgeoning personal computer and consumer electronics markets is streamlining the industry [Hoagland85]:

> Magnetic recording data storage thus not only has a level of R&D characteristic of a high technology, but also responds to a commodity market, more normally associated with a mature technology.

These trends have combined to change the magnetic disk:

(1) the large diameter disks (14-inch) are disappearing;

(2) the best price per megabyte is presently found in the 5¼-inch diameter disks and is expected to move to the 3½-inch diameter disks soon; and

(3) disk reliability, weight, and power consumption have all rapidly improved.

While these changes are auspicious for the personal computer and laptop markets, they seem incongruous when compared with the needs of large systems. Small-diameter disks have lower capacities and slower data transfer rates. These characteristics would appear to be exactly the wrong ones to satisfy the trend toward rapidly growing processing performance. However, many small-diameter disks can be packaged into an array with significantly superior performance relative to the large-diameter disks they replace [Jilke86, Patterson88]. This crucial feature of disk arrays is discussed in detail in Chapter 3.

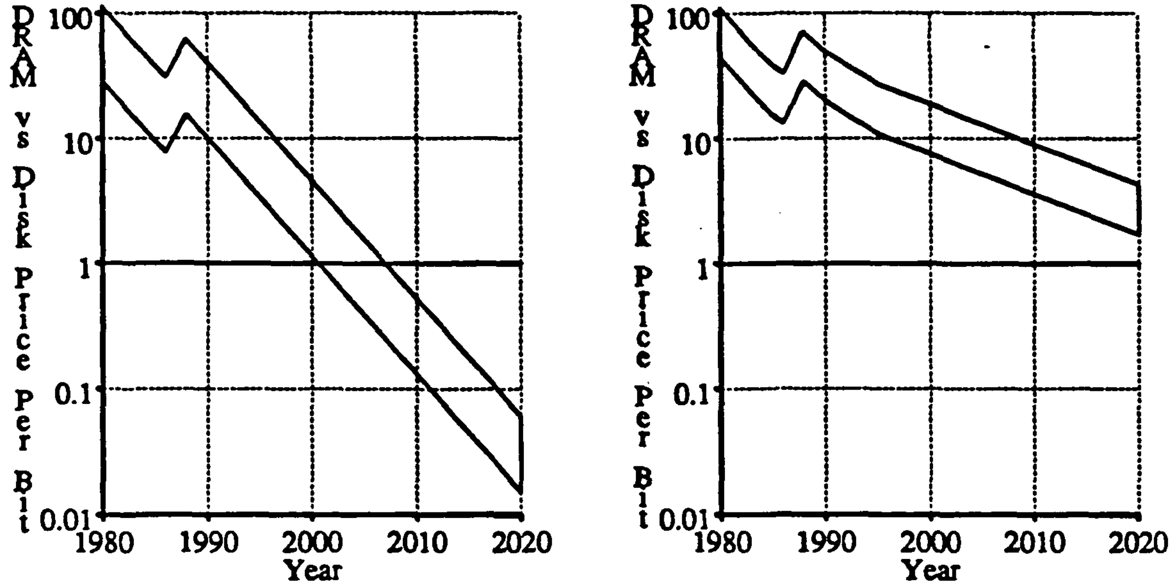
2.4. Replacing Magnetic Disks with DRAM: How Soon?

The ideal and, perhaps, inevitable way to make sure that programs run fast is to make main memory large enough to store all on-line data. Twenty years ago main memory was about 2,000 times as expensive per bit as a magnetic disk [Pugh71]. Today the difference is down to a factor between 10 and 40. Despite this large decrease in the relative cost of main memory, it is still too expensive to replace disks with DRAM. With 40% to 60% of all but the least and most expensive systems' cost invested in disk storage [Bodega89], replacing disks with DRAM increases the total cost of a system by a factor between 5 and 25!

The price gap between magnetic disks and DRAM has been rapidly narrowing. Between 1977 and 1986 the price per bit of DRAM decreased by 38% per year [Yeack-Scranton90] while disk prices per bit were decreasing by 23% per year [Hoagland89]. As Figure 2.1a shows, if this pricing trend continues for the next few decades then the price-per-bit of DRAM will cross disk's price-per-bit in 11 to 17 years (2001 to 2007). When DRAM pricing approaches disk pricing, their strong performance advantages will be sufficient reason to replace magnetic disks. From these estimates, the demise of magnetic disk storage should be expected in 11 to 17 years or less.

However, magnetic disks may survive much longer than 17 years. First of all, DRAM prices did not decrease at 38% per year during 1986 through 1988; in fact, prices rose in what is now referred to as "a temporary excess of demand relative to available supply" [Patterson90 pp 55], or in other quarters as "gouging" [Yeack-Scranton90]. This price increase had the effect (shown in Figure 2.1a) of stalling the crossover date by about five years. Since the disk market is relatively diverse and stable, high-stakes international battles for the DRAM market may lead to similar abnormalities in the future, each one delaying the crossover date still further.

Another obstacle to an early pricing crossover date is that equivalent capacity in the form of a collection of DRAM chips does not make a secondary storage system. In addition to basic



Figures 2.1a and 2.1b: Extrapolations for the Ratio of DRAM to Disk Price per Bit. *These two figures show the ratio between DRAM and disk price-per-bit against the year extrapolated to 2020. In Figure 2.1a on the left is an estimate favoring DRAMs. Using a range on the 1990 ratio of 10 to 40, it assumes that the rate of improvement for DRAM price-per-bit is 38% per year after 1988 and that the corresponding rate of improvement for disk price-per-bit is 23% per year. With this model, the date that the price-per-bit of DRAM crosses the price-per-bit of disks is between 2001 and 2007. In Figure 2.1b on the right is an estimate favoring disks. Using a range on the 1990 ratio of 20 to 50, it assumes that the rate of improvement for DRAM price-per-bit slows from 38% to 35% per year in 1995 and that the corresponding rate of improvement for disk price-per-bit rises from 23% per year to 26% per year in 1984, then to 30% per year in 1990. With this model, the date that DRAM price-per-bit crosses the price-per-bit of disks is between 2027 and 2040.*

enclosure, power, and cooling costs, replacing magnetic disks with DRAM should provide the familiar sense of confidence users experience when they know that their data is stored on a non-volatile medium. Although the technology to make DRAM as reliable as magnetic disks is not challenging – built-in battery and magnetic tape backup systems will suffice – the additional cost will further delay the pricing crossover date. Representatives from the disk industry estimate that in 1991 DRAM secondary storage will cost at least a factor of 50 more than the best price for 3½-inch magnetic disks purchased in large volumes. Beginning with this factor of 50 between DRAM and disk, the earliest crossover date is 2009.

Additionally, some people object to extrapolating the pricing trends from before 1986 into the 21st century. This objection stems from the fact that prior to 1986, magnetic disk prices were not under any pressure from DRAM. Restricting attention to the portion of the magnetic disk market that was under heavy pricing pressure, the small-diameter disks demanded in the personal computer market, disk pricing decreased at 26% per year [Hoagland89], faster than in the magnetic disk industry as a whole. This trend has been extrapolated to suggest that prices per bit will decrease by up to 30% per year in small-diameter disks [Bortz90]. At this rate, DRAM will take until 2022 to overcome its price-per-bit disadvantage, which now stands at a factor of 50.

As the DRAM price disadvantage decreases, it will capture an inflating portion of the disk market. If the magnetic disk market begins to shrink, then the number of disk manufacturers and the size of their profits should shrink as well, and the technology growth rate would probably slow. Why then is the demand for disk capacity growing at 40% per year [Hoagland88]? Today, only 1% of stored information employs a magnetic medium; about 95% is still found on paper [Kryder89]. While DRAM pricing chases disk pricing, both are rapidly decreasing; therefore, more and more data is worth storing on-line – a situation that provides lucrative growth opportunities for the magnetic storage industry as well as the DRAM industry.

Finally, taking a closer look at the technological obstacles that will be encountered by DRAM and magnetic disk technologies prior to the crossover of their price-per-bit, there may be reason to expect that this date will slide further into the future. Although there are no major obstacles to continued disk technology improvements over at least the next decade, DRAM technology will have to make the transition from optical lithography to ultraviolet and X-ray lithography [Hodges77, Kryder89, Warlaumont89]. Because of this transition, it is likely that the rate at which DRAMs will improve in price-per-bit will slow. It has been suggested that X-ray lithography will be used only in a niche market for high-performance, high-cost integrated circuits and, because continual reductions in feature size will aggravate reliability problems in semiconductor materials, that increases in DRAM capacity will soon come from alternative technologies such as multichip modules [Hodges90]. In this case, the rate of decrease in

DRAM price-per-bit will slow and the pricing crossover date may very well be delayed or cancelled. Figure 2.1b shows the trend in DRAM price-per-bit relative to disk price-per-bit for cases like these. DRAM's rate of price-per-bit decrease slows to 35% per year, and the crossover date is 2027.

Semiconductor technology may eventually provide secondary storage systems with lower costs as well as better performance than do magnetic disks, although probably not before 2000 and maybe significantly later. Until that time, exploding processor performance will exacerbate the already large access and transfer gap problems, placing even greater importance on research to Input/Output architectures.

2.5. Summary

In this chapter I have addressed the reasons for pursuing research into Input/Output, particularly into secondary storage on magnetic disks. Certainly, the gap between memory speeds and disks speeds has been and continues to be a major problem for computer systems.

Traditional approaches for coping with this gap include multiprogramming, asynchronous I/O, disk load balancing, rules of thumb that limit utilization, request reordering, contiguous file allocation, special-purpose disk devices, and file and disk caching. Because none of these solves the problem for all important applications, new solutions remain desirable.

In addition, the access gap problem is worsening because the rapid growth of computational performance induced by VLSI and multiprocessing is not being matched by performance increases in secondary storage technologies. There is also an emerging transfer gap between memory bandwidth and disk bandwidth that promises problems for applications that issue large transfers. The good news, however, is that the trend toward small-diameter magnetic disks provides a new opportunity for architectural parallelism in the I/O system, the disk array. In the next chapter, I will delve more deeply into the advantages and disadvantages of disk arrays.

Finally, alternative technologies, particularly solid-state disks, are not expected to be cost-effective replacements for magnetic disks for at least one and probably several decades.

CHAPTER 3

Redundant Disk Arrays

This chapter provides a broad understanding of redundant disk arrays. It focuses on the factors that make disk arrays an inevitability, the alternative encodings that provide failure protection for the array's data, and the performance that can be expected from redundant disk arrays. Much of the material in this chapter reviews previously published research of which I was an author.

In Section 3.1 of this chapter I explain how disk arrays exploit the emergence of high-performance, small magnetic disks to provide cost-effective disk parallelism that combats the access and transfer gap problems. The flexibility of disk-array configurations benefits manufacturer and consumer alike. In contrast, I describe in Section 3.2 how parallelism, achieved through increasing numbers of components, causes overall failure rates to rise. Failure prediction is very useful for anticipating many of these failures, but it cannot guarantee that data will not be lost. Data backups have traditionally been used for reducing the amount of data lost during failure. Backups are becoming unmanageable, however, because of the volume of on-line storage, the rate that on-line storage changes, and the requirement for minimal interruption of service during failure recovery. Redundant disk arrays overcome these threats to data reliability by ensuring that data remains available during and after component failures.

As far as the organization of redundant data in a disk array is concerned, I treat it as a coding problem in Section 3.3. The redundancy internal to a disk corrects non-catastrophic failures and identifies catastrophic failures, whereas redundancy at the disk-array level corrects catastrophic disk failures. Codes as simple as parity, which is not a single error-correcting code, can provide single-failure protection because of this internal redundancy and its ability to identify failed disks. More complex and expensive codes can be used to provide multiple-failure correction in very large or very reliable disk arrays.

In Section 3.4, I review the performance expectations for redundant disk arrays. Disk arrays derive their performance advantages by "striping" the data across multiple disks. The greatest benefit of striping is that it decreases transfer times for large requests. In addition, striping automatically distributes independent accesses to balance the workload across disks. Because each disk access involves substantial overhead, the unit of striping must be carefully chosen to avoid a mismatch with the array's workload. Redundant data reduces some of the performance benefits of data striping, however, because this redundant data must be updated as user data is updated. Without assistance from file system or application software, the main penalty to performance is as little as one and as much as three extra accesses that must be performed with every small, random access. In contrast, with a file system that groups small write accesses into large write accesses, an N+1-parity redundant disk array with block-interleaved striping can provide nearly all of the performance of its disks as well as inexpensive, high reliability.

3.1. The Emergence of Disk Arrays

The performance of secondary storage systems is not improving fast enough. In Chapter 2 I outline the access and transfer gap problems and discuss classes of important applications for which conventional approaches do not overcome these problems. Although magnetic disks

Characteristics	IBM 3380	Fujitsu M2361A	Conner CP3100
Disk Diameter (inches)	14	10½	3½
Formatted Data Capacity (MB)	7,500	600	100
MTTF (1,000 hours)	–	35	30
Number of Actuators	4	1	1
Max I/Os/sec/Actuator	50	40	30
Max I/Os/sec/Box	200	40	30
Transfer Rate (MB/sec)	3	2.5	1
Power/Box (W)	1,650	640	10
Volume (cubic feet)	56	3.4	0.13

Table 3.1: 1987 Magnetic Disk Technology Comparison. *This table compares the relative price, reliability, performance, capacity, volume, and power specifications for specific examples from three classes of disks. An IBM 3380 Model AK4 is a large, high-performance disk used extensively in mainframe computer systems. A Fujitsu M2361A "Super Eagle" is a disk of intermediate size and performance that is used in minicomputers and workstation file servers. A Conner Peripherals CP3100 is a small disk used in workstations and personal computers. Most data is taken from manuals [Conner3100, Fujitsu2361, IBM3380]. The 3380's capacity is based on a single 47,476-byte sector per track; if a 3380 is formatted with 512 byte sectors to match the smaller disks, its capacity is reduced to 51% of its listed value. Mean Time To Failure (MTTF) is not specified for the 3380. Its design goal calls for 98% of 3380 spindles to survive at least seven years [Mitoma90]. If spindles have exponential lifetimes, then the MTTF of either spindle in a 3380 is about 1,500,000 hours. Section 4.2 reports that 3380 units suffered failures at a rate of about one in six years [Balanson88]. This data suggests 3380 MTTF was about 50,000 hours. Maximum I/Os per second, per actuator refers to the maximum number of single-sector accesses per second that can be performed by each actuator, where each access requires a seek from a random cylinder to another random cylinder plus a rotational latency delay to a random location on the target track. The 3100's volume is based on the shelf dimensions in Berkeley's "RAID the Second" prototype: nine disks on a 19-inch by 30-inch shelf with a 3½-inch shelf pitch. The data in this table is derived from [Patterson87] and was collected in 1987.*

may eventually be replaced by much faster main memory, this eventuality is decades away, as is demonstrated in Section 2.4. Until that time, arrays of magnetic disks offer parallelism that can be exploited for substantial improvements to secondary storage performance. This section presents the reasons, including but not limited to increasing performance, why secondary storage systems must inevitably employ disk arrays.

From my perspective, the primary reason for adopting disk arrays into secondary storage is the parallelism that is achieved by an appropriate distribution of data over many disks. The large number of disks that enable this parallelism are by no means new to secondary storage. Traditionally, large collections of disks, sometimes called *disk farms*, arose to satisfy the need to store large amounts of data on-line. With capacity as the main reason for building a disk farm, its component disks are generally large and, as a result, expensive. Disk arrays, on the other hand, are composed of many small-capacity disks, so they deliver the same level of parallelism at a much lower capacity and cost or provide a much higher level of parallelism at the same capacity and similar cost.

To exploit the parallelism inherent in either disk farms or disk arrays, data must be distributed appropriately. Traditionally, the distribution of data across a disk farm gives rise to substantial differences in the utilization of individual disks [Friedman83, Kim87b]. To support a large throughput for the small, random accesses typical to database applications, data that is frequently accessed should be distributed or balanced over many disks as evenly as is possible. In disk farms, this is explicitly done by application programs or customers [Gray90, Livny87

Characteristics	IBM 3390	Seagate ST41600	IBM 0661	IBM WDA-260
Disk Diameter (inches)	10⅞	5¼	3½	2½
Formatted Data Capacity (MB)	22,700	1,350	320	63
MTTF (1,000 hours)	–	150-250	150	45
Number of Actuators	12	1	1	1
Max. I/Os/sec/Actuator	50	55	45	35
Max. I/Os/sec/Box	600	55	45	35
Transfer Rate (MB/sec)	4.2	3-4.4	1.6	1.1
Power/Box (W)	2,900	37	12	3
Volume (cubic feet)	97	1.0	0.13	?

Table 3.2: 1990 Magnetic Disk Technology Comparison. *Since the data in Table 3.1 was collected, IBM has introduced the IBM 3390 [IBM3390] as its new, top-end mainframe disk subsystem, the IBM 0661 model 370 "Lightning" [IBM0661] as its high-performance low-end disk subsystem, and, most recently, the IBM WDA-260 as its portable computer disk. With the 3390, IBM takes a step toward arrays of smaller disks. With the 0661 and WDA-260, IBM demonstrates that small disks can be almost as fast and reliable as their larger competitors. The 3390 capacity assumes maximum size sectors; see the note about 3380 capacity in the caption to Table 3.1. Again, as with the 3380, the 3390 does not have a specified MTTF, but each spindle is designed to survive seven years with a 98% probability. This suggests that the 3390 MTTF for spindle failures is designed to be about 500,000 hours; including failures in the electronics can be expected to substantially reduce (perhaps more than halve) this MTTF. Seagate's ST41600 "Elite" is one of the largest and highest-performing 5¼-inch disks [Seagate90]. The range of its MTTF distinguishes an office environment from a computer room environment and the range of its transfer rate distinguishes the amount of data stored on the inner tracks from the amount stored on the outer tracks. Volumes for the 5¼-inch and 3½-inch disks are taken from Berkeley's disk array prototypes. The caption in Table 3.1 describes the 3½-inch volume calculations. For the volume of the 5¼-inch disks, Berkeley's prototype disk array, called "RAID the First," contains 24 5¼-inch disks in a six-foot high, 19-inch by 30-inch cabinet (one cubic foot per disk).*

pp 72], but in disk arrays, it is automatically done by *striping* data over multiple disks [Salem86]. Striping can also counter the transfer gap problem experienced by workloads that feature large sequential accesses [Kim87a, Reddy89]. This is done in Cray supercomputing systems by striping data on their swap storage disks [Johnson84]. In Section 3.4 I examine striping's exploitation of disk parallelism for improving performance in more detail.

A second reason for the inevitability of secondary storage systems built of disk arrays is the impending demise of large-diameter disks. Section 2.3 describes how the state-of-the-art in magnetic disk technology has moved from large, high-capacity disks to smaller, lower-capacity disks. Tables 3.1, 3.2, and 3.3 show particular comparisons of some of the premier products of each diameter. With IBM's move to 10⅞-inch disks in its top-end product, there is no longer any active development of 14-inch disks. Moreover, the performance of each of the 10⅞-inch disks in IBM's 3390 is inferior to Seagate's 5¼-inch Elite disk. These comparisons are indicative of the overall trend in the disk industry; by most metrics 5¼-inch disks are currently the best, and most disk designers expect that 3½-inch disks will soon be supreme. But if large-capacity disks are being phased out and if customers' requirements for on-line storage continues

Disk Diameter (inches)	10⅞	8	5¼	3½	2½
Disk Unit Cost ($/MB)	$1.00		$1.25	$1.60	$4.00
Disk Unit Price ($/MB)	$6-$10	$1.80	$1.60	$2.50	$8.10

Table 3.3: 1990 Magnetic Disk Cost and Price. *This table shows the variation in cost per megabyte [Mitoma90] and the price per megabyte [Lomas91] of different diameter disks. Cost numbers include disk electronics and are based on industry wide analyses (they are not necessarily IBM costs). Price numbers for the smaller four diameters are based on first-quarter 1991 discount rates available on large purchases from Seagate. For the price per megabyte of 10-inch products, the IBM 3390 has a list price of about $10/MB [Mitoma90]. The range I report assumes that IBM's best customers are able to get discount rates as large as 40%. Magnetic disks achieve a low cost per megabyte because disk platters are inexpensive and, in 1990, contain 50 to 80 megabits per square inch. Although 5¼-inch and 3½-inch disks contain much less recording surface, their prices can remain low because of very high manufacturing volumes. It is estimated that in 1989, 12 million 3½-inch disks and 6.9 million 5¼-inch disks were shipped, but only 0.6 million 8-inch or larger disks were shipped. The 2½-inch disk costs are high now because this product has not yet been produced or shipped in sufficient numbers. By 1992, the cost of 2½-inch disks per megabyte should be below two dollars [Mitoma90].*

to increase as rapidly as it has over the past decade, the number of disks in secondary storage systems must swell.

Although replacing large disks with many smaller ones is inevitable, rapidly expanding disk farms are logistically undesirable. For example, Table 3.2 shows that an IBM 3390 unit contains 12 separately-accessed *volumes* with a total storage capacity of up to 22.7 gigabytes. Seventy-one, 320-megabyte, 3½-inch IBM 0661 disks are required to replace each IBM 3390; hence cabling, controllers, and administrative domains are all increased by a factor of six! Fortunately, disk arrays resolve these problems.

The *disk array* alternative is suggested by packaging because a single package with the storage capacity of previous large-diameter disks, but built with smaller capacity and smaller diameter disks, contains an *array* of disks [Jilke86, Kim86, Patterson88]. By using striping to make many disks appear as a single, larger disk, a disk array provides the opportunity for access and data parallelism without intervention by users of arrays or by administrators. The number of cables and controllers can be kept from growing as quickly as the number of disks by connecting many disks to the same interconnect and controller. This sharing of cabling and controllers can be problematic, however, because of the delays caused by the RPS reconnect misses that arise when multiple disks share the same cable and controller. This problem, described in Section 2.1.4, can be overcome with buffering and intelligence embedded into each disk.

Fortunately, most small-diameter disks connect to their controllers across *intelligent interfaces*, such as the industry standard Small Computer System Interface (SCSI) [ADS85, ANSI86] or Intelligent Peripheral Interface (IPI) [Allan83, ANSI87], or proprietary protocols such as Digital Equipment Corporation's Mass Storage Control Protocol (MSCP) [Massiglia86]. Intelligent interfaces are feasible in inexpensive disks because a significant portion of a disk's controller function can be merged into a small number of VLSI chips and embedded in the disk package at little additional cost. With a dedicated controller and a small amount of buffering in each disk, the disk-to-computer interconnect can be designed independently from the disk recording and reading process. This allows multiple devices to share a fast interconnect without the delays associated with RPS reconnect misses.

A third reason for the emergence of disk arrays as the preferred architecture for secondary storage components is the opportunity for manufacturers to spread research and development costs over a wide product line. Traditionally, a range of price and performance products is achieved by offering a variety of disk products that are designed separately with different diameters. Figure 3.1 shows how disk arrays can be configured with a variety of different numbers of component disks to provide a complete family of products all based on the same small-diameter disk. With disk arrays, a vendor can concentrate design talent on the single, small-diameter disk that will be used in all products [Mitoma89]. Thus, the required ranges of price and performance can be provided in much the same way that the design of a single processor can provide a range of computing power in a family of multiprocessor products. In this way, disk arrays reduce the development cost of a secondary storage product and shorten its time to

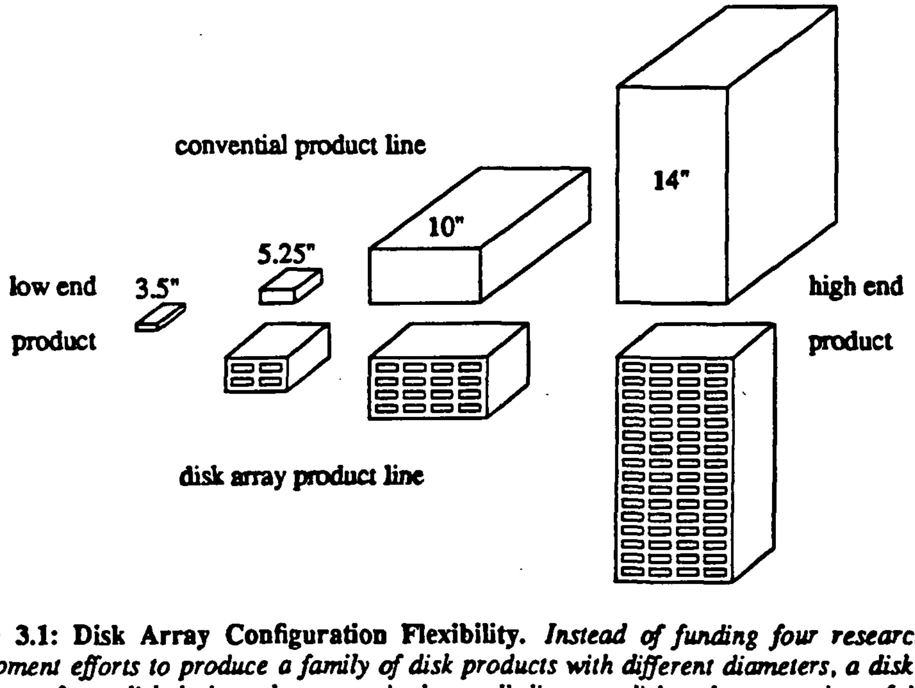


Figure 3.1: Disk Array Configuration Flexibility. *Instead of funding four research and development efforts to produce a family of disk products with different diameters, a disk array vendor can focus disk design talent on a single, small-diameter disk and use a variety of disk array configurations to provide a product family.*

market.

The fourth major reason for relying on disk arrays in secondary storage is the opportunity they provide to achieve high-reliability at low-cost. Conventional systems offer two choices for the control of reliability; either accept the inherent reliability of a disk or continually maintain a duplicate of its data on a second disk that can replace the first in the event of its failure. If the inherent reliability of a disk is not high enough, customers are obliged to double their costs and take the possibly overdesigned, much higher reliability that comes with duplication. Because disk arrays contain many more disks operating in conjunction and centrally controlled, a less expensive "N+1" parity encoding for redundancy can be employed to achieve high reliability. This encoding maintains the parity of N disks on a single parity disk that can be used to recover the contents of any single disk failure. For disk arrays organized into redundancy groups with 10 data disks, this approach reduces the overhead cost for high reliability from 100% to 10%. Redundancy in disk arrays is the theme of the rest of this dissertation; its encoding choices, performance expectations, and reliability estimations are covered in detail in later sections and chapters.

Another advantage of the disk array approach to parallelism in secondary storage is its physical space efficiency. Because small disks have a large capacity per unit volume, a disk array can provide a large capacity and a high degree of parallelism in a small box.

For the aforementioned reasons many vendors have products in the disk array market. They include Array Technology, Auspex, Ciprico, Compaq, Cray, Datamax, IBM, Imprimis, Intel Scientific, Intellistor, Maximum Strategy, Pacstor, SF2, Storage Concepts, Storage Technology, and Thinking Machines. Some customers, notably the NASA Ames NAS project and the Minnesota Supercomputer Center, have developed internal redundant disk arrays to meet their I/O needs [Klietz88, Poston88]. Nevertheless, this market is developing slowly, in large part because incompatibility problems at the application, operating system, controller, and disk interface levels blocks full exploitation of disk arrays. Moreover, customers unfamiliar with the multitude of tradeoffs in the configuration of disk arrays, such as the appropriate redundancy versus performance versus cost tradeoff, have contributed to a general "wait and see" attitude. In the last year, however, market forecasts have improved dramatically [Devlin90]. The installed base is nearing 4,600 arrays, mainly in local area networks for personal computers, and is projected to be 10,000, 30,000, 60,000, and 110,000 in 1991, 1992, 1993, and 1994 respectively.

3.2. Disk Arrays As a Threat To Data Reliability

Although the large number of disks employed by a disk array improves performance, it also threatens data reliability since systems with more parts have more frequent failures. If a disk array's performance is an order of magnitude larger than that of a single large-diameter disk because the array employs an order of magnitude more disks, then the rate of failures in an array may also be an order of magnitude larger than that of a single disk. Some form of compensation for these higher failure rates is required since large increases in disk parallelism are unlikely to succeed commercially if secondary storage data reliability is adversely affected. Before examining such compensation schemes, I discuss the failure modes of magnetic disks.

Disks fail in a variety of ways [Glover88, Schulze89]. The most frequent failures are individual bits read incorrectly off the surface. These transient errors are handled by circuits internal to each disk using a "check sum" code computed and stored at the end of each sector. Usually this code allows some bit errors to be corrected directly, rather than waiting for the much slower process of rereading.

The frequency of undetected or miscorrected errors is and must be low because such unnoticed errors will wreak havoc in an application. For example, the WREN IV 5¼-inch disk miscorrects less than one sector in 10^{21} bits transferred [CDC88]. While the detecting and correcting codes associated with each sector of existing disks suffer undetected or miscorrected errors with low probabilities, increasing processing speeds increase the rate at which data stored on disk is accessed, and this increases the opportunity for bit errors to occur. For this reason, and because each new generation of disks packs bits closer together, progressively more powerful codes are introduced into new disk products.

The frequency of detectable, but uncorrectable, permanent errors is higher than the frequency of undetected or miscorrected errors. Detectable, permanent errors are usually detected by factory tests. Once identified, the sectors containing these errors are marked so that the disk will use an alternate sector anytime a user request attempts to access a defective sector. From time to time, these flaws "grow" during operations leaving the user with a sector of corrupted data. Most disks allow these bad sectors to be dynamically remapped to other sectors nearby, but data is still lost. In addition to protecting against data loss on the occurrence of catastrophic failures of entire disks, redundant data schemes described in Section 3.3 also handle the loss of individual sectors quite well by recovering their contents from other data and parity disks.

Data coming from or going to disks can also suffer transient bit losses on datapaths or in buffers. Parity, or more powerful codes, on each path or buffer can capture most of these problems. Systematic failures in latches or error-detection/correction hardware, however, are more serious problems. Higher-level, end-to-end error-detection codes are the best method of handling these problems [Glover88, Massiglia86 pp 148 and 220]. Additionally, by embedding end-to-end detection codes into user data blocks, there is a second opportunity to detect those rare but important bit errors that the disk's internal error-correcting logic either did not detect or miscorrected. Because of this feature, per-sector and end-to-end codes should be designed to complement each other's detection and correction capabilities [Glover88 pp xiii].

Finally, there are complete failures of disk electronics or head-disk-assembly failures, such as head crashes. Such catastrophic failures are self-identifying, either by internal failure detection, end-to-end tests, or device-interface protocol violations. Although many catastrophic failures do not actually destroy data, long repair periods can be as damaging because "the greatest fear that an on-line system user has is that 'the data base is down' [Dolotta et al., 1976]'' [Katzman77 pp 450].

In this dissertation, I am mainly concerned with the effects of catastrophic disk failures on the reliability of data stored in disk arrays.

3.2.1. Avoiding Catastrophic Failures with Prediction

The occurrence of transient and correctable errors during disk operations provides information that may be used to diagnose the status of individual disks and to potentially predict imminent failures. Prediction is a special case of the diagnosis of computer system malfunctions from component-error reports logged in a central file [Billmers84, Tendolkar85].

Lin's [Lin88] examination of on-line diagnosis for failure prediction delineated two approaches to automatic diagnosis: *specification-based* and *symptom-based*. Specification-based diagnosis employs an expert system to apply collected information to an abstract model of the target machine's structure. This type of diagnosis has not been widely used because of restricted fault models and extensive learning times.

Lin focused instead on symptom-based diagnosis because trends in system behavior are good predictors of system failures [Lin88 pp 6]:

> Research in trend analysis is based upon two observations. First, most failures are preceded by a period of deteriorating behavior prior to turning into a permanent failure [Tsao83]. Furthermore, the occurrence of intermittent faults increase as circuits age [Breuer76]. Second, any sufficiently ''large'' system always has some latent problems [Shebell85]. If any of these cases can be detected at an early stage through extensive error logging, warning can be issued prior to catastrophic failure, and service can be started to minimize system unreliability and maximize the system's *effective* availability.

Lin presented a set of heuristic rules for predicting the imminent failure of a component based on logged errors. These rules, called the *''Dispersion Frame Technique,''* look for two failures in one hour, four failures in 24 hours, or three conditions that indicate increasing rates of errors [Lin90]. Lin applied these rules to data collected from 20 workstation-years of error logs taken from 13 Sun2 and Sun3 file servers. Without additional kemel instrumentation to collect detailed error records, the rules had little prediction success. With such additional instrumentation, however, 15 out of 16 permanent failures would have been predicted with only five false alarms. Although this data sample is small, it suggests that data reliability may be enhanced with intelligent error-monitoring software.

To this end, DEC has developed a monitoring and diagnostic tool, the VAX System Integrity Monitor PLUS (VAXsimPLUS), initially addressing only magnetic disk failures [Emlich89]. It is built using the earlier, knowledge-based diagnostic tool SPEAR [Billmers84]. The rules used by VAXsimPLUS are also heuristic and based on daily error-collection statistics. Depending on the type of error, diagnosis is triggered when a single day's error count exceeds a 25-day average by more than a given threshold. Once triggered, diagnostic tests are performed on various components until a failure theory is developed and a faulty field-replaceable unit can be predicted.

Because DEC is sensitive to its customer's perceptions of quality and to the cost of field visits, false prediction and excessive reporting are highly undesirable. Towards these goals, their diagnostic tools have been quite effective. For example, during seven million disk hours, VAXsimPLUS identified 150 disk failures and generated only three false alarms (2% of all triggers were false alarms). Although a faulty component was correctly predicted in 95% of cases, in those cases where data was at risk only 85% of the predictions were made early enough to allow a disk copy to be made [Lary89].

Predictive diagnosis is one component in a data reliability strategy rather than a complete solution on its own [Emlich89]. It is limited because some errors will escalate to failures too quickly to be predicted and averted, and because even under direction, human intervention to affect repair is itself prone to error. Moreover, prediction does not provide any guaranteed level of data reliability. For these reasons, predictive diagnosis must be augmented by more direct, data-protection mechanisms in a highly reliable secondary storage system.

3.2.2. Protecting Data Reliability with Off-Line Redundancy

The primary method of compensating for increased secondary-storage failure rates engendered by increased disk parallelism is to ensure that failure does not destroy data. Data, such as program output files, may be entirely re-creatable in specific cases. Although this recovery procedure can be expensive and prone to error, it is not uncommon for systems to rely on manual re-creation of at least the most recently modified data. Fortunately, most computer systems depend on one or more forms of redundant data storage for longer-term protection.

One form of data redundancy employed in most systems is off-line, or *backup* copies, of secondary storage contents. Recording backup copies limits the amount of data exposed to loss to incremental data changes in the event of a disk failure. Backups are usually made when computers are not in use, and thus the time interval between the capture of backup copies is usually fixed. Therefore, backups cannot easily compensate for large increases in secondary storage failure rates.

In fact, backups are becoming less effective and less desirable for several reasons. First, the opportunities to make a backup copy of secondary storage without affecting user performance are decreasing because computer systems are being used more continuously. This increasing use of computer resources may result from offering 24-hour service to users or from judiciously scheduling batch computation into time slots that are commonly under utilized (such as the middle of the night). Each backup can also be expected take longer to capture as computers become faster and generate more new data per day. Finally, capturing a backup copy onto off-line storage oftens involves expensive and error-prone human intervention. Thus, not only do backups fail to compensate for increases in secondary storage, but the desire to reduce the dependency on backups encourages secondary storage to become more reliable.

3.3. On-Line Redundancy: Encoding Data into Disk Arrays

Since off-line redundancy will not solve these increasing failure rate problems, automatic redundancy is needed within secondary storage. With this class of failure protection, user data is *encoded* into a form that occupies more storage capacity but allows data lost during certain combinations of one or more catastrophic disk failures to be recovered from the surviving disks.

Encoding data to increase the probability that it can be reliably transmitted or stored, usually called *error-correcting* or *error-control coding*, is a well-developed branch of mathematical theory. There are many good treatments of introductory [Tang69, Arazi88], computer-specific [Rao89], and comprehensive [Peterson72, MacWilliams77] error-control coding. The search for codes appropriate to particular transmission channels or storage mediums began in earnest with Shannon's fundamental theorem [Shannon48]. This theorem states that the probability of erroneously decoding data encoded and transmitted through or stored in a noisy environment can be made arbitrarily small, provided that the ratio of the amount of user information to the size of resulting encoded data is less than the inherent "capability" of the transmission or storage channel. Because the characteristics of transmission channels and storage media change with technology, research into appropriate codes is continuous.

Multiple metrics are needed for the selection of a strategy for encoding redundant data onto the disks of a redundant disk array. The effect on data reliability remains a primary concern because it is the reason for introducing redundancy. In error-correcting code theory, the basic metric for a code's reliability is the number and type of errors that decoding is guaranteed to correct. When the location of the error can be identified by some mechanism external to the code, then the error is called an *erasure*, and its correction is generally easier than the correction of an error whose location is not externally identified. As Section 3.2 explaines, catastrophic disk failures are self-identifying so the codes employed in redundant disk arrays correct erasures instead of arbitrary errors. In the next section, 3.3.1, I present alternative organizations for data redundancy that correct single-disk erasures, and then in Section 3.3.2, I examine double-erasure-correction encodings that may become necessary if large numbers of disks are used or if users require very high data reliability. In these discussions I will review and follow the constraints and terminology of Gibson, Hellerstein, Karp, Katz, and Patterson [Gibson89b].

In addition to a code's ability to enhance reliability, there are three other metrics that I will use to measure the quality of a code:

(1) a code's affect on user performance during normal operation should be minimized,

(2) its need for additional storage capacity should be minimized, and

(3) its affect on user performance during the recovery of the contents of a failed disk should also be minimized.

The first of these metrics should be used to insure that during normal operation, when all disks are fully functional, the performance penalty for maintaining redundant data is minimal. To this end, all codes should record user data in an unencoded form so that reads suffer no performance penalty. Because writes must affect redundant data as well as user data, codes should minimize the number of additional disk accesses required for maintaining redundancy. As a measure of this cost, the number of additional disks updated when one byte of user data is changed is called the *update penalty* of that change and, where different changes induce different update penalties, the maximum update penalty for any data change is the update penalty of the code.

The second quality metric applies to the additional storage required for redundant data. I measure the ratio of the amount of additional capacity used to store redundant or "check" data to the amount of capacity needed to store unencoded user data, and call this metric the *check disk overhead*. Of course, an encoding does not operate on an entire disk at time. Rather it treats a disk as a sequence of symbols[1] that are separately encoded. A *word* in the code or *code-*

[1] In the binary codes described in this section, a symbol is a single bit. However, in the non-binary codes of Section 3.3.2.2, a symbol may be two or more bits.

word is the set of symbols, one from each disk, that are related because the check symbols in this set are determined by the values of the set's data symbols. In any codeword, certain disks contain check symbols and others contain user symbols, so I can count the ratio of check disks to non-check disks. This ratio does not change from codeword to codeword, although the identity of the disks containing check symbols may change (see Section 3.3.1.2). For this reason, I measure overhead in terms of check disks instead of check symbols or check megabytes. In the terminology of information and check symbols, blocks of unencoded user data are *information* blocks and blocks of redundant data are *check* blocks. Clearly, codes with low check disk overheads are desirable.

My third quality metric is concerned with the impact of failure recovery on user performance. Although erasure-correction encodings allow users to access their data while a failed disk is repaired or replaced and then recovered,[2] many systems will not use this feature because it is simpler to halt operation until the correction is complete [IBMAS400]. Some applications, however, have such high requirements for availability that they will be sensitive to a redundant array's performance for user accesses while it is recovering a failed disk. A detailed model of performance during recovery has been developed for a version of N+1-parity encoded disk arrays [Muntz90]. For the purposes of comparing encodings of redundant data, I will instead use a simpler measure: the the minimum number of disks required to recover a single failure, called a disk's *recovery group size*. This measure indicates the proportion of the array that is degraded during a particular correction. The maximum size of a recovery group across all disks is equal to the size of the code's recovery group. The size of a disk's recovery group also indicates the number of disks that are more vulnerable to loss of data caused by subsequent failures during correction; larger recovery group sizes are likely to reduce data reliability. Chapter 5 is devoted to estimating reliability as a function of parameters that an array designer manipulates

[2] Recovery is the process of rederiving the contents of a disk that has been repaired or replaced; however, sometimes the term repair includes both of these steps.

such as the recovery group sizes in a single-erasure correcting, redundant disk array.

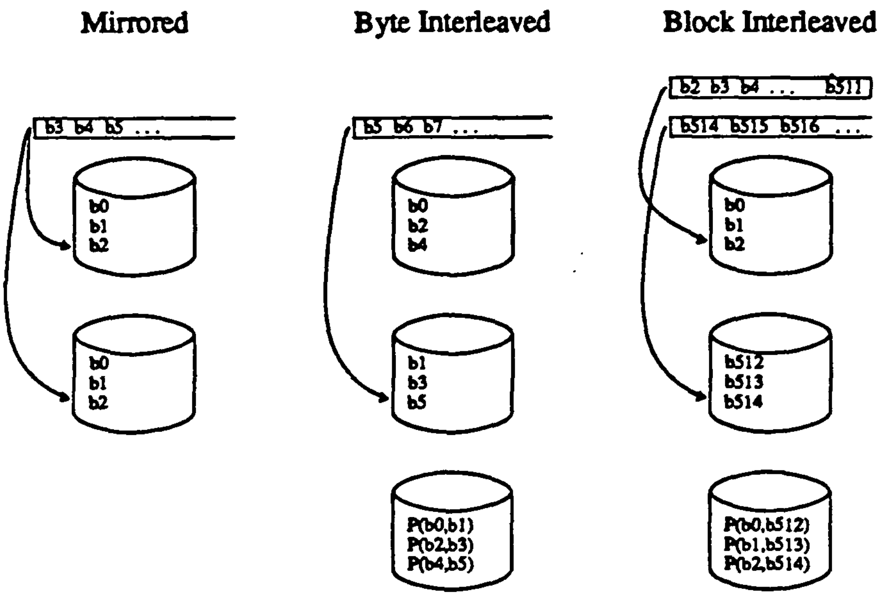
3.3.1. Single-Erasure-Correcting Encodings

Since the probability of catastrophic disk failure in any short period of time is small, the probability of a second failure during the repair and recovery of a first failure is also small. Hence, the vast majority of repairs will be applied to single failures. The probability of a second failure is increased, however, if there are many disks in the array or if the first failure causes the second. These two cases are addressed in Sections 3.3.2 and 5.5 respectively. In this section, I examine the codes for redundant disk arrays that correct all single-disk failures.

I begin by describing a full duplication code commonly called *mirroring*. This code has good data reliability, simple and fast normal operation, and fast recovery, but it also has a high overhead cost. Less expensive codes based on simple overall parity, called N+1-parity codes, exploit the nature of catastrophic disk failures. Because allocation of user data to disks affects the operation of an N+1-parity redundant disk array, I address byte-interleaved striping, block-interleaved striping, and non-striped data allocation separately.

3.3.1.1. Mirroring

Protecting disk subsystems from loss of data caused by the failure of a single disk is often accomplished by the complete duplication of all data. Shown in Figure 3.2a, this method has been alternatively called mirroring [Katzman77, Gray90], duplexing [Dishon88, Ng91], and, in its more general, multiple-copy form, shadowing [Bates89, Bitton88]. It is fairly common in systems with extremely valuable data. Each data write is applied to both disks in a pair and, in some systems, each data read chooses the disk with the shorter seek distance to the data [Bitton88, Gray90]. High data reliability and availability is achieved when complete duplication of data is accompanied by complete duplication of control software, datapaths, controllers, channels, and hosts [Katzman77]. Unfortunately, 40% to 60% of a system's costs are often attribut-



Figures 3.2a, 3.2b, and 3.2c: Striped Disks with N+1 Parity. *(a) Mirrored disks store a copy of each byte on two identical disks, (b) byte-interleaved disk arrays allocate successive bytes (b0, b1, ...) alternately to data disks (two in this case), and (c) block-interleaved disk arrays allocate successive bytes to the same disk, switching to another disk at the end of the block (with 512 bytes per block in this case). In the latter two cases, parity (P(x,y)) is computed from corresponding units on the physical disks, though the logical address of these units is different in the two schemes.*

able to its magnetic disks [Bodega89]. Therefore, complete duplication of may increase a system's costs by 40% to 60%.

Although a mirroring code is so simple that no further explanation is necessary, I present its analysis here to demonstrate the logic of the steps which I follow for the other cases.

Each word of a mirrored code contains two bits; the check bit is both an identical copy of the information bit, and, because the parity of one bit is the same as the value of that bit, the check bit is also the parity of the information bit. Because each codeword's check bit is identical to its information bit, the update penalty associated with changing a codeword's information bit is a single write to correspondingly change the codeword's check bit. With a codeword of two bits, half stored on one disk and half stored on another, the check disk overhead is 100%.

When a disk fails, one-half of every codeword stored in this recovery group of two disks may be destroyed. If the failing disk was not identified by mechanisms internal to the disk, then a mirrored code would not provide failure correction because the non-failed bit in each codeword would not be identified. However, disk failures are identified by mechanisms internal to the disk, so the location of the non-failed bit in each codeword is known, and, consequently, the value of the bit lost from each codeword is also known. With a mirroring code, data is not lost until the second disk in a recovery group fails before its contents can be copied onto a replacement disk for the recently failed, first disk in the same group.

3.3.1.2. N+1 Parity

A less costly approach to single disk-failure protection can be achieved by using a *parity code* [Arulpragasam80, Park86, Patterson88]. In a parity code, also called an N+1-parity code, each codeword contains one bit from each of N data disks and one bit from a single redundant data, or parity, disk. The value of the bit from the redundant data disk is equal to the exclusive-or of bits from the data disks. If a disk is erased, a single identified bit is lost from each codeword; the value of the lost bit is simply the value necessary to make the parity of the data bits equal the value stored in the parity disk. While N+1 parity decreases the check disk's overhead from mirroring's 100% to 1/N, it increases the size of the recovery group from two to N. The update penalty, however, is the same as its value for the mirroring code, because there is only one parity bit in each of this code's words.

In a mirroring code, the relationship between the contents of the two bits in a codeword is clear: both bits contain the same data. In contrast, in the N+1-parity code it is not clear whether the bits of a codeword are consecutive, regularly separated, or arbitrarily separated in the file system's address space. These three alternatives correspond to disk arrays that are striped with a single bit interleaving unit, striped with a block interleaving unit, and not striped, respectively. Because these three alternatives include most disk arrays of practical interest today, the next three sections look at each more closely although the performance of redundant disk arrays is not addressed properly until Section 3.4.2.

3.3.1.2.1. Byte-Interleaved Striping

A straightforward implementation for N+1 parity would follow the example of memory and bus data protection and bit-interleave data over the disks in an array [Kim86]. In practical systems, however, data is more likely to be byte-interleaved because controller electronics are more commonly designed to operate on bytes. Because an interleave unit of one byte is much smaller than the minimum unit of modification on a disk (a sector), the characteristics of byte-interleaved striping and bit-interleaved striping are essential the same.

Figure 3.2b shows an example of a small, striped disk array that interleaves data in byte units across all disks. Because the interleaving unit is small, the information bits required to compute an individual parity bit are spread over a small span of user data. Thus, parity can be computed in a small buffer as data streams into the array. With this organization, the system views an array as a single logical disk, and each access involves all disks.

Disk arrays employing byte-interleaved striping should synchronize the spindle motors on each disk. This can be seen because each request into such a disk array is striped over all disks at the same angular location, so there is no advantage to and considerable penalty from operating each disk independently. The penalty results from the fact that if N disks start searching a track on their respective disks at random angular offsets and the data they are searching for is at the same angular offset on each track, then the last disk to reach the data sought will have searched an average of N/(N+1) tracks. On the other hand, if the spindle motors on all disks are synchronized, then each disk begins and ends its search at the same location, searching 1/2 of a track on average. With data striped over 10 or more disks, on average, the slowest of unsynchronized disks requires 80% more searching than do synchronized disks.

The byte-interleaved organization presents an interface to its system similar to that of conventional disks, but it transfers data much more quickly. Because N of the N+1 disks in each recovery group contain data, the data transfer rate for a request striped over one redundancy group is N times larger than the transfer rate of a single disk. With synchronized spindles, a conventional interface, and a constant, fast transfer rate, a disk array with byte-interleaved striping functions as though it was a single disk with densely packed bits (fast transfer, high capacity, but the same rotational delay). Therefore, the byte-interleaved organization is the disk array implementation of a *Parallel Transfer Disk* [Bucher80, Fujitsu2361, Kryder89] with the addition of disk failure protection. Many of the initial offerings for commercial disk arrays, for example, the Micropolis 1804, the Storage Concepts 51, and the Imprimis Arraymaster are of this type.

An important weakness of this type of disk array organization is the parallelism it wastes by forcing every access to involve all disks. Because the time required to seek to a random track and then search for specific data is usually much larger than the time it takes a single disk to transfer one or a few sectors, the time taken by this type of disk array to service a small, random access is little better than the time taken by a single disk. In contrast to executing only one of these small, random accesses at a time on a byte-interleaved disk array, a non-striped collection of N+1 disks can execute N+1 small, random requests in about the same amount of time. Therefore, a disk array with byte-interleaved striping is unable to exploit its inherent parallelism to attain a high throughput for small, random requests. Exploiting parallelism for these workloads is the primary motivation of the organizations in the next two sections.

3.3.1.2.2. Block-Interleaved Striping

For arrays to support multiple, independent, small accesses in parallel, each access should be confined to a single disk. Figure 3.2c shows that this can be done by allocating a block of logical data to each drive. The unit of interleaving, a *block*, is one or more disk sectors in this configuration.

Because the bits of one codeword in this code are widely but regularly separated in the file system's address space, a small buffer on the datapath will not suffice for computing parity as the data blocks are transferred to the disks. Instead, the computation of parity must be done in a buffer at least as large as the unit of interleaving under the control of disk-controller firmware or device-driver software.

For writes large enough to change the entire "stripe" of one data block on each information disk, firmware or software can precompute the new parity while the new information is in memory, and the disk bandwidth spent to maintain the parity code is no more than that required to overwrite the associated parity data. For small writes, however, parity management can cost as much as four physical accesses: a pre-read of both old information and old parity, a write of the new information, and a write of the new parity. The new parity is then computed as the exclusive-or of old parity, old information, and new information. Section 3.4.2.3 discusses optimizations that reduce the cost of small writes.

Another problem arises if all parity data is located on one physical disk. Since all writes would have to access this disk, its write workload would be increased by a factor of the number of other disks, creating a bottleneck. As shown in Figure 3.3, however, the location of parity can be shifted, and the bottleneck reduced on successive rows of one block from each disk.

This block-interleaved organization of redundant arrays is currently employed by the NASA Ames NAS project [Poston88] and has begun to appear in commercial products such as Array Technology's RAID+. Lee and Katz [Lee91] examined how performance is affected by

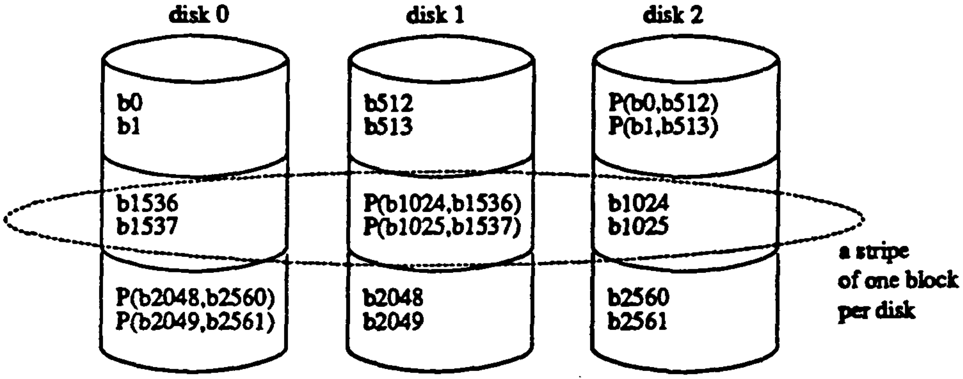


Figure 3.3: Striped Disks with Block-Interleaving and Distributed Parity. *To avoid a small-write bottleneck on the parity disk in a block-interleaved redundant array, successive stripes (the set of corresponding blocks on each disk) allocate parity (P(x,y)) on different disks. The layout of data and parity shown in this figure is called a "left-symmetric" placement [Lee90].*

the various ways parity blocks are placed. They found that the rotated parity organization in Figure 3.3, which they call *left-symmetric* [Lee90], is one of the best alternatives. Although there is little difference between alternatives at most request sizes and loads, the response time for requests large enough to access one block from every disk can vary by as much as 30% with different parity placements when the array is initially idle.

3.3.1.2.3. Non-Striped

Gray, Horst, and Walker [Gray90] explored the use of redundant disk arrays with N+1 parity in systems featuring traditional database applications. Although their customers were eager to switch from mirroring to N+1 parity to reduce the cost of higher data reliability, they were not eager to have their data automatically striped. Their customers' small and randomly located requests did not need the high transfer bandwidth achievable by striping large files. Furthermore, the automatic load balancing also provided by striping was unnecessary because their database systems already provided tools to achieve evenly loaded disks.

In the absence of a good reason to stripe data automatically, Gray et al. proposed an N+1-parity code that did not stripe. Shown in Figure 3.4, they called this scheme *parity striping;*

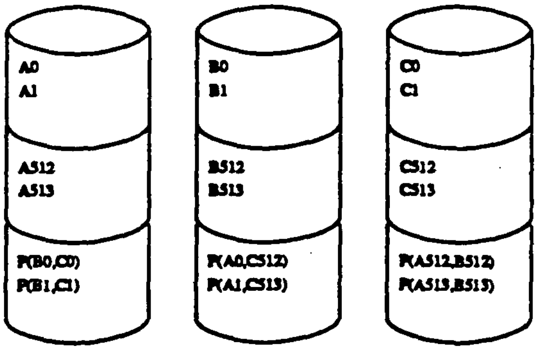


Figure 3.4: Non-Striped Disks with Distributed Parity. *Because some conventional database users do not benefit from striping, a non-interleaved N+1-parity scheme called "parity striping" has been proposed [Gray90]. In this scheme, host file systems see each disk as a separate device with (N-1)/N times as much data as the underlying disk can store. In this example, each of three disks stores separate data: A, B, and C. Parity, P(x,y), is still spread evenly across disks, but it is clustered on each disk in one large block.*

data is not interleaved, although parity remains spread over all disks so that parity updating work can be evenly distributed. Their parity striping scheme also allocates all of the parity in one large block on each disk. This simplifies the integration of N+1-parity disk arrays into existing disk management software because all of the location specific code is unaffected. For example, code that translates a file read into a list of disk blocks to be read does not need to know about interspersed parity blocks.

Although the block-interleaved striping organization shown in Figure 3.3 shows parity blocks interspersed among data blocks, this is not necessary. Lee and Katz [Lee91] found this organization for parity to be one of the best alternatives, but they also found an organization analogous to parity striping (where parity is allocated in one extent on each disk) to be another of the best alternatives. This latter organization is called *flat-left-symmetric* by Lee and Katz. Relative to the left-symmetric parity placement shown in Figure 3.3, flat-left-symmetric parity placement leads to better performance for large, sequential reads because interspersed parity blocks do not have to be skipped over, and it leads to poorer performance for large, sequential writes because it causes extra, long seek between writing information data and writing parity data.

In Section 3.4.1, I examine the advantage to performance provided by Gray's organization for workloads with large numbers of small requests evenly distributed across all disks.

3.3.2. Multiple-Erasure-Correcting Encodings

Figure 3.5 describes an estimate for the mean time to data loss (MTTDL) in single-erasure-correcting I/O systems suffering catastrophic disk failures [Patterson88]. (Chapter 5 explores estimates for data reliability in more detail.) As the number of disks soars, reliability plummets; even with single-erasure correction, an I/O system of more than one thousand disks with daily repair is less reliable than a single disk!

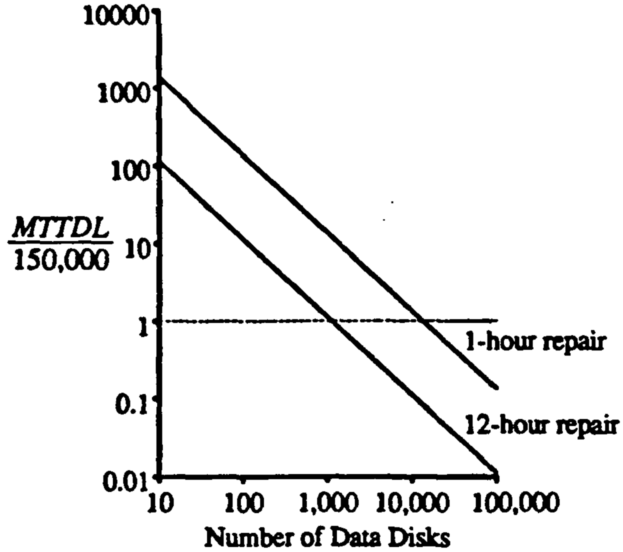


Figure 3.5: Larger Redundant Disk Arrays Have Lower Reliability. *The most optimistic estimate for the mean time to data loss in a single-erasure-correcting array is $MTTDL = (MTTF_{disk})^2 / (\#Disks \times N \times MTTR)$, where $MTTF_{disk}$ is the mean lifetime of an individual disk, #Disks is the total number of disks, N is the number of information (user data) disks associated with each redundant data disk, and MTTR is the mean time required to repair and recover a failed disk. Chapter 5 analyses this and more complex models for array reliability. In the figure above, MTTDL is shown in terms of the number of individual disk's mean lifetimes, each 150,000 hours, that are expected to pass before the array suffers an unrecoverable failure. The array shown has 10% as many redundant data disks as information disks; that is, it has $N+1 = 11$. Two values for mean repair and recovery time are shown. One hour repair requires on-line "hot spare" disks or continuous human maintenance. A simpler scheme in which repair is carried out during daily visits by maintenance personnel, has a mean time to repair of 12 hours.*

The problem for a single-erasure-correcting code with increasing disk parallelism is an increased frequency of multiple failures within a small span of time. Data will be lost the first time a second disk fails in a recovery group that has not finished the recovery of an earlier failed disk.[3] If the frequency of failures rises, then the probability of closely-spaced failures also rises, and loss of data becomes much more likely. For this reason, and for those applications that have more stringent requirements for data reliability, redundancy encodings that protect against

[3] If a sector of data on an otherwise functional disk cannot be read during the recovery of a failed disk, then two sectors of data are lost: the sector that cannot be read and its corresponding sector on the disk being recovered. Although this destroys a relatively small amount, and will happen much less frequently in a redundant disk array than in a single disk, it is considerably more likely than the loss of an entire disk's data because the inherent rate at which disks fail to read a sector's data is quite high in a large, busy disk array.

multiple coincidental failures may be useful. The next two sections discuss codes that correct all double-disk failures.

3.3.2.1. Binary Symbol Codes

An array is much like a memory chip; a codeword is the set of bits at the same position in each chip or disk. Bearing in mind this analogy to a memory system, it can be seen that the single-erasure-correcting codes of the last section correspond to the addition of a single bit of parity on each memory codeword. A lost disk's data can be recovered with parity protection because, unlike memory chips, the failures covered by this encoding are all self-identifying, so that the lost bits' values are whatever is necessary to make the parity of the data bits equal to the values in the parity bit.

Codes that compute check bits as the parity of subsets of data bits, including the single-erasure-correcting codes of the previous section, are called *binary linear* codes. The present section presents binary linear codes for double-erasure correction that minimize parity update penalties while either maximizing data reliability or minimizing check-disk overhead. Non-binary linear codes for double-erasure correction are discussed in Section 3.3.2.2.

3.3.2.1.1. Examples: Hamming and Two-Dimensional Parity Codes

By using the analogy to a memory system once again, it becomes clear that an obvious candidate code is a *extended binary Hamming code* [MacWilliams77 pp 27]. In a main memory system, a binary Hamming code with c check bits corrects any single error in 2^c-c-1 information bits. The binary Hamming code can be *extended* by one more check bit to correct all single errors and detect all double errors. A typical memory system protected by an extended Hamming code might use 7 check bits with 32 data bits and provide correction for single, non-self-identifying errors and detection for double, non-self-identifying errors. Although such a code can and has been directly used as a single-disk failure-correction encoding [TMC87], it ignores information identifying failed disks. In fact, the Hamming code that detects any two errors can be used to correct any two erasures. Any code that detects all t non-self-identifying failures can also be used to correct all t self-identifying failures [Gibson89b, Peterson72 pp 305]. This means that the unextended Hamming code is double-erasure-correcting and the extended Hamming code is triple-erasure-correcting. Figure 3.6 shows a Hamming code with four check disks that protects 11 data disks with double-erasure correction.

To correct four or more simultaneous erasures, Hamming codes can be generalized to Bose-Chaudhuri-Hocquenghem (BCH) codes [Peterson72 pp 80]. Although this provides greater correction capability and higher cost than I seek, a subclass of these codes, the Reed-Solomon (RS) codes, are useful non-binary codes, which are discussed in Section 3.3.2.2.

Hamming codes are not the only ones that provide double-erasure correction. Another straightforward approach called *two-dimensional parity* [Gibson89b] uses two orthogonal, N+1-parity codes (as shown in Figure 3.6) to ensure that any two failures can be decomposed

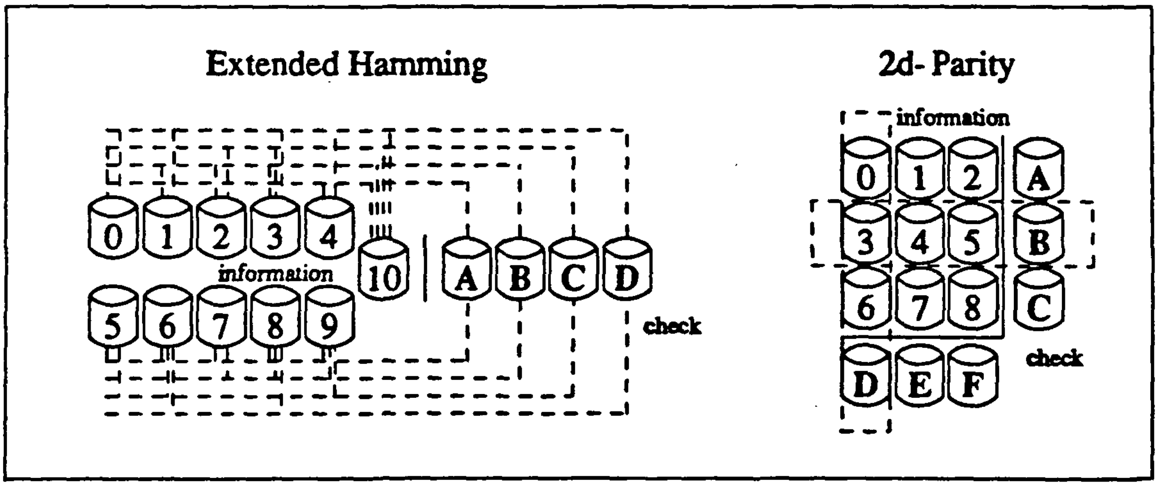


Figure 3.6: Double-Erasure-Correcting Code Examples. *With self-identifying disk failures, or erasures, the common, single-error-correcting Hamming code can be used as a double-erasure-correcting code. On the left in this example is a Hamming code that doubly protects 11 data disks with four check disks. The relationship between data and check disks is shown with dashed lines; disk 3 is part of the parity calculation for check disks B, C, and D. An alternative double-erasure-correcting code is achieved by extending a single-erasure-correcting parity code to two dimensions. By including every data disk into two parity groups that overlap on only that data disk, any two disk failures can be recovered as two separate, single erasures in different groups. For example, on the right, a two-dimensional parity encoding protects nine data disks with six check disks. In this case, disk 3 is part of the parity calculation for check disks B and D.*

into two separate single failures. The two-dimensional parity code has also been called a *cross-parity* code [Rao89 pp 65] and is a special case of a *product code* [Peterson72 pp 131]. Binary, d-dimensional product codes compose d component codes so that the codewords of each dimension intersect codewords of other dimensions at exactly one bit.

Figure 3.6 also shows a two-dimensional parity code with six check disks that corrects any two failures in nine information disks. This code has a higher check-disk overhead than does the Hamming code example beside it: 6/9 versus 4/11. If one byte on one data disk is changed, however, a two-dimensional parity code will update exactly two check disks to maintain the code, whereas the example Hamming code will update at least two and as many as four check disks (with an average update penalty of 2.5). Hamming codes with more disks have even higher update penalties. For redundant disk arrays whose requests always access every disk, such as those that use byte-interleaved striping, the update penalty is not an important metric. But for redundant disk arrays that support small, random accesses, the update penalty indicates the number of disks not available for other requests during a single disk update. By this metric, a two-dimensional parity code performs better than does a Hamming code.

3.3.2.1.2. Optimal, Double-Erasure-Correcting Binary Codes

Gibson, Hellerstein, Karp, Katz, and Patterson [Gibson89b] examined double- and triple-erasure-correcting binary codes to find those that minimize the update penalty associated with the change of a single data byte. They demonstrated that any code that guarantees correction for all double erasures must update at least two check disks with each data-disk update. Because there are codes that guarantee correction for all double erasures with exactly two check-disk updates per data-disk update, two-dimensional parity codes for example, their search was restricted to these minimal update-penalty codes.

Within the class of codes that minimize update penalties to two, they found codes that achieve minimal check-disk overhead by maximizing the number of data disks protected by a given number of check disks. For double erasures, these codes are called *full-2* codes. For a given number of check disks, c, a full-2 code has one information disk for every different combination of two check disks. The check-disk overhead for these codes is 2/(c-1).

Although full-2 codes are optimal with respect to update penalties and check-disk overhead, they do not ensure the best data reliability. Because all double erasures are correctable in a double-erasure-correcting code, maximum data reliability depends on the fraction of three or more erasures that are correctable [Peterson72 pp 100]. Gibson et al. sought minimal update-penalty codes that maximized the fraction of correctable triple erasures because this is the most common number of erasures not guaranteed to be correctable. There is at least one set of uncorrectable triple erasures for each data disk because an update to one data disk updates a particular set of two-check disks; the definitely uncorrectable set of three erasures formed by a data disk and the two check disks updated together were called *bad three-erasures*.

Among two-erasure-correcting codes, the two-dimensional parity code corrects all three erasures except for bad three-erasures. Moreover, there is no other two-erasure-correcting code that has a lower check-disk overhead, corrects as many three-erasures, and has the same number of check disks.

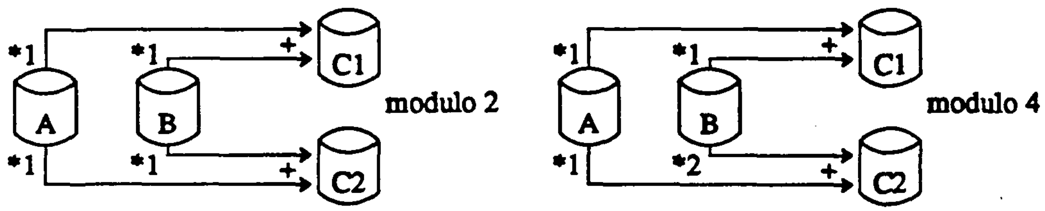
Our other metric for selecting a code was the recovery group size. This metric is the number of disks involved in recovering a single failed disk. For systems with on-line applications, the recovery group size indicates the fraction of the array whose performance is degraded during the repair and recovery of a single failure. There is a basic tradeoff between three of these codes' metrics, which can be described as:

$$\begin{aligned} &\textit{check disk overhead} \times (\textit{average recovery group size} - 1) \\ &\qquad\qquad = \textit{number of erasures guaranteed correctable.} \end{aligned} \qquad (3.1)$$

This implies that across all double-erasure-correcting codes with minimum update penalties, codes that have lower check-disk overhead require more disks to recover a single failure. Conveniently, higher check-disk overhead can be traded for lower recovery group sizes by using more check disks and subsetting the associated longer code so that all disks have nearly the same recovery group size. This way the maximum recovery group size is only slightly larger than the average group size, and the average itself is lower [Gibson89b].

3.3.2.2. Non-Binary Symbol Codes

A code whose codewords include a multiple-bit *symbol* from each disk is called a *non-binary* code. Non-binary codes can achieve much lower check-disk overhead than binary codes. They do this by encoding more information into the same amount of check data. The example in Figure 3.7 demonstrates this advantage using a non-binary code based on two-bit symbols.



Binary (b=1)

C1 = C2 = (A+B) mod 2

A = f(C1,C2) ?

B = g(C1,C2) ?

A, B, C1, C2: 1 bit symbols

Nonbinary (b=2)

C1 = (A+B) mod 4

C2 = (A+2B) mod 4

↓

A = (2C1-C2) mod 4

B = (C2-C1) mod 4

A, B, C1, C2: 2 bit symbols

Figure 3.7: Contrasting Binary and Non-Binary Codes by Example. *In a binary code that corrects all double erasures, if two data disks (A and B) share the same pair of check disks (C1 and C2) then the failure of the two data disks leaves two copies of their parity accessible but it leaves no way to extract the failed disks' separate values (one equation in two unknowns). However, in a non-binary code that corrects all double erasures, two data disks can share the same pair of check disks. In this example, each symbol on disk B is doubled (modulo 4) before it is added (also modulo 4) to the corresponding symbol from disk A and stored onto check disk C2. In this case, the loss of both data disks is correctable because the contents of the two check disks differ significantly (two independent equations in two unknowns). In fact, for each pair of check disks, a non-binary code can support as many information disks as there are distinct pairs of independent equations in two unknowns in the field of integers modulo 2^b, where each symbol contains b bits.*

Where a binary code cannot recover the contents of two data disks that share the same pair of check disks, a non-binary code can weight the contribution of the two data disks differently for each check disk; thereby, storing on the check disks two independent equations for the data disks' values. If the two data disks are concurrently failed, the lost disks' values in this example are recovered using a little bit of linear algebra in the field of integers modulo four.

Let me explain this little bit of algebra more explicitly. All linear codes compute check symbols as a linear combination of data symbols over a particular field. For the binary codes of Section 3.3.2.1, this field is the integers modulo 2; that is, check bits are computed as the exclusive-or of subsets of data bits. In a non-binary code, this exclusive-or simplification is lost. If symbols are selected to be b bits long and operations done in the field of the integers modulo 2^b, then check symbols are computed as the sum of a multiple of each data symbol modulo 2^b. For example, Figure 3.7 computes each symbol of disk C2 as the modulo-four sum of two times the value of the corresponding symbol on disk B and the value of the corresponding symbol on disk A. If in a particular codeword, disk A's symbol has value 1 and disk B's symbol has value 3, then check disk C2's value will be $3=1+2\times3$ (mod 4), and check disk C1's value will be $0=1+3$ (mod 4). If the two data disks, A and B, fail simultaneously, then their values for the above symbols can be recovered according to Figure 3.7 as $1=2\times0-3$ (mod 4), and $3=3-0$ (mod 4), respectively. Although computation in the integers modulo 2^b is not difficult to design, it is much more expensive than parity computation, and its cost grows proportionately with b.

Nonbinary codes can achieve much lower check-disk overhead than binary codes. For example, Table 3.4 shows the number of data disks that can be protected from all double erasures by the non-binary Hamming code over the integers modulo 2^b with c check disks. This code provides a powerful way of lowering check-disk overhead: increase the size of the per-disk symbols in each codeword. In particular, by enlarging the symbol size, any number of information disks can be protected from all double erasures with only two check disks!

Check	Number of data disks protected by double-erasure correction					
Disks	Computation in field of integers modulo 2^b					
c	b=1	b=2	b=4	b=8	b=16	b=32
2	1	3	15	255	65535	4×10^9
3	4	18	270	65790	4×10^9	2×10^{19}
4	11	81	4365	2×10^7	3×10^{14}	8×10^{28}
5	26	336	69900	4×10^9	2×10^{19}	3×10^{38}
6	57	1359	1×10^6	1×10^{12}	1×10^{24}	1×10^{48}
7	120	5454	2×10^7	3×10^{14}	8×10^{28}	6×10^{57}

Table 3.4: Check-Disk Overhead Limits for a Non-Binary Hamming Code. *Using a non-binary Hamming code with computations in the field of integers modulo 2^b and c check disks, $(2^{cb}-1)/(2^b-1)-c$ data disks can be protected from all double erasures [Peterson72 pp 120].*

A different code that has nearly as low check-disk overhead for double-erasure correction and extends to correct more than two erasures with comparably low check-disk overhead is the Reed-Solomon code[4] [MacWilliams77 pp 294]. Reed-Solomon codes have been used for correcting burst and random errors in magnetic and optical disks [Deodhar83, Glover88, Massiglia86]. Because of their wide-spread use, highly-integrated encoding and decoding chip sets for Reed-Solomon codes have been implemented [Rao89 pp 114, Shao88]. The availability of these compact, encoder/decoder chip sets and the wide range of industrial experience with Reed-Solomon codes has led to the use of a variation of the two-check-symbol, Reed-Solomon code in a disk array product under the name of *P+Q parity* [ATC90]. This implementation uses b=4 bit symbols to provide double-erasure correction for up to 13 data disks.

One drawback to attaining low check-disk overhead through non-binary codes with only two check symbols is a large recovery group size; any single-disk failure will involve all disks in its recovery. Reducing recovery group size by increasing the number of check disks introduces another problem; with more than two check symbols, these codes will generally not have

[4] The number of data disks that can be protected from c-erasures in a Reed-Solomon code with b-bit symbols and only c check disks is 2^b-1-c. For c=2, this is only two disks fewer than the non-binary Hamming code.

the minimal update penalty of two. However, there is a non-binary analogue to the binary full-2 code. This non-binary full-2 code protects up to $c(c-1)(2^b-1)/2$ data symbols against double erasures with c check symbols. Like the binary full-2 code, the non-binary full-2 code protects the largest number of data symbols of any double-erasure-correcting code with minimal update penalties and with the same values for c and b. The non-binary full-2 code also abides by the relationship binding recovery group size to check-disk overhead given in the last section. Using this relationship, the average recovery group size can be adjusted by changing the number of check disks, c, where

$$\textit{average recovery group size} = 1 + 2\,k/c \qquad (3.2)$$

where k is the number of data disks protected.

Another drawback to attaining low check-disk overhead through non-binary codes is that there are new sets of uncorrectable three-erasures other than a data disk and its two check disks. The opportunity for more than one data disk to share the same pair of check disks gives rise to the increased number of data disks protected. However, it also introduces these new sets of three-erasures that are not correctable. Figure 3.7 illustrates the problem because the loss of disks A, B, and either C1 or C2 is an uncorrectable three-erasure that is not a bad three-erasure (a data disk and its two check disks). It turns out that any double-erasure-correcting non-binary code with minimal update penalties cannot achieve a check-disk overhead that is lower than the appropriate binary code unless it fails to correct all three-erasures (except bad-three erasures). Further research into the tradeoff between check-disk overhead and reliability for non-binary codes with minimal update penalties is needed.

Section 3.2 describes how a disk's internal, error-correcting codes manage the most frequent and non-catastrophic disk failures. Disk-array encodings that assume the presence of these internal codes permit catastrophic failures to be identified. Kim and Patel [Kim85, Kim86] proposed to eliminate these internal, error-correcting codes in the disks of an array with a non-binary encoding across disks. Their code is derived from one designed for bubble memories [Patel82]. It uses eight-bit symbols, a single check disk, and a multi-codeword, overall parity byte embedded in the data to protect up to 31 data disks against single, random-bit errors in each codeword or a single, non-self-identifying disk failure. This code provides the same failure-correction capability as N+1 parity, and it improves on N+1 parity by recovering the space on each disk used to store its internal redundancy. However, overall it is significantly less powerful than N+1 parity because high-density disks frequently suffer losses of multiple bits, called bursts, which a disk's internal redundancy is specifically designed to correct.

3.3.3. Encoding Summary

Because the inclusion of redundant data in a disk array can be viewed as a coding problem, the selection of a best organization of user data and redundancy can be approached as a code search. For performance reasons, candidate codes should be linear, and if high-access bandwidth is required, codes should not update more check disks with each data-disk update than they guarantee correctable.

When repairs are short or arrays are not large, single-erasure-correcting codes should provide sufficient data reliability. Mirroring, the traditional mechanism for single-erasure correction in disk subsystems, has high overhead costs that can be reduced with N+1-parity codes. The characteristics of these N+1-parity codes depend on the organization of user data in the array. Although some self-tuning database applications prefer not to automatically stripe data, most disk arrays rely on striping to improve performance by balancing the load across disks and enabling the parallel transfer of large requests. Byte-interleaved striping provides increased transfer bandwidth without increasing access bandwidth in a manner analogous to, but more flexibly than, the way that parallel transfer disks increase transfer but not access bandwidth. In contrast, block-interleaved striping provides both high-transfer and high-access bandwidth at the cost of greater software complexity.

For large arrays or where high data reliability is required, codes that guarantee correction of simultaneous double failures may be employed. Using simple, binary-parity computations, codes with minimal check-disk overhead are available. These codes have been called full-2 codes. For higher data reliability, two-dimensional parity codes, ones that maximize the fraction of correctable triple failures, do not have much higher check disk overhead. Both of these codes can be pared down to a subset to trade higher check-disk overhead so that a smaller fraction of the array is degraded during the repair and recovery made necessary by a failure.

Even lower check-disk overhead can be achieved with double-erasure correction based on non-binary codes. By expanding the encoding and decoding hardware, the number of check disks can be kept to two for any number of data disks. With these codes, however, large fractions of data are degraded during repair and recovery caused by a failure, and a greater variety of triple failures are uncorrectable.

I will examine the reliability of N+1-parity-encoded disk arrays in more detail in Chapter 5.

3.4. Performance of Redundant Disk Arrays

Disk arrays derive their performance advantages from disk parallelism. Parallelism and the performance advantages it brings are further increased when large diameter disks are replaced by arrays composed of physically small disks with small capacities. Increases in parallelism improve the performance of two very different types of workloads: large numbers of small, random accesses and individually large, sequential accesses. The former are measured by an array's *access throughput* in units of I/Os per second. The latter are measured by an array's *transfer throughput* in units of megabytes per second, although the latter are sometimes also measured by the response time of each access.

By increasing the number of disks in an array, the peak number of small random I/Os per second is increased accordingly. This is basic benefit to access throughput; more disks can sustain a workload with more I/O requests per second. Alternatively, without a heavier workload, increased numbers of disks mean each disk will have less work to do, and this will result in shorter queues, fewer queueing delays, and lower response times.

The successful extrapolation of these benefits depends on the even distribution of accesses across disks. Requests are more likely to be evenly distributed when data is striped across small disks because large files are spread over many disks, and small files are allocated to disks without regard to logical relationships (such as directory membership, for example). An even distribution is the secondary benefit of disk parallelism for workloads that emphasize many small, random accesses per second.

Increasing the number of disks also increases the transfer bandwidth for individually large accesses if data is striped over many disks. For example, if requested data occupies multiple tracks on each disk, then response time will be decreased nearly linearly in the number of disks. This decreased response time has been the strongest motivation spurring the development of disk array products. The next section examines effects of data striping on non-redundant disk array performance.

The benefits of data striping in a redundant disk array are reduced by the maintenance of redundancy. Because redundant data must not be located on the disks whose data it protects, changes in user data must cause associated changes in redundant data on another disk. The extra work generated by each data update consumes access throughput and, to a lesser degree, transfer throughput. This is the primary penalty to the performance of redundant disk arrays. The second subsection of this section discusses the performance lost to the maintenance of redundant data and techniques to minimize this loss.

3.4.1. Benefits of Data Striping

Section 3.3.1.2 above introduces the differences between byte-interleaved striping, block-interleaved striping, and non-striping disk arrays with emphasis on their interaction with an N+1-parity encoding for redundant data. In the subsections of this section, I expand on my earlier discussion of the performance implications of different types of data striping. My purpose here is to provide a broad understanding of the effects on performance that result from the interaction between striping parameters, workload characteristics, and effective disk optimizations without consideration for redundancy. The effects of redundancy on performance are the subject of Section 3.4.2.

In the following subsections I discuss how data striping is most successful at decreasing the response time of large accesses by transferring 1/Nth of a request from each of N disks in parallel, and also how it increases the rate at which small, random accesses can be serviced without exceeding an average response time goal by automatically load balancing these accesses over all disks. Then I explain how data striping invokes more disk-head-positioning operations than a non-striping organization for most workloads, and how these extra positioning operations can limit or eliminate both of the aforementioned benefits of striping. After reviewing the more effective optimizations for reducing the penalties of these extra positioning operations, I demonstrate how the unit of interleaving in a striped disk array can interact with the array's workload and have a negative effect on performance. Finally, I report the results of a study that shows how to select the unit of interleaving, or striping unit, to minimize these negative effects on performance.

3.4.1.1. Striping for Parallel Transfer

The major benefit of data striping is a reduced transfer time for large accesses [Johnson84, Kim87a, Salem86, Reddy89]. By spreading the data of a single large access over N disks, the time required to transfer this data in parallel is reduced by a factor of N. If the response time of a access to a single disk is dominated by data transfer, then striping this data over multiple disks dramatically decreases response time. This reduced response time benefit is delivered by both byte-interleaved and block-interleaved striping.

The advantages of striping for large accesses are well-demonstrated. For example, the swap files in a Cray supercomputer are large and transferred contiguously; striping these files over multiple disks improves the performance of programs with large memory requirements substantially [Johnson84]. In fact, scientific computation, which includes most of the Cray programs that benefit from striping swap files, is the most common example of an application that benefits from striping. In separate studies, Kim, Nigam, Paul, and Flynn [Kim87a] and Reddy and Banerjee [Reddy89] have demonstrated the effectiveness of data striping for scientific computations that manipulates large data files.

Striping to increase transfer rates is successful in applications other than scientific computation in proportion to the frequency with which these applications operate on large files. Obvious candidate applications are database systems [Boral83, Livny87, Salem86], and image processing and visualization [Bailey91], but recent, innovative research into general-purpose file systems promises to combine many small write accesses into single large write accesses that will be able to exploit the parallel access provided by striping [Rosenblum90].

3.4.1.2. Striping for Disk Load Balancing

Block-interleaved striping can also benefit workloads that feature small, random accesses because a small access requires service from only one disk, so N disks can perform N different, small accesses in parallel. In this case, the benefit of striping is that it provides automatic balancing of the workload over all disks.

Disk-load balancing is important because most non-striped collections of disks suffer substantial imbalance between the amount of work required of the most and least busy disks. This problem is referred to as the "skewing" of disk accesses [Friedman83, Kim87b, Livny87, McNutt86]. In IBM systems disk skewing has been well-documented; a typical skew might be 70% of the accesses access 30% of the disks [Kim87b]. McNutt models skewing data captured from large disk farms, and, for example, predicts that a 64 disk system sustaining an average of 8 accesses per second per disk will have at least one disk that is the target of an average of more than 40 accesses per second, and that over 80% of the disks will be required to deliver less than the average of 8 accesses per second. With this kind of skewed access, the maximum rate at which an I/O system can perform small, random accesses with reasonable response times is determined by the disks with the heaviest load. Therefore, non-striped disk farms achieve far lower access throughputs than is possible if their workload were balanced over their component disks.

Block-interleaved striping balances a workload over its disks by spreading the contents of large files over multiple disks, and by allocating small files to different disks in a uniform fashion. As long as the striping unit is small relative to the size of files that cannot be effectively cached, the data that is accessed most frequently should be uniformly distributed across disks.[5]

[5] It is possible that striping data will not lead to uniform access, but most counter examples are quite pathological. For example, if every Nth block in a particular file is much more likely to be accessed than the rest of this file's blocks AND the file is striped over N disks, then the disk load will not be uniform [Chen90b].

Uniform distribution is not as good as a completely even load on all disks, but it is the best that can be expected in absence of specific knowledge of the workload's access patterns. Where such specific workload knowledge is available, such as in special-purpose database applications, striping might actually lower access throughput [Gray90].

3.4.1.3. Extra Positioning Operations

Striping, however, is not without drawbacks. Whenever it uses more than one disk to perform an access that would otherwise have been performed by a single disk, it increases the amount of positioning overhead associated with that access. This overhead can adversely affect both of the above described benefits of striping [Chen90b, Gray90, Kim87b, Reddy89, Salem86]. This section demonstrates that this problem can be severe, and the following sections discuss methods to limit its effects.

The reduction in response time that striping provides for large accesses can be diminished by an uneven distribution of the positioning times of each disk involved in the parallel access. This uneven distribution arises if each disk in a parallel transfer has a different distance to move its read-write head and a different angular offset from the data once its head is at the correct track. Bitton and Gray have shown that if N disks, whose heads are located at randomly distributed cylinders, all seek to a specific cylinder, the maximum seek distance, as a fraction of the total number of cylinders, is approximately [Bitton88]:

$$1-\frac{2}{3}\frac{4}{5}\cdots\frac{2N}{2N+1}\,. \tag{3.3}$$

This means that where the average fraction of the total cylinders that a single disk must travel is 1/3, this fraction for the longest seek among 10 disks is nearly 3/4. For the disks modeled by Gray [Gray90], this maximum of 10 seeks takes 25 msec to perform, larger than the single disk average seek time by 8.3 msec. Even if all N disks are already at the correct cylinder, Section 3.3.1.2.1 shows if their spindle motors are independent, the head that spends the longest time positioning, searches through an average of N/(N+1) tracks. Relative to an average search time for a single disk of 8.3 msec, again using Gray's disks, the average time for the longest of 10 searches is 15.2 ms – longer by 6.9 msec. With the last of N disks completing its overall positioning later than would a single disk by times on the order of 10 msec or more, the amount of data to be transferred will have to be quite large if the overall response time is to be reduced nearly as much as the data transfer time is reduced by data striping. For example, with 10 of Gray's disks, the amount of data that an access must request in order for parallel transfer to achieve a speedup of at least 9 over a single disk, if the overall positioning time of a parallel transfer is 10 msec larger than that of a single disk, is 4,600 KB![6]

The effects of increasing the number of positioning operations also adversely affects the access throughput of striped disks. Because positioning is a disk operation that does no useful work (it does not transfer data), the more disk seconds that are spent positioning, the fewer that are spent on doing the requested work. If the positioning operation on each disk involved in a parallel access over N disks takes the same amount of time as the positioning operation of an equivalent non-parallel request, then the total amount of non-useful disk busy time is increased by a factor of N. This loss of useful disk operating time means that the maximum throughput of a striped array will be smaller than the maximum throughput of a non-striped array (that has a completely even distribution of its workload) [Gray90]. Disk arrays with block-interleaved striping can ameliorate this decreased access throughput by choosing a striping unit large enough that most of the smaller accesses are serviced by a single disk; Sections 3.4.1.4 and 3.4.1.5 demonstrate how this is achieved.

[6] This estimate is based on solving for the *Size* of an access such that parallel transfer yields a *Speedup* $\geq 90\%$ of N, where $N = 10$ disks, and

$$Speedup = \frac{\text{single disk response time}}{N \text{ disk response time}} = \frac{16.7 + Size/2.0}{16.7 + 10 + Size/2.0/N} \geq 0.9N \text{ .}$$

3.4.1.3.1. Synchronizing Disk Spindles

The most direct way to reduce the effect on response time of the longest of multiple positioning operations is to synchronize the operation of all disks. Fully synchronized disk arrays have the read-write heads of all disks at the same track with the same offset in that track at all times. This avoids penalties to response time by insuring that parallel positioning takes the same amount of time on all disks, but it does not overcome the loss of access throughput that arises from the increased number of positioning operations of striped disks relative to non-striped disks.

A byte-interleaved striping disk array with synchronized spindle motors is an example of a fully synchronized disk array. It appears to its host as though it were a parallel transfer disk [Bucher80, Fujitsu2361, Kryder89]. This configuration is discussed in Section 3.3.1.2 and, more thoroughly, in Kim's work on this subject [Kim86].

Although synchronizing the spindle motors of multiple disks is not difficult [Sierra90 pp 208], in many disk systems it is not possible to synchronize actuator motion. Instead, read-write heads may be logically synchronized by issuing the same seek command to each independently actuated disk. A block-interleaved striping disk array with synchronized spindles will operate this way if its requests are large enough to call upon all disks. This configuration can be beneficial for the performance of large accesses in a disk array with low load (as Figure 3.8 below shows), but at high loads with varying access sizes, the logical synchronization of heads over tracks is lost unless disks are (unprofitably) inhibited from operating independently when requests do not require all disks to perform identical operations. In this latter case, because each disk may have a different operations to perform, even if some of those operations correspond to a parallel transfer on all disks, it is unlikely that all disks initiate and complete seeks at the same time, so spindle synchronization may not be exploited.

Although a definitive treatment of the benefits of spindle synchronization when actuators are not synchronized is not known to me, it appears that opportunities to exploit its benefits are few except in specialized workloads.

3.4.1.3.2. Shortening Positioning Distances

If the techniques for reducing positioning times described in Section 2.1.4 are applied to striped disk arrays, the degradation on response time and access throughput caused by data striping can be reduced. Zero latency reads and anticipatory seeks are the most appropriate of these techniques for the issues. With zero latency reads and striping units near the size of a track, synchronized spindles are not needed to minimize rotational delays; once the head is at the correct track, it can immediately begin to transfer data [Salem86]. Section 3.4.1.5 shows that a good choice for a striping unit is the size of a track. Anticipatory seeks that return all disks heads to the center of their respective surfaces helps in two ways: they may reduce the average seek distance, and they insure that all disks experience the same seek distance on a parallel transfer.

3.4.1.4. Interaction Between Striping Unit and Workload

A paper by Gray, Horst, and Walker argued that data striping adversely affects the performance of an array of disks in an on-line transaction processing system [Gray90]. Their paper is nominally about redundant disk array organizations, and is discussed further in Section 3.4.2. However, its analysis pertains mainly to striping performance, and I review it here because it demonstrates the tradeoff between striping unit size and performance under different workloads. In the next section I present work by Chen and Patterson that shows how to select a "best" striping unit.

Because the applications examined by Gray et al. employed disk load-balancing software, they believed that maintaining a balanced load is better done explicitly on non-striped disks than automatically by a striped disk array. If a non-striped set of disks has a balanced load, then a striped array will get lower throughput at most request sizes because of the additional seeks it must perform. Since their application concerns largely small requests with high concurrency,

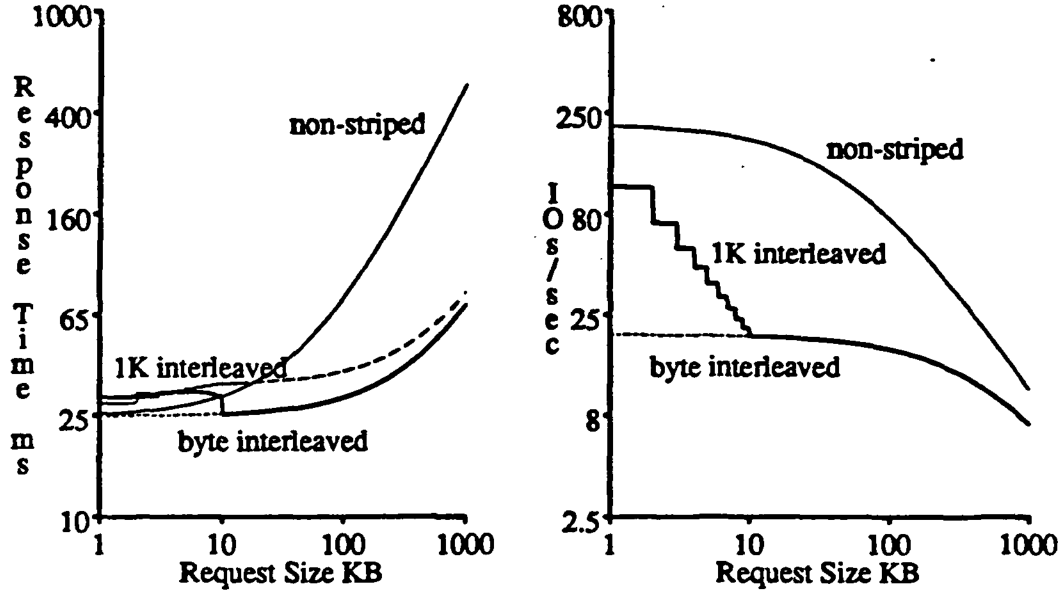


Figure 3.8: Disk Array Performance with 1 KB Striping Unit. *Gray's model for response time in an idle disk array with 11 disks is shown on the left [Gray90]. The non-striped, or "parity striped" response time, shown by the thinner solid line, assumes that all data for each request is located on a single disk. As long as there is only one request at a time, an array with byte-interleaved striping, shown by the dotted line, has the lowest response time. The other two lines both model disk arrays with synchronized spindles and block-interleaved striping. If all requests are the same size, shown by the thick solid line, then large accesses will not suffer positioning time delays and response time will match the array with byte-interleaved striping. However, as shown by the dashed line, if most requests access only 1 KB, then the infrequent larger accesses suffer the penalties associated with positioning widely distributed read/write heads. On the right, the maximum number of accesses per second for the same array organizations is shown assuming that each disk is 50% utilized. In this case, an array with byte-interleaved striping spends more disks' efforts positioning heads, so its throughput is lower than a non-striped array with evenly balanced disk load. There is only one line for arrays with block-interleaved striping on the right because this model assumes all requests access the same amount of data. See Figure 2.3 for this evaluation applied with a 32 KB block size.*

they believed that throughput, measured in accesses per second at 50% disk utilization, is an important metric.

Gray et al. presented a simple model for response time when all disks are initially idle and for throughput at 50% disk utilization[7]. They used this model to compare 11 non-striped disks with a balanced load to an 11-disk array with block-interleaved striping and synchronized spindles. Their disks were Tandem's XL80s which have an average seek time of about 16.7 ms, a

[7] Their model is for redundant arrays, but because redundancy has no effect in the read request case, the model evaluates non-redundant striping performance as well.

rotation time of 16.7 ms, and a data transfer rate of 2 MB/s. They deliberately used a small striping unit (1 KB) to demonstrate that fine-granularity striping is a bad tradeoff for on-line, transaction processing systems.

Figure 3.8 includes some of their results and adds a synchronized disk array with byte-interleaved striping. It shows the response time for a request of a particular size when the disks are initially idle and the accesses per second when the disks are 50% utilized. I have presented two response-time curves for block-interleaved striping with synchronized spindles; the first, following Gray et al., assumes that before every large request, all disk read/write heads are at random locations. This assumption is valid if nearly all requests are single-block accesses so that disk activity immediately before a large request has been independent on each disk. However, if every request has the same size and requires multiple disks, read/write heads of many disks will be left by each request at the same location on each disk. The second, block-interleaved striping response-time curve incorporates the expected positioning time based on the average number of seeks of different distances that would be expected. Figure 3.8 shows that for small requests 1KB, block-interleaved striping is slower than non-striping, and for all sizes it achieves substantially less throughput at 50% utilization of the disks.

Figure 3.9 applies the model of Gray et al. to block interleaving with a striping unit of 32 KB (about one track). In this case, non-striped disks do not display a better idle-array response time than that of block-interleaved striping at any request size, and the 50% utilization throughputs are much closer together. Because evenly-balanced, non-striped disks are rarely the case [Friedman83, Kim87b, Livny87, McNutt86], there seems to be little performance advantage for non-striped disks over that of block-interleaved striping with synchronized spindles and a striping unit of 32 KB.

Figures 3.8 and 3.9 also demonstrate the importance of selecting the correct striping unit for a particular workload. If that workload has many small, random accesses, a large striping unit seems best, and if that workload has fewer, large accesses, a small striping units seems

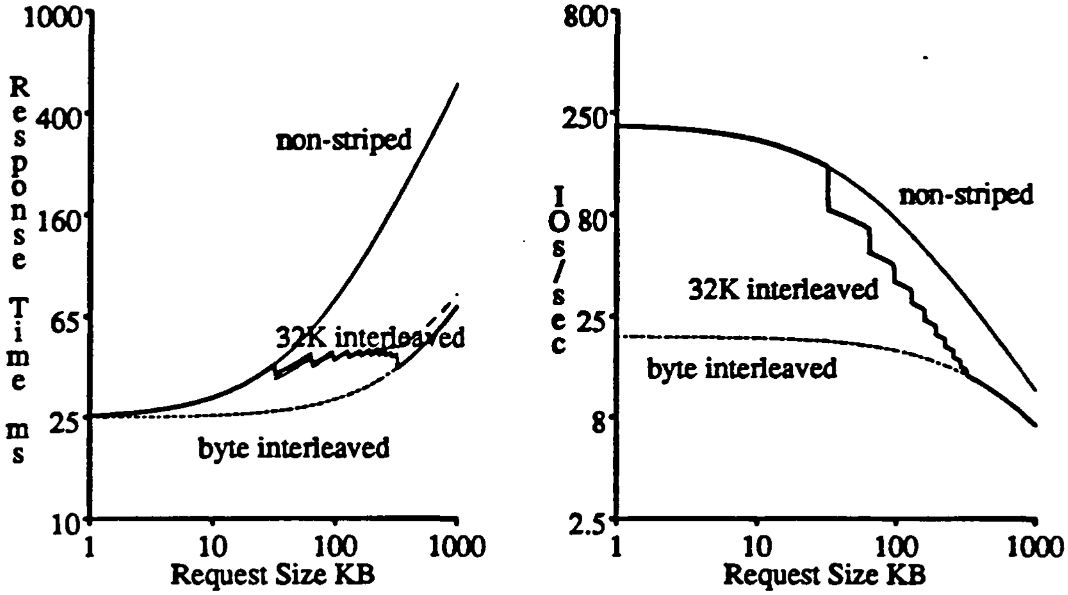


Figure 3.9: Disk Array Performance with 32 KB Striping Unit. *The models shown in Figure 3.8 with a 1 KB block size are repeated here with a 32 KB block size. The next section shows that this is the size of block that Chen et al. [Chen90b] recommend for the disk parameters used by Gray et al. [Gray90] if the array's workload in unknown. In this case, accesses per second in an array with block-interleaved striping, on the left, is much closer to a non-striped array with evenly loaded disks, and is the same for request sizes less than 32 KB. However, request sizes between 32 KB and 320 KB have longer response times at low load with this block size than with the 1 KB block size of Figure 3.8.*

best. The next section reports the results of a formalization of these expectations.

3.4.1.5. Selecting a Best Striping Unit

Because the interaction between workload and striping unit can have a substantial effect on the performance of a disk array with block-interleaved striping, Chen and Patterson developed rules of thumb for selecting a striping unit [Chen90b]. Their simulation-based model evaluated a spindle-synchronized disk array of 16 disks with block-interleaved striping. They used four, stochastic distributions describing the size of each request. These distributions had mean sizes of 4 KB, 16 KB, 400 KB, and 1500 KB. They also varied the number of concurrent, independent requests from 1 to 20. Their goal was to derive the size of a striping unit that gives the largest throughput for an incompletely specified workload. For example, when the concurrency of the workload was known, they found a striping unit that provided 95% of the maximum throughput possible for any particular request distribution. The size of this striping unit is

$$1\ sector + \frac{1}{4} \times average\ positioning\ time \times data\ transfer\ rate \times (concurrency-1) \quad (3.4)$$

where the average positioning time is an average seek time plus an average rotational delay. A striping unit selected by this expression is small when the concurrency is low so that every access can utilize all disks, and larger when the concurrency is high so that more different accesses can be serviced in parallel.[8] Intuitively, the product of average positioning time and data transfer rate balances the benefits and the costs of striping data [Chen90b pp 329]:

> The benefit is the decreased transfer time of a single request, which saves approximately the transfer time of a stripe unit. The cost is the increased disk utilization which arises from an additional disk positioning itself to access the data.

The constant, ¼, is sensitive to the number of disks in the array; they are currently engaged in research into this relationship.

A good size for a striping unit can also be provided when request sizes are known but concurrency is not, although throughput close to maximum will not be assured. For this reason they concluded that a workload's concurrency is more important than its request sizes for selecting a striping unit.

If nothing is known about a workload's concurrency or request size distributions, they proposed that a good compromise size for a striping unit is

$$\frac{2}{3} \times average\ positioning\ time \times data\ transfer\ rate\ . \quad (3.5)$$

Again, research into the relationship between the number of disks in the array and the constant, ⅔, is in progress.

[8] Applying this expression to the model studied by Gray et al., the recommended striping unit is $12.8\times(concurrency-1) + 0.5$ KB. For an average concurrency of 10, Chen et al. recommend that Gray et. al employ a striping unit of about 3½ tracks, and if the average concurrency is 30, the recommended striping unit is about one third of a cylinder (virtually assuring that all accesses involve only one disk).

Because the data transfer rate of a disk is approximately its track capacity divided by its rotation time and because an average rotation time is one-half a full rotation time, this compromise striping size is

$$\frac{2}{3} \times \left[\frac{1}{2} + \frac{\textit{average seek time}}{\textit{rotation time}} \right] \times \textit{track capacity}, \tag{3.6}$$

which will frequently be close to the capacity of one track because average seek time is comparable to rotation time.[9] This compromise size is also appropriate when average seek time is replaced by observed, typical seek time, which is frequently much shorter than a seek from random cylinder to random cylinder [Scranton83].

Together, expressions 3.4 and 3.5 provide powerful tools for the designer of an non-redundant, striped disk array. The next section presents the effects of redundancy on disk array performance.

3.4.2. Performance Lost to Maintaining Redundancy

Redundant disk arrays are able to recover failed disks because they maintain an encoding of user data on one or more disks separate from the disks containing the protected user data. The consistency of this redundant data relative to the data it protects must be maintained continually. The effect of performing this maintenance on the performance of a redundant disk array is the subject of this section.

I begin with a simple, fundamental comparison between the conventional organization of redundancy in disk farms and the two striped, N+1-parity disk arrays described in Section 3.3.1.2, and follow this comparison with the results of a measurement experiment that validates it. The result of these sections is that except for the performance of small, random write accesses, N+1-parity disk arrays with block-interleaved striping have persuasive advantages.

[9] Applying this to the model studied in Gray et al., the recommended striping unit is 34 KB – almost exactly the size of a track and very close to the striping unit used in Figure 3.9.

Consequently, I follow these sections with a discussion of three different approaches to overcoming the performance problems of small write accesses: caching and hints for avoiding unnecessary accesses, a "floating parity" organization that allows parity blocks to be updated in the time of a single access, and a new file system design that converts all small writes into large writes. I finish this section with a discussion of the performance of redundant disk arrays with double-erasure correction; it is little different from those with single-erasure correction except that small write accesses require 50% more accesses to maintain redundant data.

3.4.2.1. Fundamental Differences For Single-Erasure Correction

Figure 3.10 presents a performance comparison between the single-erasure-correcting, redundant disk array organizations discussed in Section 3.3.1. This example uses 10 data disks per array, and assumes saturated disks that perform an average seek during every access. Data is striped across all disks in all three cases. It reports performance in terms of efficiency expressed as a ratio of the accesses per second in the redundant array to accesses per second in a non-redundant array with the same number of disks.

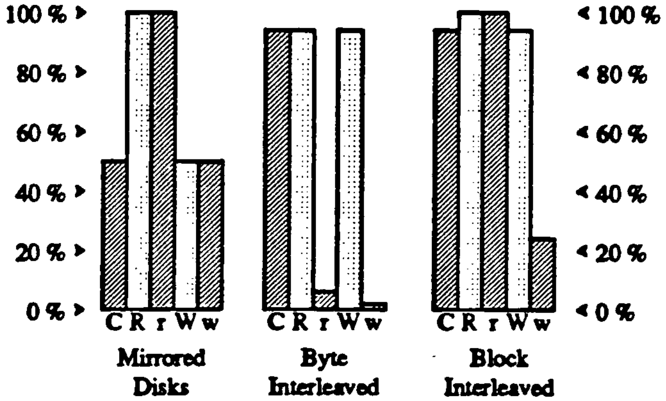


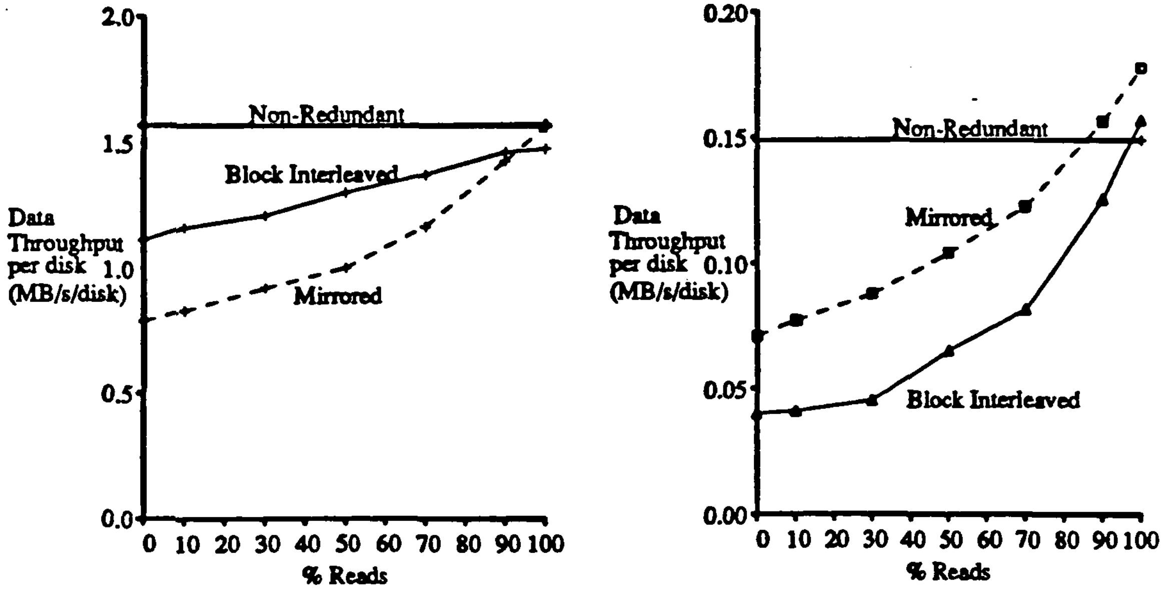
Figure 3.10: Fundamental Redundant Disk Array Performance Comparison. *This comparison of user capacity and performance efficiency in an N+1-parity redundant disk array is derived from Patterson, Gibson, and Katz [Patterson88]. Each array has 10 data disks and enough work to fully saturate it. User capacity (C) is the ratio of the total number of user data bytes to the total number of bytes on all disks. Performance for large reads (R), small reads (r), large writes (W), and small writes (w) is expressed as a fraction of the corresponding performance metric in a saturated non-redundant array of the same total number of disks. Large accesses touch 10 consecutive user data blocks aligned to access exactly those blocks associated with a single parity block (a stripe).*

During reads, mirrored disks can use every disk to access user data (because the model assumes enough work to saturate all disks), so 100% efficiency is secured. Byte-interleaved arrays have one disk, the parity disk, with no user data on it, so they achieve only 91% efficiency. Unfortunately, if a read requests the amount of data in the minimum unit of access on a disk, a sector, a byte-interleaved array will utilize all drives for as long as any single disk takes to read the data (because the minimum access unit is at least one sector per disk). Hence, no parallelism is available, and its efficiency is only 9%. Block-interleaved arrays have data on every disk because the parity is evenly distributed. As a result their efficiency should be 100% on read accesses, both large and small.

During writes, mirrored disks must do two physical writes for every user write, so efficiency on all writes is 50%. Writes large enough to change a parity stripe (one block on every data disk) only waste the bandwidth of the one parity disk in both the byte-interleaved and block-interleaved arrays. Thus, these writes achieve 91% efficiency. Small writes in the byte-interleaved arrays require a preread, merge, and overwrite of all disks, so their efficiency is 5%. Small writes in the block-interleaved array require the four accesses described in Section 3.3.1.2.2, so they get 25% efficiency.

3.4.2.2. Experimental Validation

In an experiment using an Amdahl 5890 mainframe and twenty Amdahl 6380 disks, Chen, Gibson, Katz, and Patterson [Chen90a, Chen89] measured mirroring and block-interleaving redundant arrays. The results of their experiment, summarize in Figure 3.11, showed the above model to be generally correct. In this experiment, a workload was generated stochastically. To equalize response times while allowing for access parallelism, the workload was regulated so that 90% of all accesses responded in less than four times the response time of an access of average size in an idle system. Large accesses were distributed around an average size of 1.5 MB and small accesses distributed around an average size of 0.006 MB (6 KB). They found mirrored disks were 20% more efficient than the model predicted for small reads because the



Figures 3.11a and 3.11b: N+1-Parity Performance Measurements. *This shows the main results of a stochastic measurement study using an Amdahl 5890 with 20 Amdahl 6380 disks [Chen89, Chen90a]. On the left, Figure 3.11a shows that, for large accesses, block-interleaved, N+1-parity disk arrays outperform mirrored disks. In this figure, access sizes are distributed around an average of 1.5 MB per request, or about 150 KB per data disk. The fraction of reads (as opposed to writes) is a major factor in the throughput of large accesses block-interleaved arrays, contrary to the assumptions of the simple model, because large writes are rarely aligned on parity stripes, and data must be preread to update parity at the beginning and end of the data in each request. On right, Figure 3.11b shows that, for small accesses, mirrored disks outperform block-interleaved, N+1-parity disk arrays because updating redundant disks requires two instead of four accesses. When almost all accesses are reads, mirrored disks also benefit from choosing to read the closer copy. Access sizes are distributed around an average of 6 KB per request. In both graphs, the depths of disk queues were regulated so that 90% of all response times were less than four times the idle-system response time of an average size access, and results are shown in terms of the user data rate attained divided by the total number of disks (including redundant disks). Notice that on the right the scale is 10 times smaller than on the left.*

opportunity to choose a shorter seek provided significant advantages when transfer time was small [Bitton88]. They also found that large writes in block-interleaved arrays lose about 30% of the model's predicted efficiency because the access unit is rarely aligned to an integer number of stripes; most writes will begin and end in the middle of a stripe. Writes averaged 1.5 MB, about four logical blocks (in this case, tracks) from each disk, so on average three full stripes were overwritten and two partial stripes required prereading to update parity.

3.4.2.3. Optimizations for Small Write Accesses

Except for the additional work involved in a small, random write access, Figures 3.10 and 3.11 show that N+1-parity disk arrays with block-interleaved striping are comparable or superior to mirrored disk arrays. This section describes three approaches for limiting or removing this additional work: exploiting applications hints and caches to avoid unnecessary accesses, dynamically reorganizing parity locations so that read-modify-write operations can be performed in little more than the time of one access, and massively reorganizing the file system to avoid performing small write operations in the disk array entirely.

3.4.2.3.1. Caching and Hints

First of all, once data to be written is copied into the file system's cache, the program issuing the write can continue. This means that the response time of a write access experienced by applications programs can be fast regardless of the amount of work involved in propagating the write to the disk array. Then, because these propagated write operations are delayed, they may benefit from file cache optimizations such as deletion without ever being written [Nelson88], and from request reordering when a large group of cached blocks are eventually written to disk [Bates89, Geist87, Seltzer90a].

In many applications that feature small, random accesses, (on-line transaction processing for example) most writes are the second half of a read-modify-write sequence. These applications, therefore, have sufficient information to maintain a copy of the data as it was on disk, and provide to the file system both the old and new data when it issues the write access. This reduces the number of physical operations done by an N+1-parity disk array while processing the write access from four to three.

In practice, it is more beneficial if old copies of read data are maintained in the file system instead of the user's buffers because the integrity of the disk array's redundancy depends on the old copy of the data being a current image of its corresponding disk block. Moreover, the file system cache may also have a copy of the data's old version because the space it is occupying in the cache has not yet been reused. To increase the chance that the data's old version is in the cache, an application initiating a read-modify-write sequence can inform the file system that it intends to rewrite the data. This information allows the file system to make extra efforts to hold the original copy of the data until the rewrite is issued.

In these ways the use of caches and hints hide the latency of small, random writes from applications and potentially eliminate and unnecessary reread operation during a read-modify-write sequence.

3.4.2.3.2. Floating Parity

Menon and Kasson proposed a variation on the organization of parity data in an N+1-parity disk array that shortens the read-modify-write of parity data changed by a small, random write to little more than a single disk access time on average [Menon90]. Their recommended scheme, called *floating parity*, can be viewed as an extension of an N+1-parity disk array with distributed parity (striping is incidental). In a floating parity organization, parity blocks are clustered into cylinders each containing a track of unallocated blocks. Whenever a parity block needs to be read, modified to reflect a data block change, and rewritten, the new parity block can be written on the nearest unallocated block following the old parity block. Menon and Kasson show that for disks with 16 tracks in a cylinder, this nearest block is immediately following the parity block being read on one of the tracks in the parity cylinder with probability 0.65, and that the average number of blocks that must be rotated past to get to this nearest block is between 0.7 and 0.8 in practice. With such large probabilities of finding an unallocated block nearby the parity data being read, the rewrite can be done immediately, and the entire read-modify-write is done in only a few milliseconds longer than is taken for the read access alone. For example, the read-modify-write of a parity block on a Tandem XL80 disk (described in Section 3.4.1.4 above) would take an average of 42 msec without floating parity and an average of 25.7 msec with floating parity. Because the average time to simply read a block from one of these disks is 25.1 msec, the floating parity scheme effectively completes two disk accesses in the time necessary for only one!

To implement their scheme directories for the locations of unallocated blocks and parity blocks must be stored in main memory (or controller memory). These tables are about 1 MB in size for each disk array containing 4 to 10 disks each with a capacity of 500 MB. Their scheme also surrenders space to the management of parity, but this is a small fraction (1 to 2% for their examples) of the array's capacity. To exploit unallocated blocks immediately following the parity data being read, however, this data must be modified and the disk's head probably must be switched to another track before the head rotates though an inter-sector gap on the disk's surface. Because of these deadlines, and because a disk controller is more likely to have exact knowledge of each disk's geometry, implementation of a floating parity scheme is likely to be found in disk controllers. Still, their scheme looks like a powerful way to reduce the time required to update parity to about the time of a single access.

3.4.2.3.3. Log-Structured File System

Although the combination of caches, hints, and floating parity may reduce the cost of a small, random write in a block-interleaved, N+1-parity disk array to about the same as that of a mirrored disk array, much larger savings are possible by modifying the operation of the file system. Ousterhout, Douglis, and Rosenblum proposed such a modification they call a *log-structured file system* [Ousterhout89, Rosenblum90]. Their system collects data to be written and then writes it all in a large, contiguous disk access. Because rewritten data does not overwrite its previous location on disk, the file system stored on disk is organized as a "log" of created and changed data. To be useful, a log-structured file system must be able to scavenge the space that stores versions of files that have been rewritten later in the log. Rosenblum and Ousterhout proposed doing this by partitioning disk space into large segments and moving data that has not changed in a long time into segments that are dedicated to this kind of 'long-lived" data [Rosenblum90]. This way, the log can simply skip over these segments with little loss in write bandwidth.

A log-structured file system converts all user writes into very large disk writes that can easily be aligned to stripes in an N+1-parity disk array. Not only does this avoid read-modify-write sequences for small-random-writes, but it also fully exploits the large write bandwidth advantage of N+1-parity. Log-structured file systems do not require disk arrays to improve over existing file systems, but in concert, these two technologies provide an I/O system that effectively counters the transfer gap problem described in Section 2.2.

3.4.2.4. Double-Erasure-Correcting Codes

The recipient of much less research, the performance of redundant disk arrays with double-erasure-correcting codes is analogous to that of single-erasure-correcting redundant disk arrays.

As in single-erasure-correcting codes there is an interaction with the organization of data across disks. If data is byte-interleaved across the disks of an array with double-erasure correction, performance is essentially the same as that of N+1-parity except that more disks are engaged in accessing check data. For block-interleaved disk arrays with double-erasure correction, response times and access bandwidth for read accesses and large write accesses are the same as that of block-interleaved, N+1-parity disk arrays with the same number of disks.

For small, random write accesses, all the double-erasure-correcting codes suggested in this chapter must update two check disks for each data disk update, so up to six accesses must be done: preread and overwrite each of the affected data block and its two associated check blocks. All of the optimizations described in the last section, however, are also applicable to double-erasure-correcting codes, so the cost of an application requested read-modify-write operation can be reduced to two more access than would be done in a non-redundant array, and with the log-structured file system, all small writes can be converted to large writes.

3.5. Summary

Disk arrays are the inevitable result of the magnetic disk industry's move to disks with smaller diameters. They provide both manufacturer and customer with a flexible configuration and an impressive environmental efficiency. Because each disk has a smaller capacity, disk arrays with capacities equivalent to the secondary storage systems they replace will have many more independently operable disk units. This increased disk parallelism promises to improve response times for large sequential accesses if data is striped over many disks. It also promises to improve access throughputs for small random accesses if requests are distributed evenly over all disks.

However, increased disk parallelism also brings increased failure rates. While predictive-monitoring software, like DEC's VAXsimPLUS, can help prepare for these failures, it does not guarantee to preserve data threatened by failures. Such guarantees are provided by more expensive data storage approaches, such as on-line redundancy. The first level of protection, single-failure correction, can be provided by the well-known, but expensive, mirrored-disks technique. To reduce costs, an N+1 parity approach also provides single-failure correction. The data reliability of these redundant disk arrays is examined in Chapter 5.

In large arrays or arrays with high reliability requirements, double-failure correction may be necessary. Double-failure correction can be provided by simple binary codes such as the low-cost, full-2 codes or the very high data reliability, 2-d parity codes. In either case the redundancy-maintenance penalty induced by updating a single data block is the modification of at least two other disk blocks. Non-binary codes use more expensive encoding and decoding logic than that of binary encodings but offer the capability to correct all double failures with only two more disks than are needed in a comparable non-redundant disk array. However, non-binary codes involve a much larger fraction of the array in each recovery and suffer a greater variety of uncorrectable triple failures.

The performance advantages that come with increased disk parallelism largely depend on data striping. Block-interleaved striping automatically distributes data so that many small, random accesses can be serviced in parallel, although database applications that have explicit disk, load-balancing software may be better able to do this in non-striped arrays. The unit of interleaving, or striping unit, is important to high throughput for small random accesses; a striping unit size with wide success in the absence of workload knowledge is about the capacity of one track.

For workloads that emphasize large sequential transfers, byte-interleaved striping with synchronized rotations and seeks offer the largest decreases in response time. However, byte-interleaved organizations have a much lower throughput for small random accesses. A block-interleaved striping organization provides nearly as low response times and much higher access throughputs as do byte-interleaved organizations.

Maintaining redundant data incurs costs unavoidably, but these costs can be contained. Because large updates frequently change all the data associated with a block of redundant data, the new values for this block of redundant data can be precomputed without reference to prior values, so, relative to a non-redundant array, there is no response time penalty, and little degradation in throughput. For workloads that emphasize small, random accesses, the cost of maintaining redundant data, which can be as large as four disk accesses for each user-requested data modification, can be ameliorated by caching, applications hints, and floating parity organizations, or completely overcome by a new file system design. With these performance expectations and the much lower cost for redundant data, an N+1-parity disk array with block-interleaved striping is the best organization for a single-erasure-correcting redundant disk array.

CHAPTER 4

Characterizing Disk Lifetimes

Although anecdotes of disk failure models abound, little concrete data has been widely published, and there is no consensus among the many vigorously pressed opinions. Yet the distribution of magnetic disk lifetimes is critical to the proper design of failure-tolerant disk systems. This chapter is a self-contained examination of the distribution of disk lifetimes. It serves to provide an underpinning for the disk array reliability modeling of the next chapter.

After a discussion of commonly-used models for lifetime distributions a recounting of some the disk lifetime anecdotes, and a discussion of the recent, rapid rise in disk manufacturers' advertised mean lifetimes, I offer an analysis of two particular populations of 5¼-inch disks observed over 18 months beginning in 1987. These two populations, totaling 1350 disks, have significantly different lifetime distributions. For example, assuming an exponential distribution for lifetimes and ignoring failures during an initial "breaking in" period, the mean time to failure (MTTF) of the newer product is 115,000 hours while the older product has a 368,000 hour MTTF. These differences are probably derived from the greater maturity of the manufacturing process for the older of these two disk models.

Because the assumption of exponential lifetimes is often suspected of being chosen largely for its mathematical tractability, an important goal of the analysis in this chapter is to test the fit of an exponential model to the observed data. I execute this test by fitting the data to a Weibull distribution model. This model for lifetime distributions has the property that it includes an exponential model as a particular value for its "shape" parameter. Testing the Weibull shape parameter of the data for these two disk populations indicates that the more mature product has lifetimes that are plausibly exponential, but the less mature product's lifetimes are less likely to be described by an exponential distribution.

The "breaking in" or "burn-in" period is commonly thought to be complete by the time the customer receives a disk, but this is not the experience of the data in this chapter. If failures occurring in the customer-directed burn-in test of these disks is included in the lifetime model then greater probabilities must be assigned to short-lived disks. In this case an exponential distribution model estimates a 80,000 hour MTTF for the less mature disks and a 338,000 hour MTTF for the more mature disks. If a customer begins to use disks as soon as they are received from the factory, this is the appropriate model for lifetimes.

4.1. Characterizing Catastrophic Disk Failures

The cost-effectiveness of storing redundant information in a disk array, so that failures can be tolerated during repair or replacement operations, is quite likely to be sensitive to the individual lifetimes of each disk. To use one horrific example, suppose disks really did fail soon after their warranty expired. This would mean that if all drives are purchased from the same manufacturing lot so that they age together, redundancy would be unused until the warranty expires. Then the greater chance of multiple failures in a short period of time might render parity protection, or even complete duplication, futile. Such a scenario would be a data manager's nightmare! (It would also be cheaper and more effective to replace disks just before their warranty expired and not bother with redundancy.) Fortunately, such an eventuality uses an unlikely distribution of disk lifetimes. Unfortunately, however, the true distribution of disk lifetimes are not well-known.

Figure 4.1 shows the commonly used *bathtub* representation for the instantaneous failure rate of an item [Lawless82]. Mathematically, the product of instantaneous failure rate at time t and dt is the probability of failing between time t and $t+dt$ for those devices that are still operational at time t. The function that describes instantaneous failure rate over time is called the *failure* or *hazard rate* function, $h(x)$. It uniquely describes a lifetime distribution because

$$\text{Reliability at time } t = R(t) = \text{Prob}(Lifetime > t) = e^{-\int_0^t h(x)dx} . \tag{4.1}$$

Disk designers prefer to use failure rate functions rather than reliability functions to describe lifetime distributions because these designers have better intuition about the change in failure rate over time than they do about the probability of failure at or before a particular time.

The bathtub shape for failure rates is used to describe the lifetimes of systems as diverse as humans [Lawless82 pp 11] and semiconductor components [Siewiorek82 pp 9]. It is particu-

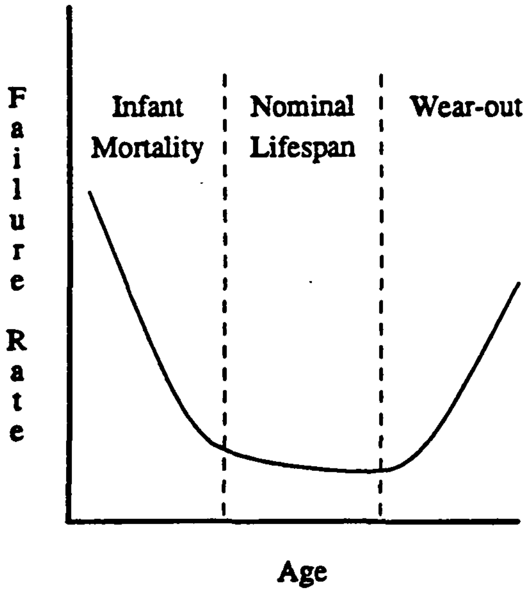


Figure 4.1: The Bathtub Lifetime Distribution. *The instantaneous failure rate of semiconductor components in particular, most manufactured devices in general, and even human lives, can be loosely illustrated by a bathtub curve. New items have a higher frequency of failure as defective ones expire, and old items have a higher frequency of failure as wear overwhelms function. Between infancy and wear-out, failure rates are often relatively constant.*

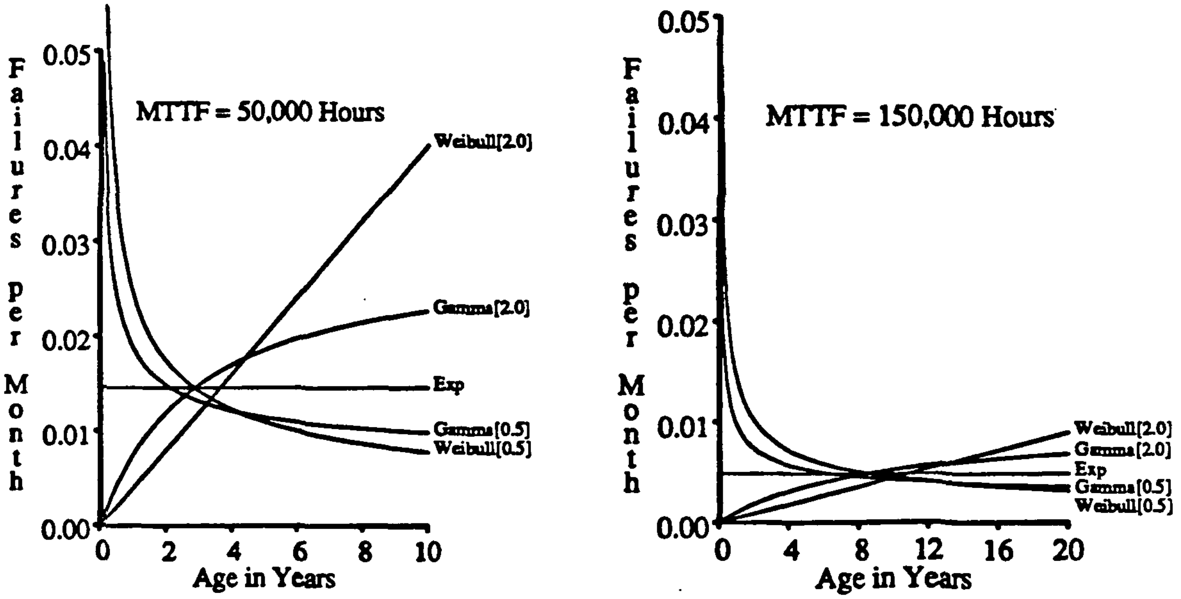
larly useful if a lifetime is to be measured from creation to failure without any initial culling or terminal retirement. If a lifetime is defined to begin after a device burn-in test that culls defective devices and to end with obsolescent disks' retirement prior to the onset of wear-out, then instantaneous failure rates may be relatively constant with time.

One of the simplest and most commonly used approximations for lifetime distribution is derived from a *constant failure-rate* function giving rise to the *exponential* lifetime distribution:

$$R_E(t) = \mathrm{Prob}(Exponential > t) = e^{-t/MTTF} \ . \qquad (4.2)$$

The exponential distribution is characterized more fully in Section 5.1.

The exponential distribution's constant failure rate constrains the probability that it is a good approximation for an arbitrary lifetime distribution. Two more powerful distributions also



Figures 4.2a and 4.2b: Examples of Exponential, Weibull, and Gamma Lifetime Distributions. *The failure rates of three common lifetime distributions are shown here. On the left in Figure 4.2a, the mean lifetime is 50,000 hours (5.7 years), and, on the right in Figure 4.2b, it is 150,000 hours (17.1 years). The exponential distribution* (Exp) *has a constant failure rate equal to the reciprocal of its mean lifetime. Both the Weibull* (Weibull($shape_W$)) *and Gamma* (Gamma($shape_G$)) *distributions have two parameters, but because I have fixed their mean lifetime, only one parameter is free to vary. For these two distributions, I show curves with their shape parameters set to 0.5 and 2.0. Although this graph does not show ages large enough, the Gamma failure rate approaches the exponential failure rate as age approaches infinity. On the other hand, the Weibull failure rate approaches 0 if its shape is < 1 and infinity if its shape is > 1, as age approaches infinity.*

commonly-used to model lifetimes are the Weibull and the Gamma distributions [Lawless82].

$$\mathrm{R}_W(t) = \mathrm{Prob}(Weibull > t) = e^{-(t/scale_W)^{shape_W}} \tag{4.3}$$

$$\text{where } MTTF_W = scale_W \times \Gamma(1 + 1/shape_W) \text{ and } \Gamma(k) = \int_0^\infty u^{k-1} e^{-u}\, du \ .$$

$$\mathrm{R}_G(t) = \mathrm{Prob}(Gamma > t) = 1 - \frac{\int_0^{t/scale_G} u^{shape_G - 1} e^{-u}\, du}{\Gamma(shape_G)} \tag{4.4}$$

$$\text{where } MTTF_G = shape_G \times scale_G \ .$$

Figure 4.2 shows the failure rate of an exponential distribution in comparison with the Weibull and the Gamma distribution. Both of these latter two distributions have parameters called *shape* and *scale*, names which were derived from their effect on the distributions' Probability Density Function (pdf). The shape of the pdf is controlled by the distribution's shape parameter and the time axis scale is determined by the distribution's scale parameter [Lawless82 pp 16]. In Figure 4.2a and 4.2b I have fixed mean lifetime to be 50,000 hours and 150,000 hours respectively. With a fixed-mean lifetime, I have determined the scale parameter as a function of its shape parameter and then displayed the failure rates for shapes equal to 0.5 and 2.0. The exponential distribution, however, is a special case of both the Weibull and Gamma distributions; when either of these has a shape of 1.0, they reduce to an exponential distribution and their scales become equal to one over their *MTTF*. Figure 4.2 shows that over 10 years, if shape is less than 1.0, failure rates increase with age, and if shape is greater than 1.0, failure rates decrease with age. The Gamma distribution also has the property that as age grows to infinity, failure rates return to the failure rate of an exponential with the same mean. But, as Figure 4.2 shows, this is unlikely to effect the useful life of a disk with an hour-mean lifetime between 50,000 and 150,000 hours. The Weibull distribution with a shape of less than 1.0 has the unattractive feature (shared with another common distribution, the log-normal distribution [Lawless82 pp 24]) that, as age increases, the instantaneous probability of failure continues to decrease asymptotically to zero. As a result, care must be taken when using these models to avoid drawing conclusions sensitive to their unintuitive failure rates at large ages.

Section 4.4 of this chapter applies a Weibull model to data on 1350 5¼-inch magnetic disks. I have also applied a Gamma model to this data, but its results were not enough different from the results of the Weibull model to warrant inclusion.

4.2. In Search of Public Data on Lifetime Distributions

From the earliest stages in my research, I sought data on mean disk lifetimes and their lifetime distributions to form the basis for modeling disk array lifetimes in Chapter 5. Because published data was scarce, I asked disk manufacturing companies for theirs. Although these companies apparently have the data I sought, most considered it proprietary and hence too valuable for general publication. Nevertheless, I have collected some overall statistics that pertain to large disk populations.

Section 3.2.1 contains evaluation statistics for the VAXsimPLUS disk-monitoring and failure-prediction software. In that evaluation, 150 disk failures were observed during 7,000,000 disk hours [Lary89]. If the lifetimes for these disks are exponential, then their mean lifetime is 46,667 hours or 5.3 years.

During a presentation in November 1988 [Balanson88], a representative of IBM's disk products division gave the rate of failure in their then top-end product, the 3380, as one every six years. This rate reflects a mean time to failure of about 53,000 hours. An IBM 3380 contains two separate disk enclosures, each with its own spindle and platters. Balanson also said that each of these enclosures had an average lifetime of fifteen years, and that power supplies and control electronics contributed substantially to failures. From this I computed that the mean time to failure of an IBM 3380 disk enclosure was about 131,000 hours and, if failures in different components are independent and exponentially distributed, power supplies and control electronics had a mean time to failure of about thirty years or 263,000 hours. Although there is no information on lifetime distribution either in this data or in that for VAXsimPLUS, both are useful for establishing a mean lifetime in products upon which many companies rely heavily.

Another IBM representative reported that some IBM 2314 disk drives, first shipped to customers in 1966, were not retired until 1980. At retirement, their failure rates were not substantially larger than those of the same model after infant mortality subsided [Brady89]. While increases in the wear-out failure rate had apparently not begun, these disks were fifty times less dense with ten times slower transfer rates and four times slower seek times than those of the models replacing them [IBM80]. This example provides evidence that obsolescence occurs earlier than wear-out in the lifetimes of disks.

In the IBM-compatible peripheral market, the value of a customer's investment in disks and the competition between vendors is large enough that a company was formed to collect failure data from its clients and report summaries back to them. This company, R^+, distills data written into a log file by status system software. The data reported represents the ratio of error activity to use; that is, it gives an overall failure rate. In February 1988, they switched from measuring device operations to measuring actuator months. Actuators are the rigid structure of read/write heads that seek together. Since there are two actuators in each 3380 disk enclosure, there are four actuator hours per hour in a 3380 "box." The switch to measuring actuators represents the collective opinion that a disk wears with each passing revolution more than with each user requested transfer; in particular, a disk whose average workload is half that of another disk is not expected to survive twice as long, and a disk spinning unaccessed will not last forever.

Because old and new drives are treated identically, R^+ data gives no evidence about changes in the rate at which an individual disk fails with age. However, it does give evidence on the relative failure rates of different products. It also gives evidence about the maturity of the manufacturing process for a particular product. By examining product reports spanning several years, then the maturity of the manufacturing process is displayed by trends in average failure rate. As the number of units produced (yielding more experience with thea product) increases, their average failure rate should drop. Similarly, as the model is superceded by newer technology, the number of units produced decreases and average failure rates increase.

During 1988, various IBM 3380-compatible products were reported by R^+ to operate between 100 and 400 actuator months per hard failure. With 730 hours in a month, this indicates that the mean time between failures ranged between 100×730 or 73,000 and 400×730 or 292,000 hours per actuator or 18,000 and 75,000 hours per 3380 box. The older IBM 3350 disk model was in the twilight of its production; with less than 10% as many disks reporting, the 3350 actuators failed once every 30,000 hours.

One source of published data providing some lifetime distribution information described seven Fujitsu Eagle [Fujitsu2351] failures that occurred in a system of 13 Sun2 and Sun3 file servers observed for an average of 19 months each between February 1986 and December 1987 at Carnegie Mellon University [Lin88]. Lin applied a Weibull model to this small sample of failures. The maximum likelihood estimator obtained for the Weibull shape parameter was 0.92. Because 1.0 was included in the 95% confidence interval around the shape, Lin accepted the possibility that the lifetimes of these disks were exponentially distributed, with a mean lifetime of 86,900 hours. Because these disks are advertised as having a mean time to failure of only 20,000 hours [Fujitsu2351], Lin's results suggest that estimated mean lifetimes significantly underestimate actual mean lifetimes. Some representatives of disk manufacturers have also suggested, and quite tantalizingly, that observed average lifetimes are much higher than is claimed by product specifications for mean time to failure. Their contention has been borne out by a leap in advertised mean lifetimes to a range of 100,000 hours to 250,000 hours. Section 4.3 explains how this change was largely the result of changes in the way that advertised mean lifetimes were calculated.

After encountering the intransigence of disk manufacturers and discovering the expense of tracking reliability, I was not surprised to find that computer users with large collections of disks were also unable to provide data appropriate for describing distributions of disk lifetimes.

It seems that although accounting and security departments might track devices by serial number, service and repair departments rarely do. Even when operations staff do maintain records, they are often uninterpretable and irreconcilable.

4.3. Improving Disk Mean Lifetimes

A scan through specifications sheets for disk products introduced during the last decade reveals an intriquing trend. Until 1989, the mean lifetimes of disks were specified to be between 20,000 hours to 40,000 hours, but, beginning in the second half of 1989, specifications for mean lifetimes jumped to 150,000 hours, and in a few cases 200,000 hours or more [EET89]. If similar improvements for the mean lifetimes of disks can be expected to occur in the near future, is it possible that disk arrays will not need to be redundant?

A closer examination of the events of 1989 reveals that most of the improvement is caused by changes in the way that mean lifetime specifications are calculated. Two of the more important changes were the inclusion of data from failed disks returned to the manufacturer and a reduction in the expected frequency with which disks are turned off and on. Taken together, these "one time" changes make the methods for calculating mean lifetimes much more optimistic.

Before 1989 mean lifetimes were estimated by a conservative, theoretical model of the component parts in a disk product. In 1989 the producers of 5¼-inch disks at Hewlett-Packard began to include data on the failed drives that had been returned to them into their estimates for mean lifetime. These returned disks are called *field returns*, but they do not include all failed disks. There are many reasons that failed disks may not be returned; for example, their warranties may have expired, they may have been employed at a "secure site" such as a military intelligence organization, or their owner may have violated conditions of their warranty. Because field returns underestimate the number of failures that have occured, mean lifetime estimates based on them are much more optimistic than the estimates of a purely theoretical model.

The other important change in methods for estimating mean lifetime was a reduction in the frequency that customers are expected to cycle the power to their disks. Because all but disks with the smallest diameter are usually employed in systems that operate 24 hours a day, disk designers decided that their prior estimates for the frequency of power cycling were too high. Because turning power off and on is a particularly stressful operation for electronic parts, eliminating power cycles greatly improves their reliability. Similarly, turning the power off and on causes a disk's platters to stop and then start spinning. When a disk's platters stop spinning, its read-write heads "land" on the platter's surface; these heads must "take-off" again when power is reapplied. Reducing the frequency of power cycles greatly enhances mechanical reliability because one of the most important causes of mechanical failure is the inability of the heads to "take-off" (called the stiction problem). It is no surprise then that by assuming customers power cycle their disks less frequently, manufacturer's theoretical estimates for the mean lifetimes of disks were also substantially improved.

Although these changes have had a "one time" effect on the specifications for the mean lifetime of disks, other changes are causing on-going improvements to disk reliability. Primary among these latter changes is the increasing number of disks built each year by the major manufacturers. With larger numbers of disks being built every year, the cost of research and development of more reliabile disks contributes less to the cost of each disk. Similarly, the more disks that get built, the better the manufacturing process can be expected to become. Additionally, with more disks being sold, the cost of retroactively correcting a design fault becomes so prohibitive that product must be more exhaustively tested before being shipped to customers.

Finally, the magnitude of 1989's change in mean lifetime specifications stirred considerable interest for more highly reliable disks among disk customers. Manufacturers now compete for the priviledge of offering the disk with the highest reliability. Although this is not expected to caused mean lifetimes to jump in the way they did in 1989, it does mean that reliability should improve more quickly than it had been in the decade before.

Returning to the question with which I began this section, the recent improvement in disk product specifications for mean lifetimes is largely a "one time" change. Although residual emphasis on mean lifetime specifications and the increasing volume of disks being manufactured will cause continued improvements in disk reliability, this effect will not overcome the increasing numbers of disks in arrays designed to match the growth of performance in processor and multiprocessor technology.

4.4. A Sample of Lifetime Data

One productive approach to collecting data was canvassing the vendors of computer systems who purchase disks to resell in their own products. Even in this case most companies did not track disk failures by serial number. I was fortunate to find that Thinking Machines Corporation, a computer company specializing in massive parallelism, had tracked failures in two different 5¼-inch disk products used in their systems [TMC87]. From January 1989 to June 1990 inclusive, they collected statistics from an in-house, burn-in screening test and from customers who returned field failures. These statistics, shown in Tables 4.1 and 4.2, were made available for publication with the understanding that the identities of disk vendors and Thinking Machines Inc.'s customers were to remain private.

In-house Burn-in Testing			
Test Quarter	Disk Count	Observation Hours	Failure Time
Q1/89	3	500+	?
	45	500+	no fail
Q3/89	45	100	no fail
	210	?	no fail
Q4/89	1	240+	47
	1	240+	48
	1	240+	240
	44	220	no fail
	42+	?	?
Q1/90	1	348	310
	1	500+	343
	1	1500+	1440
	88	142	no fail
	45	348	no fail
	48	500+	no fail
	56	1500+	no fail
Q2/90	1	112	93
	45	112	no fail
	45	225	no fail
	45	100+	no fail
	46	70	no fail
	45	213	no fail

Field Return Results			
Install Quarter	Disk Count	Observation Hours	Failure Time
Q1/89	2	12792	792
	1	12792	6120
	2	12792	8016
	1	12792	10176
	1	12792	10800
	1	12792	11184
	1	1608	no fail
	1	1992	no fail
	1	2616	no fail
	2	4776	no fail
	1	6672	no fail
	2	12000	no fail
	34	12792	no fail
Q2/89	1	10536	3048
	1	10536	5928
	2	10536	8064
	2	2472	no fail
	1	4608	no fail
	1	7488	no fail
	38	10536	no fail
Q4/89	1	5448	2160
	1	5448	2832
	1	2616	no fail
	1	3288	no fail
	40	5448	no fail
Q1/90	1	2568	0
	1	3552	1080
	1	3624	1152
	1	3624	1512
	1	3624	1536
	1	3576	1848
	2	3576	1896
	1	2568	2208
	1	360	no fail
	2	1680	no fail
	1	1728	no fail
	1	2088	no fail
	1	2112	no fail
	2	2472	no fail
	41	2568	no fail
	41	3552	no fail
	39	3576	no fail
	81	3624	no fail
	42	3960	no fail
Q2/90	42	72	no fail
	84	624	no fail
	84	1536	no fail
	42	1560	no fail
	42	1800	no fail
	126	1824	no fail

Table 4.1: Lifetime Data Sample One. *This table shows the burn-in testing and field return results of 5¼-inch disks with about one-third of a gigabyte of capacity. A total of 859 disks went through burn-in testing; 10, or 1.16%, failed. Of the disks that passed burn-in testing, 821 have been operational in the field, and 23, or 2.80%, failed prior to the end of June 1990. Disks that did not fail during the observation period have "no fail" in their Failure Time column. Unknown data is indicated with a "?". Where a number is followed by a "+" the actual value may be larger than is shown. All times are expressed in units of hours. The burn-in testing information for the last entry in Q4/89 has been misplaced, although at least 42 disks were purchased and installed in a customer site. Field data is shown for the quarter that disks went into service. Failed disks returned on the same day are assigned the same lifetime, so multiple identical lifetimes are more likely to indicate return logistics than coupled failures. In addition, the number of newly purchased disks that were "dead on arrival" in each of the quarters shown in the in-house, burn-in testing table was 0, 0, 0, 1, 4, 7, and 0, respectively. These are not included in the analyses that follow.*

In-house Burn-in Testing			
Test Quarter	Disk Count	Observation Hours	Failure Time
Q1/89	? ?	? ?	? no fail
Q2/89	1 1 1 3 ?	900+ 900+ 900+ ? ?	45 47 140 failed? no fail
Q3/89	1 1 43	900+ ? ?	187 360 no fail
Q4/89	3 ?	? ?	failed? no fail
Q1/90	1 ?	? ?	failed? no fail
Q2/90	? ?	? ?	? no fail

Table 4.2: Lifetime Data Sample Two. *This table shows the burn-in testing and field-return results of 5¼-inch disks with about one-fifth of a gigabyte of capacity. The data on in-house, burn-in testing is sparse, but 18 failures were observed in the field between January 1989 and June 1990 from a population of at least 523 operational disks installed after January 1987. The asterisk marks an infeasible entry, originally 18048, assumed to have a transcription error in the 2nd digit. In Q3/89, four or 8.2%, of the newly purchased disks were "dead on arrival"; after Q3/89 no new disks of this type were purchased. For more comments, refer to the caption on Table 4.1.*

Field Return Results			
Install Quarter	Disk Count	Observation Hours	Failure Time
Q1/87	42	30264	no fail
Q4/87	42	22248	no fail
Q1/88	42	20088	no fail
Q2/88	1 1 1 1 80	19344 19344 19344 19344 19344	8808 14232 14376 16728 no fail
Q3/88	1 1 1 81	15672 15672 15672 15672	4152 5640 12000 no fail
Q4/88	1 2 1 1 1 1 77	13488 13488 13488 13488 13488 13488 13488	10752 11376 11880 12672 12696 13048* no fail
Q1/89	2 1 42 1 39	12744 12744 11328 11520 12744	4776 10272 no fail no fail no fail
Q2/89	1 1	10032 10536	no fail no fail
Q3/89	42 2	7656 7968	no fail no fail
Q4/89	1 1 1	5088 4968 5112	672 no fail no fail
Q1/90	1 1 1 1 1	2472 2616 2736 3672 4416	no fail no fail no fail no fail no fail
Q2/90	1 1 1 1 2	440 792 816 1608 2112	no fail no fail no fail no fail no fail

My primary interest in this data was to estimate the distribution of disk lifetimes. First I estimated an exponential model for disk lifetimes. Then I estimated parameters for the more general Weibull model and used this model to test the hypothesis that the data is not taken from an exponential distribution. While this procedure assumes that asymptotic large-sample results apply, it could (but did not) provide strong evidence against using an exponential distribution for disk lifetimes.

Considering the data from field returns in Tables 4.1 and 4.2, if I wished to compute the average observed lifetime, I would get 4,310 and 10,013 hours respectively. But this result ignores the hundreds of field disks that did not fail! To properly calculate the average lifetime for this sample, I would need to wait for each drive to fail. Because this is clearly not feasible, I needed statistical techniques to incorporate what I know about the lifetimes of disks that did not fail while under observation. This kind of lifetime data is said to be *type I* or *time censored* [Lawless82 pp 34]. Fortunately, there are techniques for handling type I censored data, if there is enough data and if particular distributions, including the exponential and Weibull distributions, are under consideration. In the next three sections I examine the data in Tables 4.1 and 4.2 using well-known model estimation techniques [Lawless82].

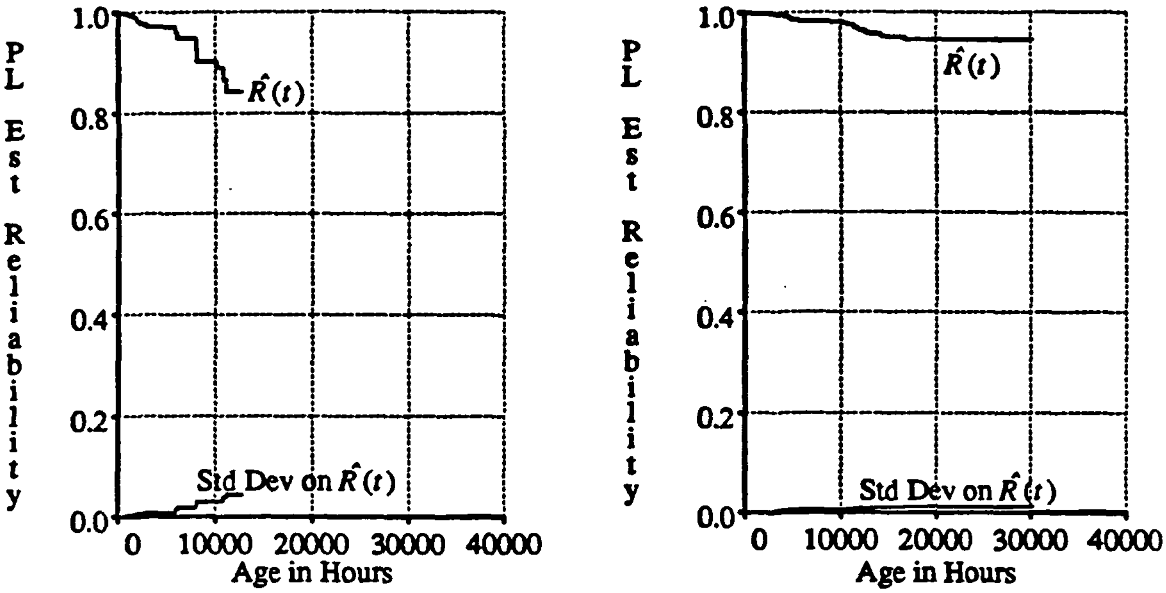
An important problem with the data from the field returns in Table 4.2 is that they include systems installed before data collection began in January 1989. Therefore, all failures occurring before 1989 were not recorded. If I had a large amount of data I could have simply excluded all disks installed before the data collection period. Unfortunately, such a strategy applied to Table 4.2 would eliminate all but four failures – too few to obtain statistically significant estimations of a lifetime distribution. Alternatively, I could have optimistically assumed that no failures occurred before data collection began. In Section 4.4.4 I examine the effect of these data completion treatments in comparison to the extrapolation treatment described next.

I used a third alternative for treating incomplete data; I estimated failures that occurred before data collection began. I used the number of failures in the population, observed over their initial lifetimes, to estimate the number of failures occurring in the unobserved period. After including these estimated failures into the data, I repeated the process for successively longer unobserved periods. The description that follows reflects the adjustments I made to the field returns data in Table 4.2. Because the 84 disks installed in the fourth quarter of 1988 were unobserved for their first 408 hours and because there were no failures in the 145 disks that were observed during their first 408 hours, I estimated zero additional failures in the fourth quarter of 1988. By repeating this with the 84 disks installed in the third quarter of 1988, the 145 + 84 = 225 disks observed in their first 2592 hours suffered one failure, I estimated 84/225 ≈ 0 additional failures for these disks before 1989. For the 84 disks installed in the second quarter of 1988, the 313 disks observed in their first 6264 hours suffered five failures, and so I estimated 5×84/313 ≈ 1 failure from these disks before 1989. I then placed this estimated failure in the middle of its unobserved period at 3132 hours. As for the 42 disks installed in the first quarter of 1988, the 397 disks observed in their first 7008 hours suffered six failures, therefore I estimated 6×42/397 ≈ 1 failure at 3504 hours. Once again for the 42 disks installed in the fourth quarter of 1987, the 439 disks observed in their first 9168 hours suffered eight failures. From this, I estimated 8×42/439 ≈ 1 failure at 4584 hours. Finally, for the 42 disks installed in the first quarter of 1987, the 481 disks observed in their first 17184 hours suffered 21 failures, and so I estimated 21×42/481 ≈ 2 additional failures. I placed these failures at one-third, 5728 hours, and two-thirds, 11456 hours, during their unobserved periods.

4.4.1. Empirical Reliability

A general way to look at lifetime data is to compute and graph the *empirical reliability* or *survivor* function, $\hat{R}(t)$. When data has no censoring, this can be computed as the number of lifetimes greater than or equal to t divided by the number of disks under observation. With censored data, an alternative formulation, called the *product-limit* (PL) estimate of the reliability function, is needed because the number of lifetimes longer than t is not generally known [Kaplan58 from Lawless82 pp 71].

The PL reliability function estimate is defined as a group of n disks observed to collectively suffer one or more deaths at each of k distinct times, $t_1<t_2< \cdots <t_k$. The number of deaths at time t_j is $deaths_j$. Also, pop_j, the number of items at risk at t_j, is defined to be the



Figures 4.3a and 4.3b: Product-Limit Estimates for Reliability. *These two figures show product-limit estimates, $\hat{R}(t)$, for the reliability function on the field returns data in Tables 4.1 and 4.2 respectively. Reliability at time t, R(t), is the probability that an item will survive at least until t. Also shown is the standard deviation on the PL estimate for R(t). There are 365.25 × 24, or 8766, hours in a year.*

number of items operational and under observation (uncensored) just prior to t_j. The PL estimate of the reliability function is then,

$$\hat{R}(t) = \prod_{t_j<t} \frac{pop_j - deaths_j}{pop_j}. \qquad (4.5)$$

The estimate produced by this formula converges to the true reliability function, independent of any assumed or modeled distributions, in large data samples. This property of the estimator means that without knowing the true distribution, the PL estimate of the reliability function can be used to look for possible distributional matches.

Figures 4.3a and 4.3b show the PL estimate of the reliability function, together with its standard deviation, for the field return results in Tables 4.1 and 4.2 respectively. Even though the data in Table 4.1 covers less than half as much observation time as the data in Table 4.2, it is noticeably less reliable. Because the disks noted in Table 4.1 are newer and denser products than the disks in Table 4.2, their lower reliability is probably attributable to a less mature manufacturing process. I would like the distributions of their disks' lifetimes to be apparent from these figures, however, there is not enough data to make any models obvious. Probing further, I assumed particular distributions and assessed their match with the data.

4.4.2. Exponential Model

If the true distribution of disk lifetimes is exponential, then its true reliability function is

$$R(t) = e^{-t/\theta}, \tag{4.6}$$

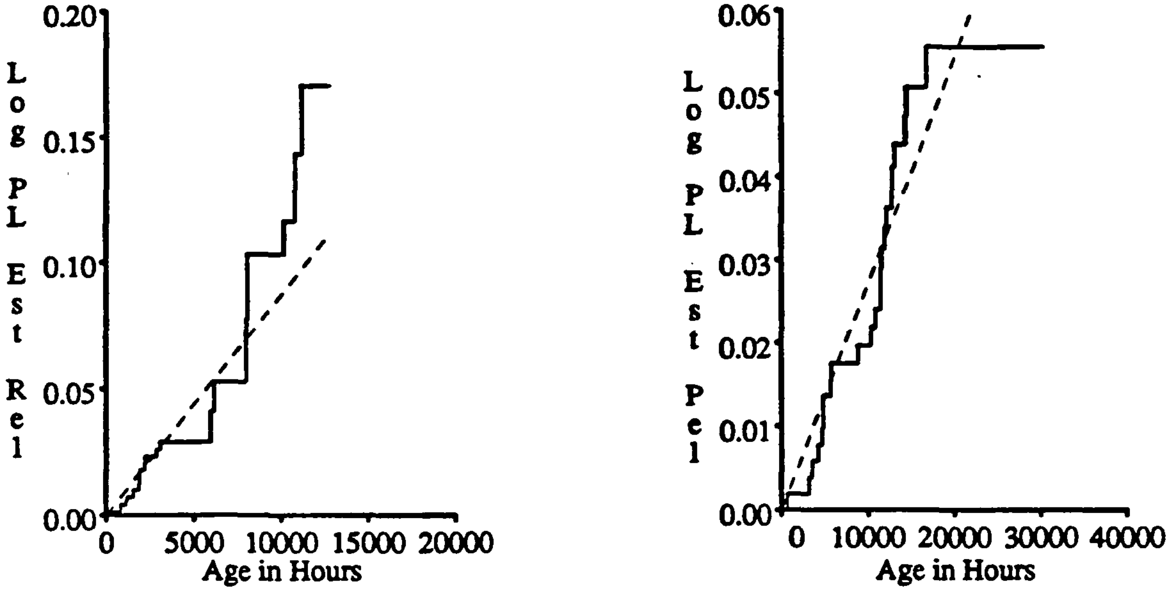
where θ is a parameter that is equal to the mean lifetime ($MTTF$) for this distribution. In this case, $-\text{Log}(\hat{R}(t))$ should be linear in t, pass through the origin, and have a slope of $1/\theta$.

Figures 4.4a and 4.4b show $-\text{Log}(\hat{R}(t))$ corresponding to Figures 4.3a and 4.3b respectively. While the data in Figure 4.4b may be linear in t, it is less clear that the data in Figure 4.4a is linear in t. Also shown in these figures are the maximum likelihood estimators derived next.

Assuming that the true distribution of disk lifetimes is exponential, the *maximum likelihood estimator* (mle) of mean lifetime is the total observation time, T, over the number of failures, r. The total observation time is the sum of the lifetimes of disks that were observed to fail plus the sum of observation times for disks that were not observed to fail. To obtain confidence intervals for mean lifetime with this data I must assume that I have a large sample of data; however, a simple transformation of the statistics has been shown to be accurate for the kind of small samples I present here [Lawless82, Sprott73]. Using this transformation, the statistic

$$3(\hat{\phi}-\phi)/(\hat{\phi}\sqrt{r}), \text{ where } \phi = MTTF^{-1/3} \text{ and } \hat{\phi} = (T/r)^{-1/3}, \tag{4.7}$$

has a normal distribution with a mean of zero and a standard deviation of one. The 95% confidence interval for the standard normal distribution, -1.96 to 1.96, is then put through an inverted transformation again to give a 95% confidence interval on the mean lifetime of these disks.



Figures 4.4a and 4.4b: Graphical Estimation of Exponential Reliability. *These figures show the negative natural logarithm of $\hat{R}(t)$ against t corresponding to Figures 4.3a and 4.3b respectively. If disk lifetimes are exponential, these figures should look approximately linear in t. The dashed lines show maximum likelihood estimations for R(t) assuming lifetimes are exponential.*

Data Source	Data Type	MTTF mle	95% conf. int.
Table 4.1	Field Returns Opt. Burn-in Merged	115,000 24,000 80,000	78,000 - 178,000 14,000 - 48,000 58,000 - 115,000
Table 4.2	Field Returns Opt. Burn-in Merged	368,000 40,000 338,000	251,000 - 571,000 13,000 - 255,000 234,000 - 516,000

Table 4.3: Exponential Lifetime Distribution Maximum Likelihood Estimates. *Assuming Tables 4.1 and 4.2 show data that are type I-censored samples of an exponential lifetime distribution, this table shows the maximum likelihood estimation of the mean lifetime and its 95% confidence interval. Because burn-in data has so many unknown entries, I have made optimistic assumptions where possible. In particular, I assumed that failure occurs, if unknown, at the end of the test. I also assumed that burn-in observation periods were 100 hours for all of the Q3/89 data in Table 4.1 and 900 hours for the Q3/89 data in Table 4.2. However, where the numbers of disks tested and failed were not known, I did not include these entries; this causes me to neglect the last entry in the fourth quarter of Table 4.1's 1989 burn-in data and all but the third quarter of Table 4.2's 1989 burn-in data. With these deletions, the burn-in results for Table 4.2 are not very significant because only two failures were included. The "merged" set of estimations assume that the field disks and burn-in failed disks are independent samples from the same population and combines their statistics.*

Table 4.3 shows the results of these statistical techniques. Based on this analysis, the in-house, burn-in test is useful because it culls disks with much higher failure rates than are ordinarily seen in the field. Since most customers do not conduct an in-house, burn-in test of all disks they purchase, I have assumed that the field disks and the burn-in disks that failed are independent and estimated the exponential mean lifetime for their merged statistics. In this calculation I have neglected the disks that passed burn-in testing because they became the disks in the field. Based on these results, I model the lifetime of a newly delivered disk as exponential with a mean between 58,000 and 115,000 hours for the disks in Table 4.1 and between 234,000 and 516,000 hours for the more mature disks in Table 4.2. It should be remembered, however, that field-return data errs on the optimistic side as I explained in Section 4.3. In the next section, I give evidence that exponential lifetimes for both populations of disks are plausible.

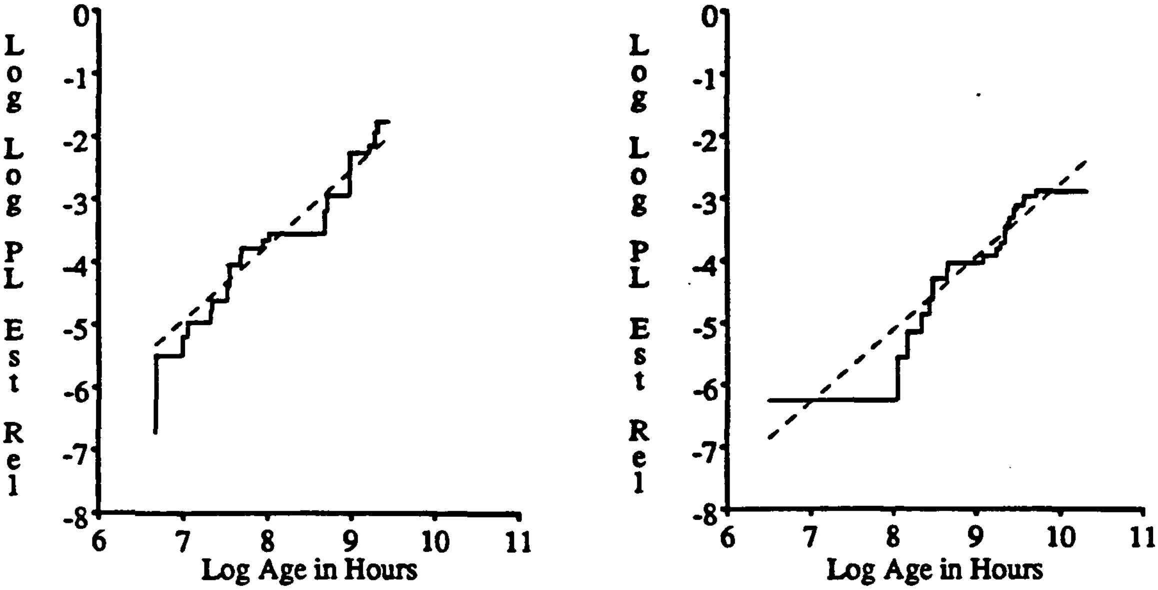
4.4.3. Weibull Model

Fitting the data in Tables 4.1 and 4.2 into a Weibull distribution offers an alternative model for disk lifetimes. More importantly, it offers a technique for testing the hypothesis that the data fits an exponential distribution because it includes the exponential distribution as a special case.

In the same way that I graphically compared the PL estimate of the reliability function with an exponential distribution in the previous section, I can now compare the reliability function with a Weibull distribution. For the Weibull distribution, the reliability function is

$$R(t) = e^{-(t/scale_W)^{shape_W}}, \tag{4.8}$$

so Log(–Log($\hat{R}(t)$)) should be linear in Log(t), have a slope $shape_W$, and have an x-intercept equal to Log($scale_W$). Figures 4.5a and 4.5b show graphs of Log(–Log($\hat{R}(t)$)) against Log(t) corresponding to the data of Figures 4.3a and 4.3b, respectively. Because both figures are plausibly linear, a Weibull model seems to be appropriate for both disks' field-returns data.



Figures 4.5a and 4.5b: Graphical Estimation of Weibull Reliability. *These figures show the natural logarithm of the negative natural logarithm of $\hat{R}(t)$ against* Log(t) *corresponding to Figures 4.3a and 4.3b respectively. If disk lifetimes fit a Weibull distribution, these figures should look approximately linear in* Log(t). *The dashed lines show maximum likelihood estimations for $R(t)$, assuming disk lifetimes have a Weibull distribution.*

The theory for determining maximum likelihood estimators for Weibull parameters *shape*$_W$ and *scale*$_W$ is more complex than that for the exponential parameter θ. If there are n disks of which r fail under observation, let x_i be either the lifetime of the ith disk, if this disk fails, or its censoring time, if it does not. Recall that an observation is said to be censored if it is not seen to fail; the censoring time for any observation is the end of the observation period. Also, I define the mutually exclusive sets *Deaths* and *Censors* to be the disks that fail under observation or are censored before failing, respectively. The maximum likelihood estimator, $\hat{\beta}$, for the Weibull *shape*$_W$ parameter is obtained by solving, iteratively,

$$\frac{\sum_{i=1}^{n} x_i^{\hat{\beta}}\mathrm{Log}(x_i)}{\sum_{i=1}^{n} x_i^{\hat{\beta}}} - \frac{1}{\hat{\beta}} - \frac{1}{r}\sum_{i\in Deaths}\mathrm{Log}(x_i) = 0 \qquad (4.9)$$

for $\hat{\beta}$. Then the maximum likelihood estimator, $\hat{\alpha}$, for the Weibull *scale*$_W$ parameter is calculated as

$$\hat{\alpha} = \left[\frac{1}{r} \sum_{i=1}^{n} x_i^{\beta} \right]^{\frac{1}{\beta}} . \tag{4.10}$$

Because one of my goals for building a Weibull model was to test the hypothesis that the data is appropriately modeled by an exponential distribution, I also needed to compute confidence intervals on $shape_W$. (A 95% confidence interval on $shape_W$ that does not include 1.0, indicates strong evidence against an exponential distribution.) As I did in the previous section for computing confidence intervals, I also assumed that large data samples are available. However, the method I used, based on the log-likelihood ratio statistic

$$\Lambda = -2\mathrm{Log}\left[\frac{L(\alpha_0,\beta_0)}{L(\hat{\alpha},\hat{\beta})} \right] \text{, where} \tag{4.11}$$

$$\mathrm{Log}(L(\alpha,\beta)) = r\,\mathrm{Log}\beta - r\beta\mathrm{Log}\alpha + (\beta-1) \sum_{i \in Deaths} \mathrm{Log}(x_i) - \sum_{i=1}^{n} (x_i/\alpha)^{\beta} , \tag{4.12}$$

has been shown to be fairly accurate when as few as 20 failures were observed [Lawless82 pp 178]. Under the hypothesis that $shape_W = \beta_0$, Λ is distributed as a chi-squared random variable with one degree of freedom, $\chi^2_{(1)}$, where

$$\alpha_0 = \left[\sum_{i=1}^{n} \frac{x_i^{\beta_0}}{r} \right]^{\frac{1}{\beta_0}} . \tag{4.13}$$

The 95% confidence interval on $shape_W$ is then the set of β_0 that satisfies $\Lambda \le \chi^2_{(1),.05} = 3.841$. With a computational tool that manipulates and solves mathematical expressions, these calculations are quite manageable. I used the *Mathematica* mathematics package [Wolfram88] whose plotting and root-finding facilities, and special functions library greatly enhance exploratory data manipulation.

Table 4.4 shows the results of my computation on the data in Tables 4.1 and 4.2. Because all of the 95% confidence intervals on $shape_W$ in Table 4.4 include the value 1.0, there is not strong evidence against the hypothesis that this data is taken from an exponential distribution. Because of this result, I use an exponential model for disk lifetime in Chapter 5.

Data Table	Data Type	$shape_W$ mle	95% conf. int.	MTTF int. (1,000 hours)	R(1 year) int.
4.1	Field Merged	1.2 0.8	0.85 - 1.56 0.59 - 1.04	221 - 34 1,037 - 71	0.933 - 0.905 0.924 - 0.894
4.2	Field Merged	1.2 0.9	0.78 - 1.66 0.62 - 1.30	1,000 - 100 3,220 - 160	0.973 - 0.985 0.967 - 0.979

Data Table	Data Type	$shape_W$ 70% c.i.	MTTF int. (1,000 hours)	R(1 year) int.
4.1	Field Merged	1.00 - 1.40 0.68 - 0.92	114 - 45 418 - 107	0.926 - 0.911 0.918 - 0.902
4.2	Field Merged	0.95 - 1.42 0.75 - 1.11	441 - 137 1,105 - 244	0.975 - 0.982 0.970 - 0.976

Table 4.4: Weibull Lifetime Distribution Maximum Likelihood Estimates. *Assuming Tables 4.1 and 4.2 show data that are type I censored samples of a Weibull lifetime distribution, I show the maximum likelihood estimators for the shape parameter and ranges for the shape parameter, the corresponding MTTF, and one-year reliability intervals. The first table shows ranges that have a 95% confidence of containing the actual shape parameter's value. This range is quite large because the collected data is not large enough to tightly describe Weibull parameter values. For a tighter range, the second table shows ranges that only have a 70% confidence of containing the actual shape parameter's value. The MTTF, and, in some cases, the one year reliability intervals, are inverted so that their left endpoint corresponds to the left endpoint of the 95% confidence interval around the shape. The Weibull shape, and not the scale parameter, is shown because it is not likely that the data is drawn from an exponential distribution when its value is unlikely to be 1.0. The "merged" set of estimations assume that the field disks and burn-in failed disks, described in the caption to Table 4.3, are independent samples from the same population, and thus merges their statistics.*

A closer examination of the results in Table 4.4, however, reveals less positive support for an exponential model. Because the 90% confidence interval on $shape_W$ for the merged data in Table 4.1 is 0.617 to 0.998, excluding 1.0, there is good evidence that this data is not taken from an exponential distribution. Similarly, the 70% confidence interval on $shape_W$ for the field-returns data in Table 4.1 barely includes 1.0, so there is some evidence that $shape_W$ is larger than 1.0. Not only do these two results present evidence against an exponential distribution, they actually argue for opposing types of lifetime distributions. While these field-returns results hint at an increasing failure rate distribution (shape > 1.0), the inclusion of early deaths overwhelms this effect and suggests a decreasing failure rate distribution (shape < 1.0) overall.

What has happened here is that the Wiebull lifetime model is not appropriate for the merged data. A Weibull model allows either strictly increasing or strictly decreasing failure rates, but this data suggests that the failure rate is initially decreasing and is eventually increasing. That is, the merged data for these disks has a pronounced "bathtub" shape. The Weibull model's weakness is demonstrated by the counter intuitive increase in mean lifetime when the short lifetimes of burn-in testing are added to the field-returns data. Notice that this is also true, to a lesser degree, of the data in Table 4.2.

One way to deal with this discrepancy would be to "stitch" together two separate failure rate curves, one representing infant failure rates and the other representing field failure rates. Because an exponential lifetime approximation is satisfactory for my purposes, I have not pursued this further.

4.4.4. Sensitivity

The field-returns data in Table 4.2 reports failures in systems installed before the start of the data collection period. Table 4.5 presents the sensitivity in the exponential and Weibull lifetime models to three alternative treatments of the unobserved period. In the previous sections, I have extrapolated the collected data by estimating unobserved failures from data on fully-observed disks, which I described when I introduced Table 4.2. Table 4.5 shows results from previous sections using this extrapolation in the entry labeled "extrapolation." The obvious way to avoid data with unobserved periods is to neglect all observations that begin before January 1989. Unfortunately, this reduces the number of observed field failures to four. Such a small sample means that the methods of likelihood ratio that I used to estimate confidence intervals should not be regarded as useful. Despite this, however, Table 4.5 presents confidence intervals under this treatment of the unobserved period in the entry labeled "pessimistic." The smallness of the sample is evident in the largeness of the width of these confidence intervals in

Exponential Model of Table 4.2 Data

Data Type	Modification	MTTF mle	95% conf. int.
Field	Pessimistic	350,000	150,000 - 1,150,000
	Truncated	367,000	239,000 - 607,000
	Extrapolated	368,000	251,000 - 571,000
	Optimistic	470,000	305,000 - 778,000
Merged	Pessimistic	233,000	115,000 - 593,000
	Truncated	330,000	219,000 - 531,000
	Extrapolated	338,000	234,000 - 516,000
	Optimistic	423,000	280,000 - 682,000

Weibull Model of Table 4.2 Data

Data Type	Modification	*shape*$_w$ mle	95% conf. int.	MTTF int. (1,000 hours)	R(1 year) int.
Field	Pessimistic	0.9	0.30 - 2.10	1,300,000 - 56	0.973 - 0.980
	Extrapolated	1.2	0.78 - 1.66	1,000 - 100	0.973 - 0.985
	Optimistic	1.3	0.82 - 1.92	1,070 - 88	0.979 - 0.991
Merged	Pessimistic	0.5	0.22 - 1.06	24,000,000 - 196	0.960 - 0.963
	Extrapolated	0.9	0.62 - 1.30	3,220 - 160	0.967 - 0.979
	Optimistic	0.9	0.60 - 1.37	3,714 - 179	0.973 - 0.984

Table 4.5: Parameter Estimates for Alternative Data Extensions. *This table shows the effect three alternative treatments have upon the unobserved period in the field-returns data in Table 4.2 on exponential and Weibull lifetime models. A pessimistic approach neglects all observations that began before data collection. This is ineffective because it reduces the sample size below a useful level. An optimistic approach assumes no field failures occurred prior to the beginning of data collection. Where disk lifetimes are thought to be strictly exponential, truncating a period from the beginning of a lifetime is a rigorously correct alternative. The data extrapolation used in the previous sections is shown for the sake of comparison.*

the Weibull model.

If the only lifetime model under consideration is the exponential one, I can use the *memoryless* property of an exponential random variable to correct for the unobserved period. Exponential random variables are said to be memoryless because the distribution of remaining lifetime of a disk that is currently operational is the same as the distribution of its lifetime from birth. More precisely, the probability of surviving for time $t+s$, given that a disk has survived for time s, is the same as the probability of surviving for time t from birth. This means that I can truncate the unobserved period from the data, or in other words, I can assume all disks that were put into service before 1989 actually went into service at the beginning of 1989. The entry labeled "truncated" in the exponential model of Table 4.5 shows the mean lifetime MLE and 95% confidence interval under this treatment. The strong agreement between results for this treatment and the extrapolation I have used in previous sections occurs because the disks in Table 4.2 are plausibly exponential, and the extrapolation procedure adequately estimates failures during the unobserved period according to data in the observed period.

A third and optimistic treatment for the unobserved period assumes that there were no failures before January 1989. Table 4.5 shows the exponential and Weibull lifetime models under this assumption in the entry labeled "optimistic." In comparison to the extrapolation I used in previous sections, this optimistic treatment has more effect on an exponential lifetime model than on a Weibull lifetime model. In the exponential model, it increases mean lifetime by 25% and widens confidence intervals by 115% and 43%. In the Weibull model, this optimistic assumption widens the confidence intervals only slightly and increases one-year reliabilities. The difference in the effects on exponential and Weibull models is attributable to the greater freedom in which to fit the data provided by the additional parameter in the Weibull model.

Although the extrapolations I have used to estimate failures in the unobserved period for the data in Table 4.2 are not rigorous, they do not change the plausibility of an exponential model for the lifetime of these disks and, in a strictly exponential lifetime model, they match the more rigorous truncation treatment.

4.5. Summary

In this chapter I examined the lifetimes of magnetic disks using a conventional model for disk lifetime distributions – an exponential random variable. This model is mathematically convenient and has been successfully used in a wide variety of lifetime models including magnetic disk lifetimes. A popular alternative lifetime model is the Weibull random variable. A special characteristic of the Weibull model is its ability to provide evidence against an exponential model; if there is little chance that data is modeled by a Weibull random variable with a shape parameter value of 1.0, then there is little chance that the data is modeled by an exponential random variable.

Summary data from large populations of high-performance disks suggests that the mean lifetime for disks is about 50,000 hours if these lifetimes have an exponential distribution. I also quoted evidence for an exponential distribution with a mean lifetime of 86,900 hours from a small sample of disks used in a popular minicomputer. Additionally, anecdotal evidence indicates that device wear-out did not significantly affect IBM 2314 magnetic disk drives even when these units were severely obsolete.

Disk reliability specifications have greatly increased in the past few years. Although the bulk of this change results from changes in the way that mean lifetimes are calculated, there was an acceleration in the underlying rate of improvement in disk reliability.

In most of this chapter, I explored an exponential model for disk lifetimes through the examination of data on 1350 samples from two 5¼-disk products. This data was collected by Thinking Machines Corporation between the first quarter of 1989 and the second quarter of 1990. There is reasonable evidence to indicate that the lifetimes of the more mature of these products can be modeled by an exponential distribution with a mean lifetime of over 200,000 hours. For the less mature of these products, there is evidence that an exponential random variable is too simplistic a model, although it cannot be ruled out. The less mature disk product, without customer burn-in testing, has a Weibull shape parameter between 0.59 and 1.04 and at least a 89% chance of surviving its first year (with 95% confidence). At a 70% confidence level, this data has a Weibull shape parameter between 0.68 and 0.92, and its disks have more than a 90% chance of surviving their first year. If lifetimes for this disk product are modeled with an exponential distribution, there is 95% confidence that the mean lifetime is between 58,000 and 115,000 hours with a best estimate of 80,000 hours.

Although I conclude that an exponential distribution for the lifetimes of these disks is plausible, a more careful examination of the Weibull model results indicates that both an exponential and a Weibull model do not have enough degrees of freedom to model the lifetimes of disks not subjected to customer burn-in testing. In this case two distinct failure rate models could be stitched together, or perhaps a three parameter model found, that would be more accurate.

CHAPTER 5

Reliability Modeling

Recent advances in computing speeds can be matched by the I/O performance afforded by parallelism in striped disk arrays. Arrays of small disks further utilize advances in the technology of the magnetic recording industry to provide cost-effective I/O systems based on disk striping. But because arrays of small disks contain many more components than do larger single disks, failure rates can be expected to rise. For most users, increased failure rates are incompatible with secondary storage systems because secondary storage is thought to be the stable part of a computer system – the part expected to survive periodic malfunctions.

In our information society, malfunctions in computational components threaten the process by which new information is generated. But losses suffered by long-term storage components are even more debilitating because these destroy assets. It is no surprise then that institutions often consider the reliability of long-term storage crucial to their operation. Unfortunately, increasing reliability usually also increases cost.

The goal of this chapter is to facilitate the cost-effective design of reliable secondary storage by developing analytic models of the reliability of redundant disk arrays. The models include a wide spectrum of disk array designs so that individual designers will be able to characterize the reliability of the system they want to build. I use Markov-model-solving software and simulation in this chapter largely to validate the models I present.

In this chapter I present four models for the reliability of redundant disk arrays that correct all single disk failures. The most fundamental model considers the effect of independent, random disk failures on an array's data lifetime. The lifetime of data in an array ends when a failed disk's data is lost. This first reliability model is based on a well-studied Markov model and yields a simple expression for reliability. A cost-effective method for improving reliability by maintaining a small number of on-line spare disks is addressed in a second, more complex model. It yields an analytic expression for reliability by solving separate submodels for data loss derived from spare-pool exhaustion and concurrent, independent disk-failures. A third model uses similar methods to address dependent disk failures induced by sharing interconnect, controller, cooling, and power-supply hardware (collectively called support hardware). Although N+1-parity protection only insures the correction of a single disk in a parity group, disk arrays can be organized so that each disk in a support-hardware group is contained in a distinct parity group. In this way, dependent disk failures are tolerable because they affect at most one disk per parity group. Finally, the fourth model bounds the reliability of disk arrays that incorporate on-line spare disks with dependent and independent disk failures. These bounds allow estimates of reliability with no on-line spares and with sufficient on-line spares to provide one- and two-spare, support-hardware groups. These four models show how disk arrays can provide high reliability with modest amounts of redundancy.

I use these models and simulations to explore the cost-reliability tradeoffs between arrays with different levels of redundancy. Traditionally, redundancy has been provided by full duplication, or mirroring, without on-line spares. Although this type of organization provides higher reliability than an N+1-parity organization of the same user capacity without spares, the addition of a few spares reverses this relationship. One of the most important results of this chapter, specific to the design of practical disk arrays, is that an N+1-parity disk array with a few spares can yield greater reliability at lower cost than traditional mirrored disk arrays.

Metric	IBM 3390	IBM 0661
Units	1	70
Formatted Data Capacity (MB)	22700	22400
Number of Actuators	12	70
Avg Access Time (msec)	19.7	19.8
Max I/Os/Sec/Box	609	3535
Track Transfer Rate (MB/sec)	15.3	118.3

Table 5.1: Comparison of Strawman Disk Array to IBM 3390. *An array of 70 IBM 0661 disks is used in this chapter to exemplify reliability models. This "strawman" was selected to match the capacity of an IBM 3390 disk subsystem, IBM's high-end disk product. This table shows that the strawman array has superior throughput and comparable response time. Relative price is not as clear, but, according to Table 3.3 in Chapter 3, 70 3½-inch disks will cost a disk array manufacturer about 22,400 MB × 2.5 $/MB = $56,000 whereas IBM's best customers must pay almost three times this price for an IBM 3390 and a portion of an IBM 3990 controller.*

Throughout this chapter differences in reliability models will be exemplified by their effects on an array of 70 3½-inch disks. Table 5.1 shows that this disk array is selected to match the capacity of an IBM 3390 disk subsystem with 70 IBM 0661 (Lightning) disks. Because the 3390 is IBM's newest, largest, and most expensive disk product, there is a lucrative market for a disk array that can exceed the performance and reliability of the IBM 3390 while matching its cost per megabyte.

Without redundancy, this example disk array unfortunately has virtually no chance of surviving three years without data loss because of the aggregate failure rate of its large number of components. With as little as 10% overhead for parity information, however, this disk array can be made about as reliable as a single disk. Then, if the failure of support hardware does not damage multiple disks simultaneously, the addition of a single on-line spare disk yields a mean time to loss of data that is about 10 times larger than a single disk. With two on-line spare disks, the mean time to loss of data in this disk array is marginally less than if it had an infinite number of on-line spare disks, specificly, about a factor of 20 times larger than a single disk. Even if this disk array is subject to dependent disk failures, an orthogonal arrangement of parity groups and support hardware groups and a single on-line spare disk yield about an 80% chance that data is not lost in 10 years. If this is not satisfactorily larger than the 56% chance that a single disk survives 10 years without data loss, raising the overhead for on-line spares to 10% allows failed support hardware to be replaced immediately and delivers a 98.7% chance of surviving 10 years without data loss.

5.1. Reliability Metric

The *Reliability*[1] of a system is defined for any target lifetime, t, as the probability that an individual system survives for time t given that it is initially operational [Siewiorek82 pp 7]:

$$R(t) = \text{Prob}(\,lifetime > t \mid initially\ fully\ operational\,)\,. \tag{5.1}$$

In stochastic terminology, $R(t) = 1-F(t)$, where F is the cumulative distribution function (CDF) of system lifetimes:

$$F(t) = \text{Prob}(\,lifetime < t \mid lifetime > 0\,) = 1 - R(t)\,. \tag{5.2}$$

In this work, *survival* means all user data is available or recoverable, so the reliability at time t is the proportion of systems that have not lost any user data in time t.

Because a function can be a cumbersome metric, reliability is frequently quoted as a simple probability with an implied time interval. For example, designers may be most interested in a system's one-year reliability ($R(1year)$), or, for the pessimistic among us, the probability that it will survive the duration of its warranty. Where I do not have a complete description of $R(t)$ or where its presentation involves too much data, I use estimates of the system's 1-, 3-, and 10-year reliabilities.

Perhaps the most commonly encountered measure of a product's reliability is its *Mean Time To Failure, MTTF*, or its *Mean Time Between Failures, MTBF*. As Chapter 3 shows, this metric does not give much information unless the lifetime distribution is known. Fortunately,

[1] The use of the term *reliability* as a mathematically defined metric and as an intuitive concept can lead to ambiguity. I have chosen to follow the former convention and avoid the latter use.

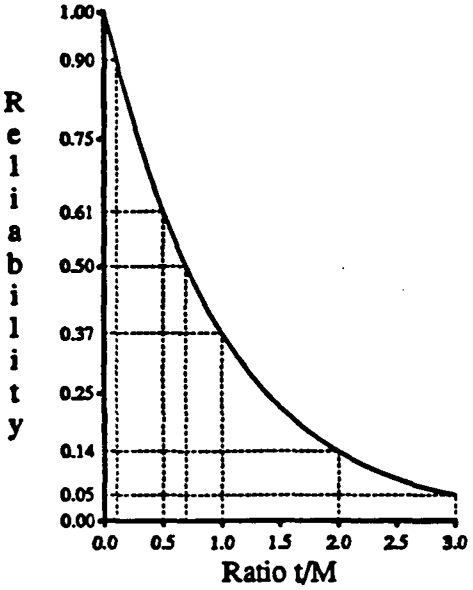


Figure 5.1: Exponential Reliability versus Ratio of Time to Mean Lifetime. *This figure shows the reliability of a system with exponential lifetimes with time expressed as a multiple of the system's mean lifetime, M. I have marked a few interesting points; the system has a 90% chance of surviving 0.1 M, a 61% chance of surviving 0.5 M, a 37% chance of surviving 1.0 M, a 14% chance of surviving 2.0 M, and a 5% chance of surviving 3.0 M. I have also marked the median lifetime, the time yielding a 50% chance of survival, which is 0.69 M.*

little strong evidence against an exponential model for disk lifetimes was found, so the *MTTF* of a disk drive may be assumed to imply a complete reliability function. The current chapter shows that under most conditions, the lifetimes of disk arrays have an approximately exponential distribution, so that the equivalent metric, *Mean Time To Data Loss, MTTDL,* is a complete description of a disk array's reliability. Because exponential lifetime distribution plays a prominent role in this chapter, I include a close examination of its characteristics.

Where lifetimes are distributed exponentially as random variables, $R(t)$ has an exceptionally simple form fully described by the product's mean lifetime, M:

$$R_{exp}(t) = e^{-t/M} . \qquad (5.3)$$

Figure 5.1 shows the reliability of a system with exponential lifetimes as a function of time, which is expressed as a fraction of the system's mean lifetime. There is a 10% chance that this system will have a lifetime less than one-tenth its mean lifetime, a 37% chance of surviving one mean lifetime, and only a 5% chance of surviving three mean lifetimes.

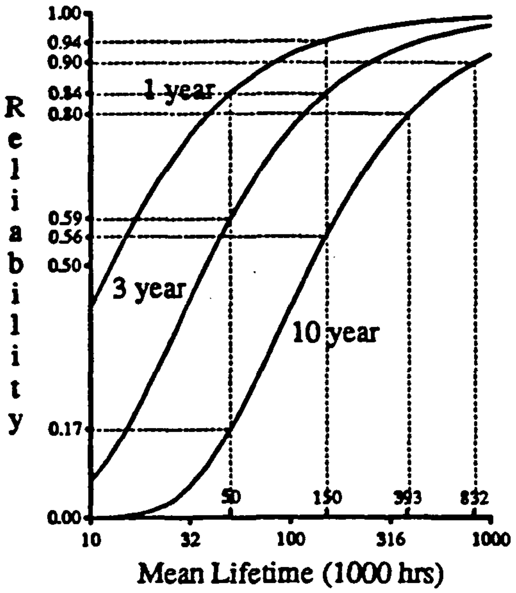


Figure 5.2: Exponential 1-, 3-, and 10-Year Reliabilities versus Mean Lifetime. *This figure shows the reliability of a system with exponential lifetimes over 1, 3, and 10 years as a function of the system's mean lifetime (in 1,000 hours, where there are 8,766 hours in a year). The x-axis scale is logarithmic. With a mean lifetime of 50,000 hours, the system has an 84% chance of surviving one year, a 59% chance of surviving three years, and a 17% chance of surviving 10 years. If the mean lifetime can be increased to 150,000 hours, the chance of surviving one year rises to 94%, the chance of surviving three years rises to 84%, and the chance of surviving 10 years rises to 56%. To achieve an 80% or 90% chance of surviving 10 years, the mean lifetime must exceed 390,000 hours or 830,000 hours, respectively.*

Figure 5.2 shows the reliability over 1, 3, and 10 years for a system with exponential lifetimes as a function of its mean lifetime. This figure underscores the attractiveness of extremely high mean lifetimes. Although it may seem silly to spend time and money changing a *MTTF* from 6 years (50,000 hours) to 45 years (390,000 hours) or 95 years (830,000 hours) because most products are obsolete in less than 10 years, these changes increase the probability of surviving the first 10 years from 0.17 to 0.80 or 0.90, respectively.

A powerful property of exponential lifetimes is their *age-independence*. For exponentially distributed lifetimes, the probability of surviving time t given that the system has survived time s, $s < t$, is exactly the probability of surviving time $t-s$. Thus, if the system is currently operational, its remaining lifetime has the same distribution as its initial lifetime, and its age is not important. Although this assertion is counter-intuitive for real systems, it provides the basis of many common reliability approximations. Specifically, age independence indicates that an exponential distribution's failure rate, $1/M$, is constant. (This is also discussed in Section 4.1 of Chapter 4.) Age independence leads to the simple approximation $R(t) = 1 - t/M$ as long as the time period, t, is small relative to the mean lifetime, M. This approximation underestimates $R(t)$ by less than 1% when $t < 0.13M$ and by less than 10% when $t < 0.39M$. For example, if a system's mean lifetime is greater than 5,500 hours, its one-month reliability is no more than 1% larger than $1 - (365.25 \times 24/12)/M = 1 - 730.5/M$.

Another useful way to look at this approximation is

$$\text{Prob("death" before } t) = 1 - R(t) \approx t/M \tag{5.4}$$

which means that doubling the mean lifetime, M, halves the chance of "death" in time periods, t, that are small relative to M. This approximation overestimates $1-R(t)$ by less than 1% when $t < 0.02M$ and by less than 10% when $t < 0.19M$. For example, if a system's mean lifetime is greater than 37,000 hours, the probability that it will "die" in one month is no more than 1% larger than $(365.25 \times 24/12)/M = 730.5/M$.

The failure rate of a lifetime with an exponential distribution is the reciprocal of mean lifetime, $1/M$. Failure rate is a metric often preferred over that of mean lifetime because of the $R(t) = 1 - t/M$ approximation. In this work I will use whichever of these is most intuitive to the matter at hand.

5.2. Related Work

The reliability of computer systems is a widely researched field. But because my goals in this chapter are specific to the reliability of disk arrays only, I will not attempt a complete review of it here. An excellent treatment of the field can be found in the book by Siewiorek and Swarz [Siewiorek82]. In addition to presenting design methodologies and detailed case studies, this book contains a practical discussion of frequently used mathematical techniques including the basic ones I employ in this chapter. Briefer treatments of the reliability of computer systems can be found in Avižienis's classic survey [Avižienis78] and Nelson's up-to-date overview [Nelson90]. More detailed understanding of the modeling mathematics can be found in a variety of textbooks [Bhat72, Ross85, Wolff89] and survey articles [Geist90].

One result important to understanding the reliability of disk arrays relates to the distribution of the time until failure of a system with redundant parts and dynamic repair. In this case, the system fails when too many components fail before repair can be completed. Arthurs and Stuck develop intuition for this distribution in a paper modeling the reliability of a machine with a single backup machine and a dedicated repairman[Arthurs81]. They show that the distribution of the time until both machines are concurrently being repaired approaches the exponential distribution as the probability that one machine will fail before the other is repaired approaches zero. Moreover, they show that this is true regardless of the distributions of the time until machine failure and of the time until repair is complete. Their result can be generalized to apply to any system experiencing short periods (e.g. repair) during which it is vulnerable to improbable events [Wolff91].

The result obtained by Arthurs and Stuck is important for my research because it is applicable to repairable, redundant disk arrays. These systems suffer infrequent failures and can be repaired within a small number of hours by replacing the failed component and recovering any affected data. With their result, I expect that the time until a disk array suffers a failure causing some data to be unrecoverable – the time until data loss – will be approximately distributed as an exponential random variable. This approximation improves when repair is made faster or failures are made less frequent. Although I will present evidence throughout this chapter that this expectation is borne out by my simulations, it should be remembered that my premise is not sensitive to the distributional assumptions I use.

The conventional method for constructing redundant disk systems is based on duplication [Katzman77]. The reliability of duplexed, or mirrored, systems and of the closely related

Triple-Modular-Redundant (TMR) systems have been well studied [Siewiorek82 pp 262]. These are both special cases of the disk-array reliability model that I examine in Section 5.4.1. Duplexed disks, however, double disk costs. But, as I discuss in Section 3.4.2, they can also improve performance. To extend these performance advantages, rather than to enhance reliability, some researchers have suggested disk systems with more than two copies of every disk [Bitton88, Matloff87] and, although costly, this is available in Digital Equipment Corporation's disk subsystem products [Bates89]. Because the cost of duplicating data is prohibitive for most systems, I will concentrate on the less expensive N+1-parity encoding for redundancy in disk arrays.

In an early paper on repairable, redundant disk arrays, Park and Balasubramanian present an optimistic estimate for the mean time until data is lost [Park86] – optimistic because their model underestimates the period of time that a disk array is vulnerable during a disk repair. Section 5.4.1 examines a more appropriate model for N+1-parity disk arrays that was first applied to disk arrays by Patterson, Gibson, and Katz [Patterson88]. Although Park and Balasubramanian discuss both failures in the hardware that supports disks and the inclusion of on-line spare disks, they do not model the effect these factors have on disk array reliability. Sections 5.5 and 5.6 in this chapter do, however, model the effects on reliability of each of these factors, respectively, and Section 5.7 examines their combined effects. These sections show that although failures in disk-support hardware can drastically reduce the reliability provided by redundant data, these effects can be substantially overcome. I have previously presented preliminary analysis of these factors [Gibson89c].

Ng has studied Section 5.4.1's model for reliability in a disk array and extensions for including on-line spare disks [Ng90]. By employing a Markov model simulation tool, he determined that there is little benefit from including more than one spare disk in a disk array of a single parity group of up to 32 disks. In Section 5.6, I present an analytic model that substantiates and extends his result.

5.3. Tools and Methods

In my quest to quantify reliability, I developed and used three different types of stochastic tools. The most general tool is an event-driven simulator I wrote called RELI. RELI explicitly generates failure events in a specified disk array from its installation until the first time a failed disk's data cannot be recovered. Each of the durations from installation until data loss is a sample lifetime of the specified disk array. RELI samples enough lifetimes to be able to estimate a reasonably narrow confidence interval for the expected lifetime of the specified disk array. RELI is described more fully in Appendix A.

The second tool I used was a software package called Sharpe that solves Markov models. (Sharpe is distributed by Duke University [Sahner86, Sahner87].) It generates a cumulative distribution function for array lifetimes from a completely specified Markov model. To provide results without the trouble of developing programs, my final tools are fully-parameterized analytic expressions. These expressions can be evaluated with simple programs or hand-held calculators.

The greatest advantage of simulation is that it allows complexity to be modeled realistically, although complex simulators are vulnerable to lurking bugs. Markov models are well-understood (and the more widely-used tool, Sharpe, can be expected to have fewer bugs), but the specification of a complex Markov model frequently requires many arguable approximations, and its specification task is nearly as difficult as coding a simulator. Furthermore, the solution of Markov models by a tool such as Sharpe requires all parameters to be fully specified, and it generates numeric results. For highly reliable designs, moreover, Sharpe's general solution technique suffers from numerical approximation problems, and it becomes necessary to resort to separate solutions for each point in time of interest. For these reasons I use Markov models mainly to verify simulation and analytic models.

In contrast to either Markov models or simulation models, analytic models offer simpler computation and greater intuitiveness. Unfortunately, they offer point estimates only and usually rely on arguable simplifications. A major goal of this chapter was the development of analytic models to obtain these computational and intuitional advantages, but I rely on more complex simulation models to provide information about distributions and to support assumptions in analytic models.

As this chapter proceeds, I use a general technique with which to compare models. First I construct a large number (thousands) of sets of the models' parameters. This defines the "design space" within which I desire agreement among my analytic, Markov, and simulation models. Then I randomly sample this design space for about 100 parameter sets. For this collection of parameter sets, I evaluate both (or all) models. If the models define a single value, I show a scatter plot of the relative differences between pairs. Where the models define a reliability function, $R(t)$, I evaluate the agreement between two curves with the root-mean-square (RMS) of the difference between curves. Because nearly all computer components are obsolete and replaced in less than 10 years, I evaluate only the root-mean-square difference over the first 10 years (87,660 hours). This gives an "average" absolute difference between the two reliability curves and, once again, I display the results of each root-mean-square comparison in a scatter plot

5.4. Independent Disk Failures

The simplest model for single-erasure-correcting arrays of redundant disks is based on a three-state Markov model with independent and exponential disk lifetimes and exponential repair durations. This is a specific application of a model that has been featured in stochastic-process textbooks and which exemplifies simple redundancy in a collection of identical, repairable equipment [Bhat72 pp 268-272]. Using the well-known solution for this model, I evaluate the reliability of a single parity group of disks. To extend my solution to a disk array containing multiple parity groups, I approximate the time until data is lost in a single parity group as an exponential random variable and evaluate the time until the first parity group losses data. This section ends with the application of these models to my strawman disk array.

5.4.1. Markov Model for a Single Parity Group

For a single-erasure-correcting group of $N+1$ disks, where disk lifetimes are exponential with mean $MTTF_{disk} = 1/\lambda$ and repairs are exponential with mean $MTTR_{disk} = 1/\mu$, Figure 5.3

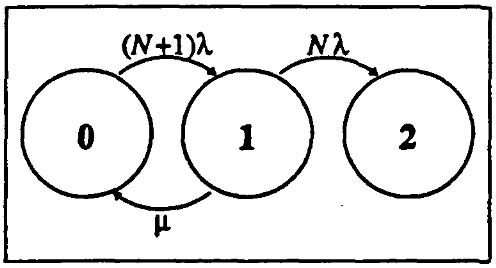


Figure 5.3: Data Loss Model for Independent Disk Failures in a Single Parity Group. *A single disk-array parity-group of $N+1$ disks can be modeled by a three-state Markov model if disk lifetimes are exponential with mean $MTTF_{disk} = 1/\lambda$ and disk repairs are exponential with mean $MTTR_{disk} = 1/\mu$. The states are labeled with the number of disk failures evidenced by the array. When there are no failures, the rate of failure is $N+1$ times the rate of an individual disk failure, λ. When there is one failure, the rate of repair is μ and the rate of a second failure is N/λ. Once there are two concurrent failures, data has been lost, and there are no more transitions.*

shows a three-state Markov model labeled by the number of concurrent failures. Using Laplace transforms on $P_n(t)$, the probability of being in state n at time t after originating in state 0 has been shown to be

$$P_0(t) = \frac{(N\lambda+\mu+\xi)e^{\xi t} - (N\lambda+\mu+\zeta)e^{\zeta t}}{\xi-\zeta},$$

$$P_1(t) = \frac{(N+1)\lambda}{\xi-\zeta}(e^{\xi t} - e^{\zeta t}),$$

$$P_2(t) = 1 - P_0(t) - P_1(t),$$

$$R(t) = P_0(t) + P_1(t) = \frac{\xi e^{\zeta t} - \zeta e^{\xi t}}{\xi-\zeta}, \tag{5.5}$$

where

$$\xi = \frac{-((2N+1)\lambda+\mu)+\sqrt{\lambda^2+\mu^2+2(2N+1)\lambda\mu}}{2},$$

and

$$\zeta = \frac{-((2N+1)\lambda+\mu)-\sqrt{\lambda^2+\mu^2+2(2N+1)\lambda\mu}}{2}.$$

The reliability, $R(t)$, is the probability that no data has been lost during time t. With this expression the mean time until a group suffers data loss, $MTTDL$, is

$$MTTDL = \int_{t=0}^{\infty} R(t)\,dt = \frac{(2N+1)\lambda+\mu}{N(N+1)\lambda^2}. \tag{5.6}$$

The mean time until a group suffers data loss is simply the expected time beginning in state 0 and ending on the transition into state 2 in Figure 5.3.

If $R(t)$ is not necessary, $MTTDL$ can be found more easily. Beginning in a given state i, the expected time until the first transition into a different state j can be expressed as

$$\text{E[state } i \text{ to state } j] = \text{E[time in state } i \text{ per visit]}$$
$$+\sum_{k\neq i} \text{P(transition state } i \text{ to state } k)\,\text{E[state } k \text{ to state } j] \tag{5.7}$$

where

$$\text{E[time in state } i \text{ per visit]} = \frac{1}{\sum \text{rates out of state } i}$$

and

$$\text{P(transition state } i \text{ to state } k) = \frac{\text{rate of transition to state } k}{\sum \text{rates out of state } i} .$$

The solution to this system of linear equations includes an expression for the expected time beginning in state 0 and ending on the transition into state 2, that is, for *MTTDL*. For the Markov model in Figure 5.3, this system of equations is

$$\text{E[state 0 to state 2]} = \frac{1}{(N+1)\lambda} + \frac{(N+1)\lambda}{(N+1)\lambda}\text{E[state 1 to state 2]}$$

$$\text{E[state 1 to state 2]} = \frac{1}{\mu+N\lambda} + \frac{\mu}{\mu+N\lambda}\text{E[state 0 to state 2]}$$

$$+ \frac{N\lambda}{\mu+N\lambda}\text{E[state 2 to state 2]}$$

$$\text{E[state 2 to state 2]} = 0 .$$

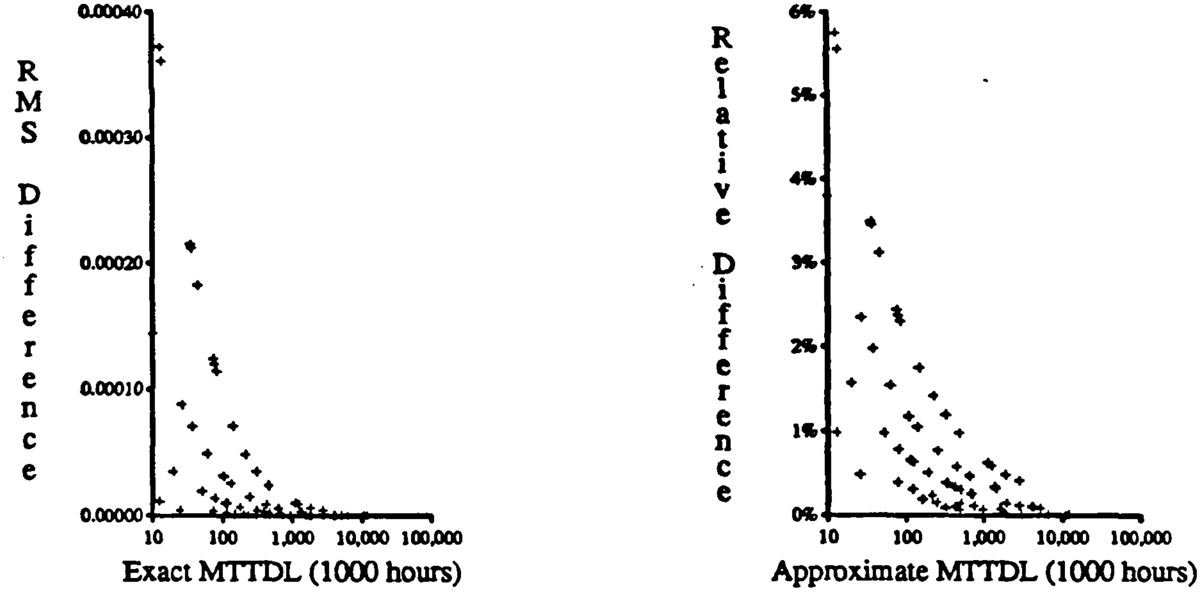
Solving, the above expression for *MTTDL* is, happily, rederived as

$$MTTDL = \text{E[state 0 to state 2]} = \frac{(2N+1)\lambda+\mu}{N(N+1)\lambda^2} .$$

The expressions for reliability, $R(t)$, and mean time to data loss, *MTTDL*, in this section are exact with respect to the Markov model in Figure 5.3. The methods used to derive these, particularly that used to derive *MTTDL*, will be used in later sections without displaying intermediate results explicitly.

5.4.1.1. Approximations and Simplifications

Although the reliability, $R(t)$, given in the previous section is exact, it is not computationally simple. However, the value of ξ is likely to be much larger than ζ because all variables are positive, and $\mu = 1/MTTR_{disk}$ is much larger than $\lambda = 1/MTTF_{disk}$. As a result, it is reasonable to assume that the reliability function, $R(t)$ would be well-approximated by an exponential function, $R_{Indep}(t) = e^{-t/MTTDL}$, with the same mean, *MTTDL*. Figure 5.4a compares $R(t)$ with such an exponential approximation over a collection of 94 parameter sets selected at random. For each parameter set, the root-mean-square of the difference between $R(t)$ and $R_{Indep}(t)$ intuitively represents the average distance between these two curves. For example, the largest root-mean-square difference in Figure 5.4a is less than 0.0004, so the difference between $R(t)$ and



Figures 5.4a and 5.4b: Single Parity Group Reliability versus Approximations. *Figure 5.4a, on the left, shows the root-mean-square difference between the exact reliability function for the Markov model of Figure 5.3 and an exponential approximation with the same mean. The root-mean-square difference is evaluated over the first 10 years of operation. There are 94 comparisons shown, one for each of 94 parameter sets selected at random. Figure 5.4b, on the right, shows the relative difference between the exact MTTDL and its simplified approximation $MTTDL_{Indep}$ over the same 94 parameter sets.*

$R_{Indep}(t)$ is no more than 0.0004 on average. Because this average difference is small, $R(t)$ can be well-approximated by an exponential function with the same mean.

As to the exact value of *MTTDL*, another approximation is appropriate. Because $(2N+1)\lambda = (2N+1)/MTTF_{disk}$ should be much less than $1/MTTR_{disk} = \mu$, the mean time until data is lost can be approximated as

$$MTTDL_{Indep} = \frac{\mu}{N(N+1)\lambda^2} = \frac{MTTF_{disk}{}^2}{N(N+1)MTTR_{disk}} . \qquad (5.8)$$

Figure 5.4b shows the difference between $MTTDL_{Indep}$ relative to *MTTDL* for the same 94 parameter sets used in Figure 5.4a. Because the approximation is pessimistic by less than 6% for all of these parameter sets, I will adopt this expression for the mean time until a single parity group loses data.

Together, these two approximations are

$$R_{Indep}(t) = e^{-t/MTTDL_{Indep}} . \qquad (5.9)$$

5.4.2. Multiple Parity Groups

In practice, a disk array is composed of more than a single parity group. If each group fails independently, the time until data loss in a multiple-group disk array is the time until the first component group fails. Given that the lifetime of a single-erasure-correcting group can be modeled as an exponential random variable, the lifetime of a disk array has the same distribution as the shortest lifetime of its component groups. Conveniently, one of the properties of exponential random variables is that the minimum of multiple exponential random variables is

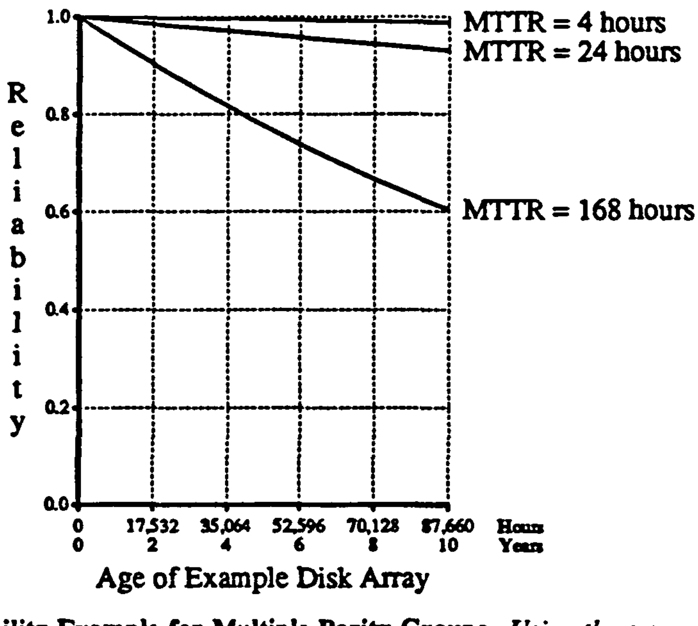


Figure 5.5: Reliability Example for Multiple Parity Groups. *Using the exponential approximation for multiple parity groups, this example shows the reliability of 70 data disks organized into seven parity groups over 10 years. All disks have average lifetimes of 150,000 hours. Separate curves are shown with 4, 24, and 168 hours as the average duration of disk repairs. With seven groups of 10 data disks each, there are 10% more disks for redundancy. The age of this example disk array is shown in hours, where there are 365.25×24=8766 hours in a year. Note that this x-axis, the age of an individual disk array, is different from the x-axis in Figure 5.2, the mean lifetime of a collection of disk arrays, which is also expressed in hours.*

also an exponential random variable whose rate is the sum of the component groups' rate. Hence, a disk array composed of G single-erasure-correcting groups each containing $N+1$ disks has

$$MTTDL_{Indep1} = \frac{(2N+1)\lambda+\mu}{GN(N+1)\lambda^2} = \frac{(2N+1)MTTF_{disk}MTTR_{disk}+MTTF_{disk}^2}{GN(N+1)MTTR_{disk}}, \qquad (5.10)$$

or

$$MTTDL_{Indep2} = \frac{\mu}{GN(N+1)\lambda^2} = \frac{MTTF_{disk}^2}{GN(N+1)MTTR_{disk}}, \qquad (5.11)$$

and

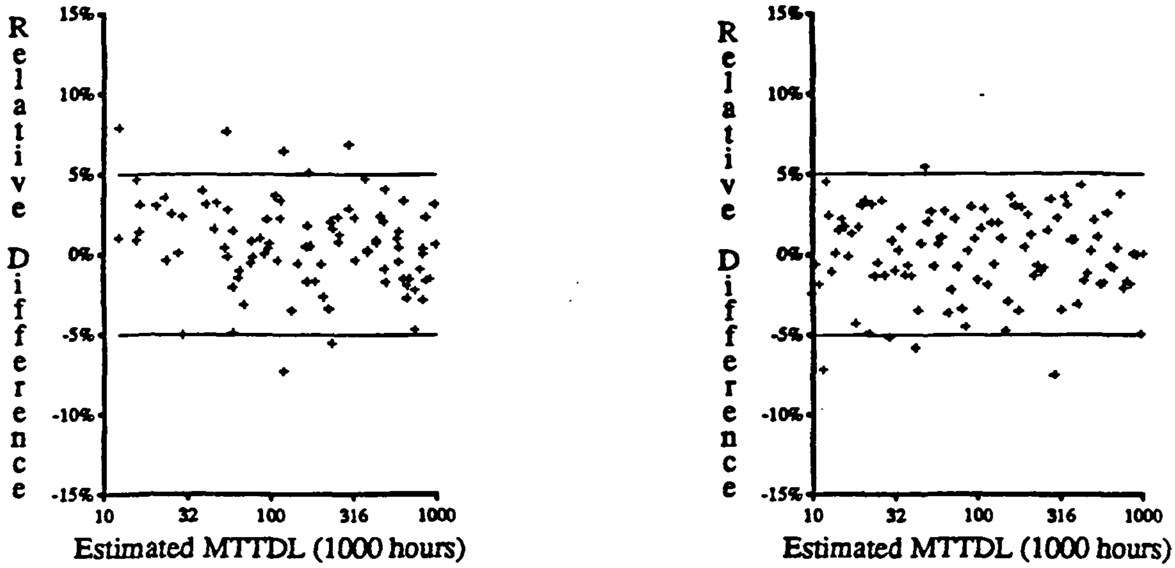
$$R_{Indep}(t) = e^{-t/MTTDL_{Indep}}. \qquad (5.12)$$

Figure 5.5 shows an example of these approximations for the reliability of multiple parity groups. Seven groups each with 10 data disks are protected by one parity disk per group. Each disk has a $MTTF_{disk}$ of 150,000 hours and three average repair durations, $MTTR_{disk}$, of 4 hours, 24 hours, and 168 hours. This example shows the advantage of fast repair; Section 5.6 presents a model for array reliability where repair times are accelerated by a pool of on-line spare disks.

5.4.3. Calibrating Consistency with Simulation

In order to evaluate the match between these equations and direct simulation of an array's lifetime, I have collected 94 simulated estimates for $MTTDL$ and matched them to the equations' estimates. Because the simulation is driven by exponential disk lifetimes and exponential repair times, this comparison, shown in Figure 5.6a, can be viewed as a consistency check between the simulation code and the Markov model. The 94 parameter sets are the same as those used in Figures 5.4a and 5.4b.

Because the simulation is stochastically driven, its estimates of $MTTDL$ will vary from the true $MTTDL$. To control this variation, the simulator collects independent lifetimes until the size of the 95% confidence interval on $MTTDL$ is no more than 10% as large its estimate of $MTTDL$. Figure 5.6b shows a graph similar to that in Figure 5.6a where I have replaced each



Figures 5.6a and 5.6b: Error in Estimated MTTDL Relative to Simulation. *On the left in Figure 5.6a, I show the error in MTTDL as estimated by $MTTDL_{Indep2}$ in Equation 5.11 relative to simulation. The horizontal lines in this figure border the region for the 95% confidence interval on MTTDL. The simulation parameters are selected randomly from a large group of parameter sets that yield an estimated MTTDL between 10,000 and 1,000,000 hours. On the right in Figure 5.6b, I show the same calculation for 100 samples taken from a population where, instead of simulation estimates, MTTDL is estimated by sampling an exponential distribution whose mean is equal to $MTTDL_{Indep}$. The similarity between these two figures provides evidence that the variability shown on the left arises from variability in simulation estimates.*

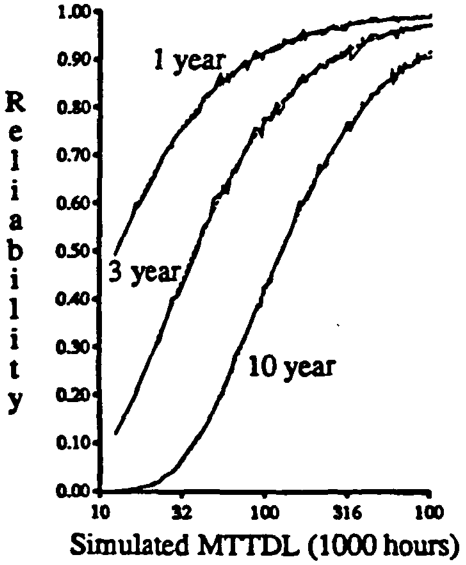


Figure 5.7: Estimated versus Simulated Reliability. *For each of the 94 simulations I plotted estimations of its 1-, 3-, and 10-year reliabilities. Solid lines are all 94 simulation estimations for a particular reliability duration. Dotted lines are the exponential estimations for 1-, 3-, and 10-year reliabilities.*

simulation lifetime with a sample from an exponential distribution whose mean equals the equation's estimate for *MTTDL*. The similarity between Figures 5.6a and 5.6b shows that variation in Figure 5.6a does not indicate error in the approximate estimate, $MTTDL_{Indep}$, but instead indicates variability in simulation's *MTTDL* estimate.

Given that simulated and calculated *MTTDL* agree to a degree comparable to the variation inherent in simulation, it remains to be shown that simulated lifetimes are exponentially distributed. Figure 5.7 shows the 1-, 3-, and 10-year reliabilities collected during simulation in comparison to corresponding estimations $R_{Indep}(t)$ at 1, 3, and 10 years. This figure shows that the simulation estimates for reliability are well-matched by an exponential approximation at least for these three points in time.

In another approach to demonstrating that simulated lifetimes can be suitably modeled by an exponential distribution, I have applied the well-known Pearson's chi-square (χ^2) *goodness-of-fit* test to the simulation data [Kirk90 pp 533, Lawless82 pp 441]. (An explanation of this procedure appears in Appendix C.) Because this test is actually designed to provide evidence against an assumed distribution, the correct way for me to report the results of this test is to say that I did not find strong evidence against the hypothesis that simulated disk array lifetimes are exponentially distributed which agrees with my expectations discussed in Section 5.2.

5.4.4. Implications for the Design of Disk Arrays

Figure 5.8 shows how 10-year reliability in arrays degrades with increasing numbers of disks and decreasing parity overhead. In this figure it is assumed that each disk has an exponential lifetime with a mean of 150,000 hours. Small arrays attain high 10-year reliabilities with small overhead for redundant disks. Large arrays are more reliable than a single disk at less than 50% overhead for redundant disks. By comparison, the 10-year reliability of 10 data disks with no redundancy is less than 0.003; that is, there is a 0.3% chance that 10 disks with exponential lifetimes having a mean of 150,000 hours will run for 10 years without any failure.

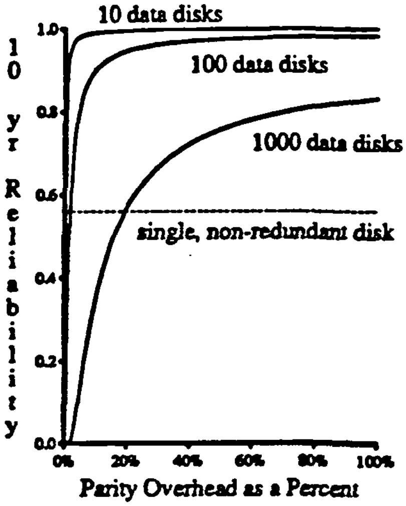


Figure 5.8: Reliability versus Redundancy Overhead. *This graph shows the trend for the 10-year reliability in a disk array suffering only independent exponential failures when each disk has a mean lifetime of 150,000 hours and it takes an average of 24 hours to repair each disk failure. Three array sizes are shown: 10, 100, and 1000 data disks per array. The size of a parity group, N+1, is related to the parity overhead expressed as a percent, O, by N = 100/O Recall from Figure 5.2 that a single disk with exponential lifetimes and a mean lifetime of 150,000 hours has a 10-year reliability of 0.56 (dotted line). Therefore, even a disk array with 1,000 data disks can be made more reliable than a single disk at about 20% overhead!*

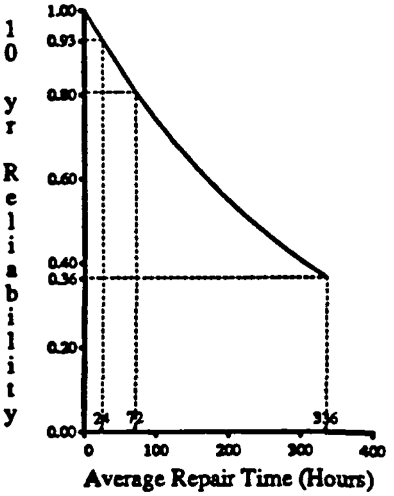


Figure 5.9: Strawman Reliability versus Repair Time. *This figure shows the probability that a disk array containing seven parity groups of 10 data and one parity disk survives 10 years without loss of data. Each disk has an exponential lifetime with a mean of 150,000 hours. This figure shows the dramatic effect on reliability of decreasing the average time it takes to repair a disk. Three specific, average repair times are highlighted. With an average disk-repair time of 2 weeks (336 hours), there is a 36% chance of surviving 10 years of operation. If the average disk-repair time is reduced to 3 days (72 hours) or 1 day (24 hours), then the chance of surviving 10 years rises to 80% and 93% respectively.*

There is virtually no chance of 100 or 1000 of these disks surviving 10 years without loss of data.

Figure 5.9 shows how the 10-year reliability of my stawman disk array example depends on the average disk repair time. I have partitioned 70 data disks of this array into seven groups (G=7) of 10 data disks and a parity disk (N=10). This organization has a low overhead cost amounting to 10% extra storage for redundancy. Because each of these disks has a mean lifetime of 150,000 hours, the mean time until data is lost is $MTTDL = 29{,}200{,}000/MTTR$. Figure 5.9 shows the effect of repair time on the 10-year reliability for this array. An average repair time of two weeks (336 hours) yields a 10-year reliability as low as 0.36, an average repair time of three days (72 hours) yields a much better (0.80) 10-year reliability, an average repair time of one day (24 hours) yields a 0.93 10-year reliability, and an average repair time of four hours yields a 0.99 10-year reliability. This example demonstrates the advantages of fast repairs, which can be achieved by maintaining off-line or on-line spare disks. Section 5.6 presents the effect of spare pool depletion on this estimate for reliability with fast repair.

5.5. Dependent Disk Failures

In the previous section I calculated the reliability of a disk array based on the optimistic assumption that all failures are independent. However, most I/O subsystems require support hardware that is shared by multiple disks. For example, power supplies, cabling, cooling, and controllers are often shared across multiple disks. Figure 5.10 shows an example of such support hardware and their failure rates for commonly available components that might be used in

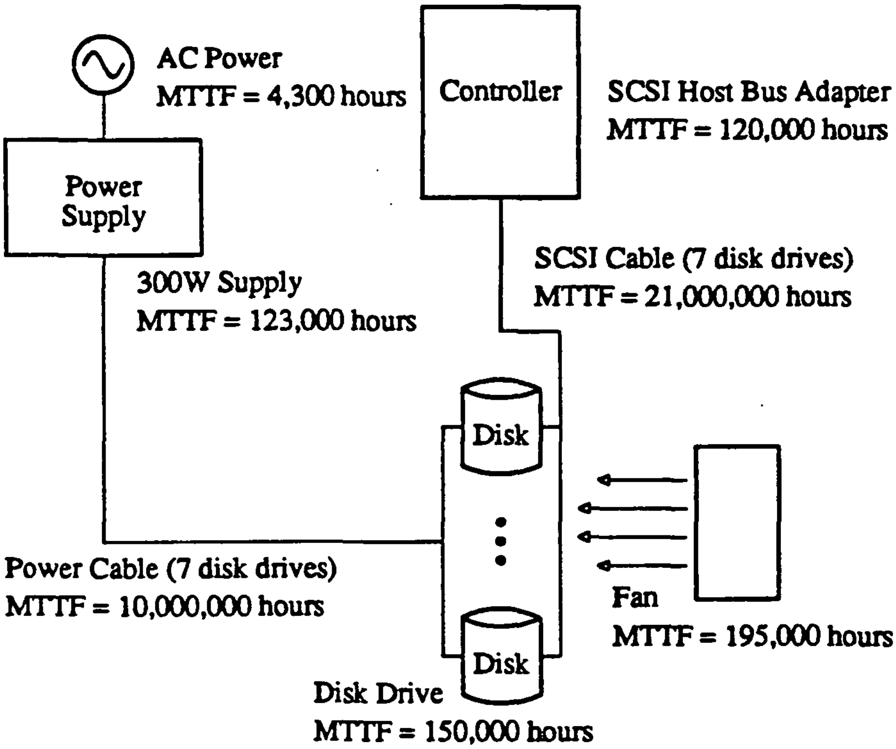


Figure 5.10: Example of Support Hardware Shared by Multiple Disks. *Disk subsystems are usually partitioned into* strings *that share datapath cabling and controllers. Although it is possible for different collections of disks to share power supplies and cooling support, this figure shows an example of a single string where cabling, controller, power, and cooling are all shared by the same disks. Sample component reliability figures are shown for a relatively low-cost, SCSI interface design [Schulze89]. Assuming that the support hardware's components have exponentially distributed lifetimes, the overall failure rate of the* non-disk *portion of a string is the sum of its components' failure rates. For high data reliability it is essential that the external power grid be isolated from the system. Using this data and assuming (possibly battery-backed-up) power supplies uneffected by power grid irregularities, the mean time to failure of a string is about 46,000 hours. This estimate can be easily increased by using more reliable and expensive parts or by incorporating redundant support hardware components, but its low value relative to mean disk lifetimes suggests that dependent disk failures can have a severe effect on reliability. In the rest of this chapter I use 150,000 hours as the mean time to string failure in my examples of a strawman disk array.*

a disk array. This figure assumes that the disks that share cabling also share power and cooling. I call such a configuration a *string*. Strings may fail if any of the support hardware fails, but, for purposes of this analysis, not because of disk failures. String failures can render many disks unavailable. In these cases multiple disks cannot be said to fail in an independent manner. Since RAID parity groups only guarantee recovery of a single disk failure, dependent disk failures may defeat my redundancy scheme.

String failures can severely limit the reliability of a disk array because each failure may render data unavailable for a sufficiently long period that this data is declared effectively lost.[2] Assuming that the time until a string fails is exponentially distributed, string failures cause the rate of data loss to be larger than that of the previous section by up to the rate of string failures $(G_{string} / MTTF_{string})$,

$$\frac{1}{MTTDL_{RAID}} = \frac{G\,(N+1)\,(N+2)\,MTTR_{disk}}{(MTTF_{disk})^2} + \frac{G_{string}}{MTTF_{string}} \qquad (5.13)$$

where G_{string} is the number of strings. String failures limit $MTTDL$ to a maximum of $MTTF_{string} / G_{string}$, regardless of redundancy among the disks.

The standard approach for limiting loss of data caused by string failures is to duplicate power, cooling, and controller components so that $MTTF_{string}$ is maximized. Although full duplication substantially improves the reliability of strings, it is an expensive solution that reduces the frequency of, but does not tolerate, string failures. A more powerful solution capitalizes on the larger number of identical components in a disk array.

With smaller and more numerous disks, an array is likely to contain a sufficient number of strings so that parity groups can be organized with no more than one disk from each group on

[2] There are a variety of reasons for assuming that string failures "erase" the data on their disks. First of all, some string failure, such as power failure, increase the probability that each disk will not restart when power is next applied. More significantly, applications such as on-line transaction processing may stand to lose many times the value of their computer systems when data is unavailable. However, in situations where string failures are non-threatening, the model in the next section will be more appropriate.

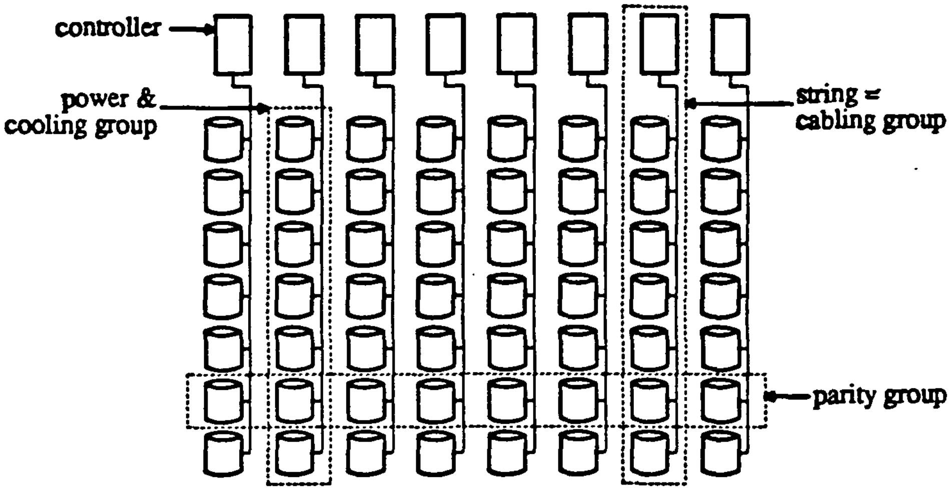


Figure 5.11: Orthogonal Organization of Parity and Support Hardware Groups. *By organizing support-hardware groups orthogonal to parity groups, the failure of a support-hardware group, or* string, *will destroy at most one disk in each parity group. Since parity-based redundancy schemes handle one failure per group, single-string failures are survivable.*

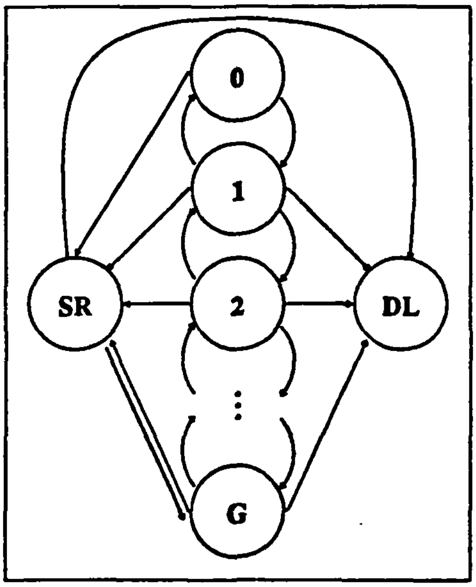


Figure 5.12: Markov Model for an Orthogonal Disk Array. *In this model G parity groups have G+3 states. In state* DL *data has been lost. In state* SR *a string is under repair. In each of the other states,* i, *i* ε *{0,1,...,G}, there are i different parity groups recovering the contents of a single failed disk. Transitions from state* i *to* i+1 *occur if a disk fails in a group that has no other currently failed disks; otherwise, data is lost. The reverse transition, from state* i+1 *to* i, *occurs when disk repair and recovery is completed. When a string fails the system enters state* SR *if all disks under repair are contained in the newly failed string; otherwise, data is lost. After a string is repaired. the contents of each of its disks may need to be recovered, so the model enters state* G.

any one string. As shown in Figure 5.11, this *orthogonal* organization of strings and parity groups guarantees that a single string failure can be endured as long as no other disk or string failure occurs before the string is repaired.

Repairing a failed string is more complex than repairing an independently failed disk. It involves a service visit or replacement operation for the component of the string that failed and then the recovery of multiple disks. This latter step may be necessary because the disks damaged or rendered unavailable by string failure have been replaced during repair of the string. It may also be necessary because the contents of these disks have been outdated by changes applied to parity disks instead of to the unavailable string-failed disks. Because of the multiple disk recovery step, the array may lose data if other components fail before the slowest disk recovery is complete.

5.5.1. Markov Model for Orthogonal Disk Array

Modeling the reliability of an orthogonal disk array with dependent failures is more complex than the independent disk-failures model of Section 5.4 Parity groups cannot be modeled individually because string failures cause all groups to experience a failure simultaneously. In Figure 5.12, I show a Markov model for a disk array with G groups organized orthogonally. This Markov model would yield complete solutions by the methods used to solve the simpler model of Section 5.4, but this cannot be done without assigning values to most parameters, and it requires messy inverse Laplace transformations. Instead, I have employed the Sharpe reliability and performance evaluation software package developed at Duke University [Sahner87] to evaluate the models of Figure 5.11. When Sharpe is given a model and values for all parameters, it will produce a *MTTDL*, a variance, and a *finite exponential polynomial* representation for the cumulative distribution function (CDF). The CDF evaluated at time t represents the probability that loss of data will occur before time t. Sharpe produces CDFs that can take the form

$$CDF(t) = 1 - \sum_j a_j t^{k_j} e^{b_j t} = 1 - R(t)$$

where $t \geq 0$ and j, a_j, k_j, and b_j are parameters estimated by Sharpe.

I have also applied Sharpe to a version of Figure 5.11 appropriate for my strawman disk array of seven parity groups with 10 data disks and a parity disk spread orthogonally over 11 strings of seven disks each. Once again mean disk lifetime, $MTTF_{disk}$, is 150,000 hours. Although Figure 5.10 suggests that mean string lifetime, $MTTR_{string}$, might be as low as 50,000 hours, I will optimistically assume that string components of higher quality lead to a mean string lifetime of 150,000 hours. For repair processes, I use an average disk repair time of 24 hours, $MTTR_{disk}$, and an average string repair time of 72 hours, $MTTR_{string}$. I further assume that string repair is a more complex operation – more likely to require a qualified repairperson's

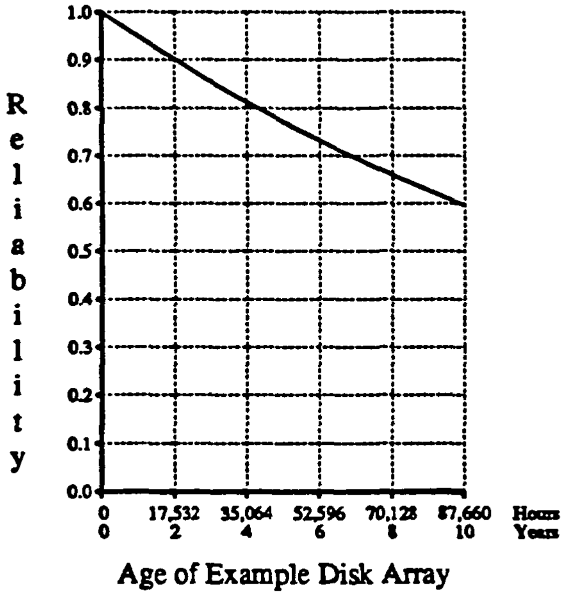


Figure 5.13: Reliability for Strawman with an Orthogonal Organization. *This figure shows the probability that my strawman disk array will survive its first 10 years with an orthogonal organization. This reliability function is derived from Figure 5.11's Markov model by Sharpe. My strawman disk array has seven parity groups of 10 data disks and a parity disk each. Disk lifetimes and repair times are exponential with means of 150,000 hours and 24 hours, respectively. String lifetimes and repair times are exponential with means of 150,000 hours and 72 hours, respectively. An exponential approximation with the same mean is in such close agreement that the two lines overlap.*

visit. For this example, Sharpe reports a *MTTDL* of 186,900 hours and

$$CDF(t) = 1 - (\ 1.0003e^{-5.3523\times 10^{-4}t} - 3.0048\times 10^{-4}e^{-0.014606t} - 1.9945\times 10^{-5}e^{-0.041650t}$$

$$- 2.6833\times 10^{-6}e^{-0.083938t} + 1.1164\times 10^{-6}e^{-0.12526t} - 4.0830\times 10^{-7}e^{-0.16746t}$$

$$+ 1.1291\times 10^{-7}e^{-0.20904t} - 1.9427\times 10^{-8}e^{-0.25093t} + 1.5765\times 10^{-9}e^{-0.29272t}\ ).$$

Figure 5.13 shows this example's reliability, $R(t) = 1 - CDF(t)$, over 10 years. An exponential approximation with the same mean, $R_{exp}(t) = e^{-t/186,900}$, differs from $R(t)$ by less than 0.04% over this range. Because this difference is so small, it is reasonable to expect that the models in Figure 5.11 will lend themselves to exponential approximations. To provide stronger evidence for this, I selected a random sample of 99 parameter sets constrained to yield an estimated *MTTDL* of between 10,000 and 1,000,000 hours and solved each with Sharpe. Because of Sharpe's limits on the number of states in a Markov model, 12 parameter sets could not be evaluated. For each of the remaining 87, I evaluated the root-mean-square difference between Sharpe's estimated $R(t)$ and an exponential approximation, $R_{exp}(t)$, with the same mean over

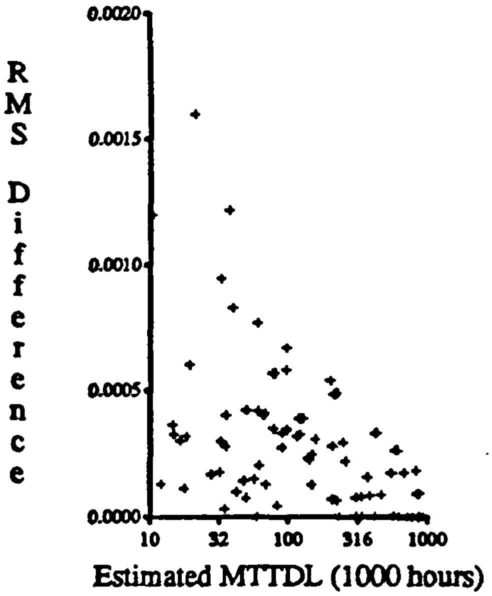
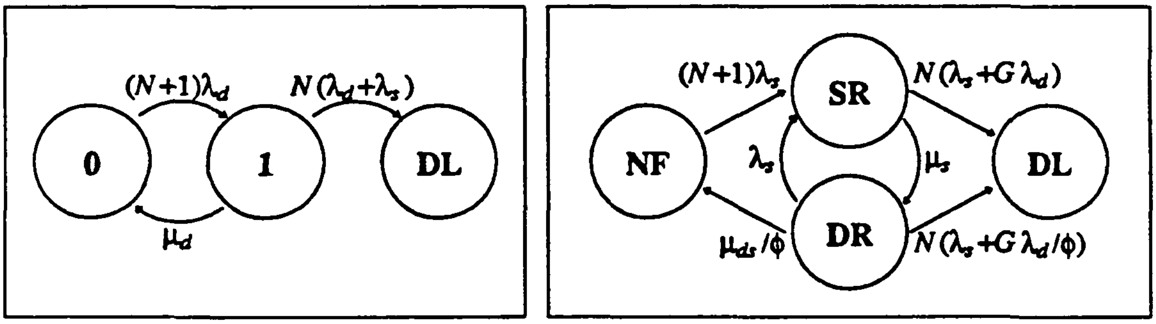


Figure 5.14: Exponential Estimate versus Markov Solution for Reliability. *This figure shows the root-mean-square difference between the reliability estimated by Sharpe, $R(t)$, and an exponential approximation, $R_{exp}(t)$, for 87 parameter sets selected at random and constrained to have MTTDL of between 10,000 and 1,000,000 hours. The parameter set with the largest root-mean-square difference has a maximum difference between Markov and an exponentially-approximated reliability of only 3% over the first 10 years.*

the first 10 years. Figure 5.14 shows that the largest root-mean-square difference is less than 0.002; the maximum difference between Sharpe's $R(t)$ and an exponential approximation, $R_{exp}(t)$, for the parameter set with the largest root-mean-square difference is 3%. This evidence supports an exponential approximation for array lifetimes modeled by Figure 5.12 provided that a close approximation of *MTTDL* is available. The next section presents an estimate for *MTTDL*.

5.5.2. Estimating Mean Array Lifetime



Figures 5.15a and 5.15b: Submodels for Data Loss in Orthogonal Disk Arrays. *The two sources of data loss in Orthogonal disk arrays are failures during a disk repair and failures during a string repair. The ambitious reader could derive the results presented in this section by applying the method of Section 5.4.1 for estimating MTTDL to each of these Markov models. Figure 5.15a, on the left, shows the sub-model for data loss in a single parity group caused by a second failure during a (non-string-failed) disk repair. Each of the $N+1$ disks in a parity group fails independently with $MTTF_{disk} = 1/\lambda_d$ and is repaired with $MTTR_{disk} = 1/\mu_d$. While a disk is being repaired, the failure of any of the other N disks or their strings causes data loss. (This model is the same as the model of Figure 5.3 with different transition rates.) Figure 5.15b, on the right, shows the sub-model for data loss caused by a second failure during a string repair. Each of the $N+1$ strings in the array fails independently with $MTTF_{string} = 1/\lambda_s$ and is repaired with $MTTR_{string} = 1/\mu_s$. During the repair of a string, the failure of any of the other N strings or the remaining GN disks will cause data loss. Once a string is repaired, the data on its disks is recovered either because the repaired string has new disks or because user data has been updated for the repaired string's disks using the array's parity disks. The average recovery time of a disk after a string repair, $MTTR_{disk-recovery} = 1/\mu_{dr}$, may be less than the average disk repair time, $MTTD_{disk} = 1/\mu_d$, because the string repair process is likely to include necessary disk replacements. While the disks of a recently-repaired string are recovering, the renewed failure of the same string reinitiates string repair. To avoid data loss after a string repair all G disks must complete recovery; therefore, the transition rate to state* NF *is the reciprocal of the expected maximum of G disk repairs, $MTTR_{Gdisks} = \phi/\mu_{dr}$, where $\phi = 1/1+1/2+...+1/G$. The factor ϕ also modifies the rate of data loss during string-induced disk recovery to account for the reduced vulnerability to other disk failures as indvidual recoveries are completed.*

One approach to estimating *MTTDL* begins by recognizing that failures, particularly closely-spaced double failures, are rare in real disk arrays. Where failure events are rare, I assume that different types of data loss are mutually exclusive and sum their rates. There are two sources of data loss in an orthogonal disk array: component failures during the repair of a disk and component failures during the repair of a string. Figures 5.15a and 5.15b describe sub-models for these two sources of data loss, respectively. Considering each of these cases separately, I apply the methods given in Section 5.4 for obtaining a mean lifetime without a complete solution for the reliability function. The contribution to the overall rate at which data is lost arising for vulnerabilities initiated by a disk failure is

$$G \times \frac{N(N+1)(1/MTTF_{disk}+1/MTTF_{string})/MTTF_{disk}}{1/MTTR_{disk}+(2N+1)/MTTF_{disk}+N/MTTF_{string}} . \tag{5.14}$$

Add to this the contribution arising from vulnerabilities initiated by a string failure,

$$\frac{1-1/((1+N(\varepsilon_{ss}+G\varepsilon_{sd}))(1+(N+1)\phi\varepsilon_{ds}\alpha_{Rd}+GN\varepsilon_{dd}\alpha_{Rd})-\phi\varepsilon_{ds}\alpha_{Rd})}{\dfrac{MTTF_{string}}{N+1}+\dfrac{1+MTTR_{string}}{1+N(\varepsilon_{ss}+G\varepsilon_{sd})-\phi\varepsilon_{ds}/(\alpha_{Rd}+(N+1)\phi\varepsilon_{ds}+GN\varepsilon_{dd})}} \tag{5.15}$$

$$\text{where } \alpha_F=\frac{MTTF_{disk}}{MTTF_{string}},\ \alpha_R=\frac{MTTR_{disk}}{MTTR_{string}},\ \alpha_{Rd}=\frac{MTTR_{disk}}{MTTR_{disk-recovery}},$$

$$\varepsilon_{dd}=\frac{MTTR_{disk}}{MTTF_{disk}},\ \varepsilon_{ss}=\frac{MTTR_{string}}{MTTF_{string}},\ \varepsilon_{sd}=\frac{MTTR_{string}}{MTTF_{disk}} \text{ and } \varepsilon_{ds}=\frac{MTTR_{disk}}{MTTF_{string}} .$$

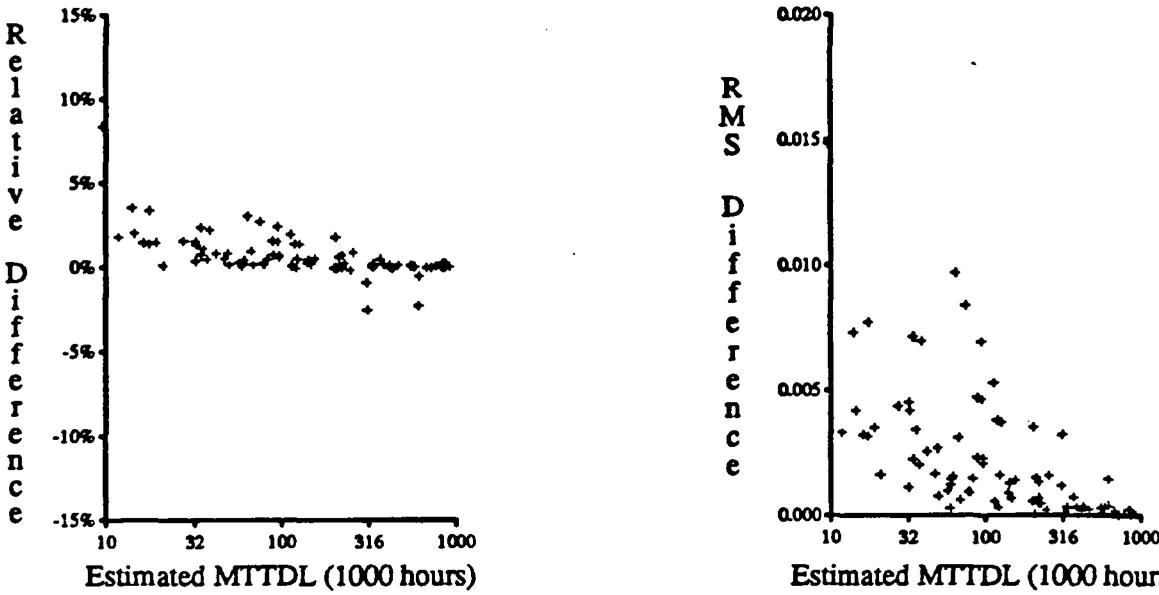
Summing these contributions and inverting this sum, the mean time to data loss, $MTTDL_{Ortho}$, is

$$\frac{\dfrac{MTTF_{disk}{}^2}{GN(N+1)MTTR_{disk}}}{\dfrac{1+\alpha_F}{1+(2N+1)\varepsilon_{dd}+N\varepsilon_{ds}}+\alpha_F\dfrac{1+\alpha_F\phi/G+(1+\alpha_F/G)(\alpha_R/\alpha_{Rd}+GN\varepsilon_{sd}+(N+1)\phi\varepsilon_{ss})}{((N+1)/MTTF_{string}+(2N+1)\varepsilon_{ss}+GN\varepsilon_{sd})\Psi(N+1)+\Psi(N)}}$$

$$\text{where } \Psi(g)=\alpha_{Rd}+GN\varepsilon_{dd}+g\phi\varepsilon_{ds}$$

$$\text{and } \phi=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{G} . \tag{5.16}$$

Figure 5.16a contrasts $MTTDL_{Ortho}$ given in Equation 5.16 against its error relative to a Sharpe evaluation of Figure 5.12. Differences in the *MTTDL* estimate are no larger than 9% across the 87 randomly-selected parameter sets that Sharpe was able to evaluate. In fact, there is only one estimate with more than 4% error and it has the unlikely combination of a 72-hour



Figures 5.16a and 5.16b: Orthogonal Estimated MTTDL versus Markov Solution. *Using a Sharpe evaluation of Figure 5.12 as the correct value for MTTDL, Figure 5.16a shows the relative error introduced when both sources of data loss are assumed to be independent, individually modeled using the methods of Section 4.4, and merged by summing their rates. From a random sample of 99 parameter sets constrained to yield estimated MTTDL between 10,000 and 1,000,000 hours, I show the 87 parameter sets that Sharpe was able to evaluate (12 parameter sets exceeded Sharpe's limits on states in a Markov model). Figure 5.16b shows the root-mean-square difference over 10 years between Sharpe's estimate of $R(t)$ and the exponential estimate, $R_{Ortho}(t)$, which uses the MTTDL estimated by the Equation 5.16. The increase in RMS values from Figure 5.14 to those in this figure, both of which use the same set of test cases, shows the effect of differences between estimations of MTTDL given in Figure 5.16a.*

disk-repair time, and a two-hour string-repair time, with an estimated disk-array lifetime of about 10,000 hours.

Using $MTTDL_{Ortho}$ as an estimate of the correct $MTTDL$ and following Figures 5.13 and 5.14 by assuming that time-to-data loss is an exponential random variable, reliability can be modeled as

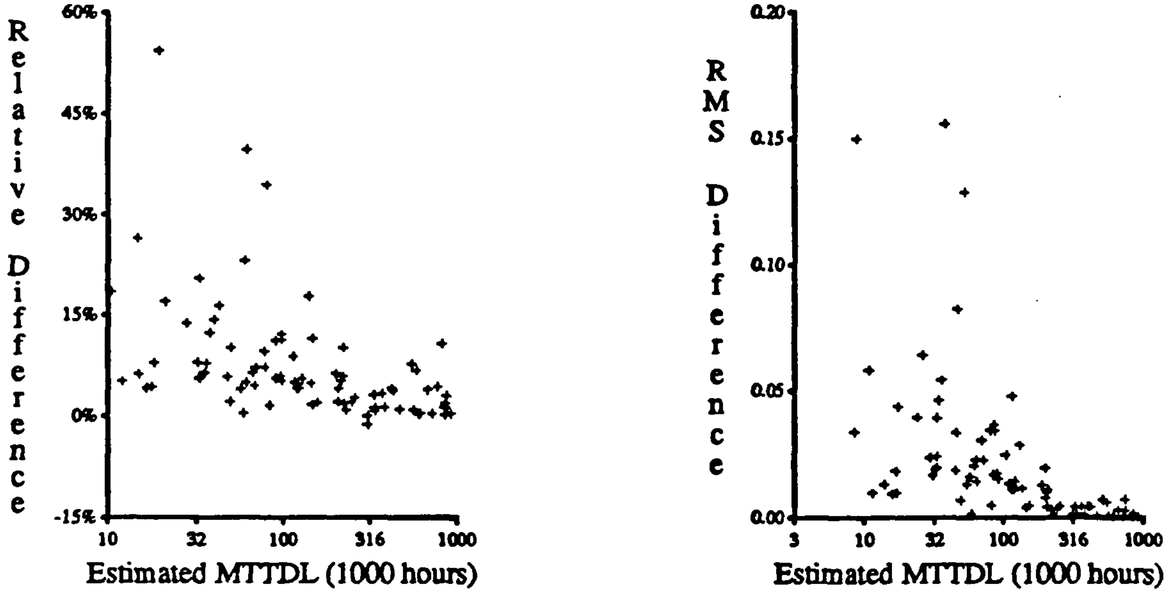
$$R_{Ortho}(t) = e^{-t \,/\, MTTDL_{Ortho}} \,. \tag{5.17}$$

Figure 5.16b shows the root-mean-square differences between Sharpe's estimate for $R(t)$ and its exponential approximation, $R_{Ortho}(t)$. In contrast to Figure 5.14, this figure has much larger RMS values for the same set of 87 parameter sets. Because both of these figures compare solutions of the same Markov models to exponential approximations, this difference in RMS values is the manifestation of the differences between $MTTDL_{Ortho}$ and Sharpe's estimate for $MTTDL$. Appendix B demonstrates the effect of errors in $MTTDL$ estimates on exponential reliability functions.

Although Equation 5.16 is accurate, it is certainly not simple! The values of ε_{dd}, ε_{ss}, ε_{sd}, and ε_{ds} are expected to be small because they are ratios of a mean repair duration to a mean time until failure. If each ε is assumed to be zero, $MTTDL_{Ortho}$ simplifies to

$$MTTDL'_{Ortho} = \frac{\dfrac{(MTTF_{disk})^2}{G\ (N+1)(N+2)MTTR_{disk}}}{(\,1+\alpha_F(1+\dfrac{1}{\alpha_{Rd}}+\dfrac{1}{\alpha_R})+\dfrac{{\alpha_F}^2}{G}(\dfrac{\phi}{\alpha_{Rd}}+\dfrac{1}{\alpha_R})\,)}\,. \tag{5.18}$$

As can be seen by comparing Figure 5.17a to Figure 5.16a, this expression has a substantially larger relative error. While in some cases the simplicity of this expression makes it preferrable



Figures 5.17a and 5.17b: Simplified Orthogonal MTTDL Estimate versus Markov Solution. *Letting each ε go to zero, Figure 5.17a shows that the relative errors of Figure 5.16a increase significantly. About one-third of the estimated MTTDL is more than 10% in error relative to the Markov model of Figure 5.12. Figure 5.17b shows the root-mean-square differences in corresponding reliability functions over 10 years. In contrast to Figure 5.16b, the much larger RMS values in this figure reflect the increased error in estimated MTTDL. This figure's x-axis begins at 3,000 hours, lower than the x-axis on Figure 5.16b, because the large errors in estimated MTTDL shift points around. Because this error is so pronounced, I will not use this simplified estimate for MTTDL in any other part of this chapter.*

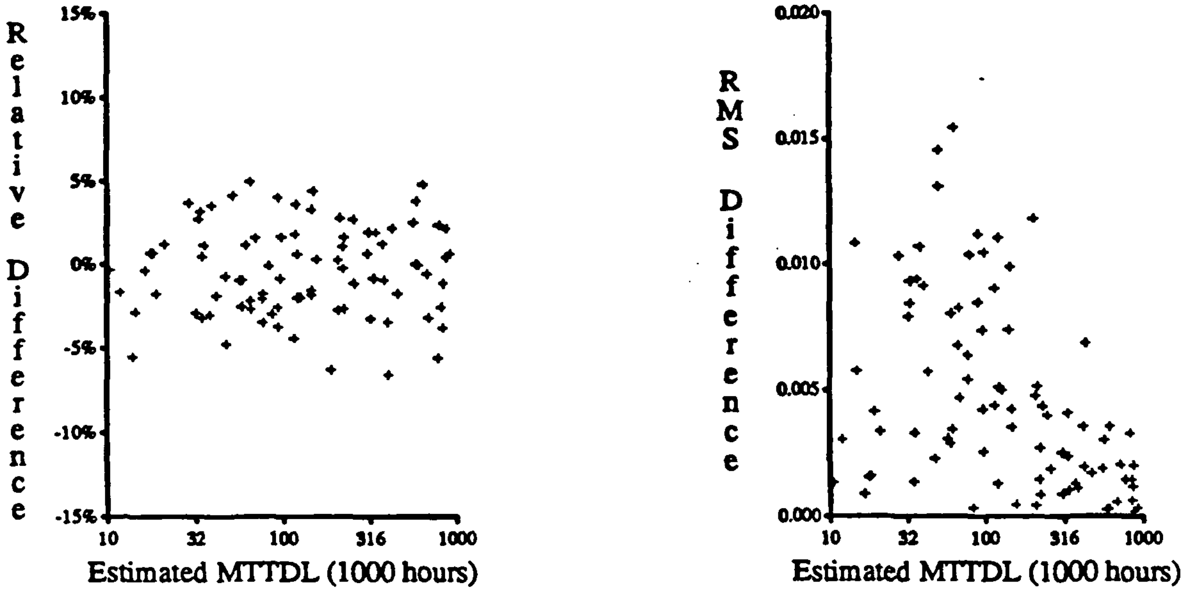
to Equation 5.16, I will not use this estimate further. The well-versed reader may note that an earlier version of this expression [Schulze89, Gibson89c] neglected to differentiate the disk-repair time after a string repair from the isolated, independent disk-repair time.

Now that I have demonstrated a good match between Sharpe's solution of the Markov model in Figure 5.12 and the $R_{Ortho}(t)$ estimation, I turn my attention to the agreement between simulation and estimation by first examining their respective estimates for mean lifetime. Figure 5.18a shows the relative error between simulated and Markov-modeled estimates for *MTTDL*. Because simulation provides an estimate of *MTTDL* whose 95% confidence interval is ± *5%* of *MTTDL*, errors of ± *5%* are to be expected. Figure 5.6b was presented in Section 5.4.3 to show the type of error I expected between a mean lifetime estimated by simulation and its true mean lifetime. The similarlity of these two figures indicates a high degree of agreement between simulated and Markov-modeled estimates for *MTTDL*.

To round out the comparison between simulation and estimation, I calculate their root-mean-square difference. Figure 5.18b shows this difference between a Markov-modeled reliability function and an exponential approximation with the mean lifetime estimated by simulation. In contrast to Figure 5.16b, this figure shows that an exponential approximation for reliability with the simulated mean is as good a match to the Markov-modeled reliability as an exponential approximation for reliability with the $MTTDL_{Ortho}$ estimate for its mean.

Figure 5.14 shows that the Markov model yields exponential reliability functions for the 87 parameter sets tested. Consequently, I expect simulated lifetimes to also have an approximately exponential distribution. Following the methods described in Appendix C, I again applied Pearson's chi-square, goodness-of-fit test. As in Section 5.4.3, this test does not yield strong evidence against the hypothesis that simulated disk-array lifetimes are exponentially distributed.

More positive evidence that simulated disk-array lifetimes have an exponential distribution is found in Figure 5.19. This figure shows simulation statistics estimating the 1-, 3-, and



Figures 5.18a and 5.18b: Simulated Orthogonal MTTDL versus Markov Solution. *Figure 5.18a shows the relative error in MTTDL between a simulated estimate and Sharpe's Markov solution. Figure 5.18b shows the corresponding root-mean-square differences between $R(t)$ and $R_{exp}(t)$ with the simulation's estimate for MTTDL.*

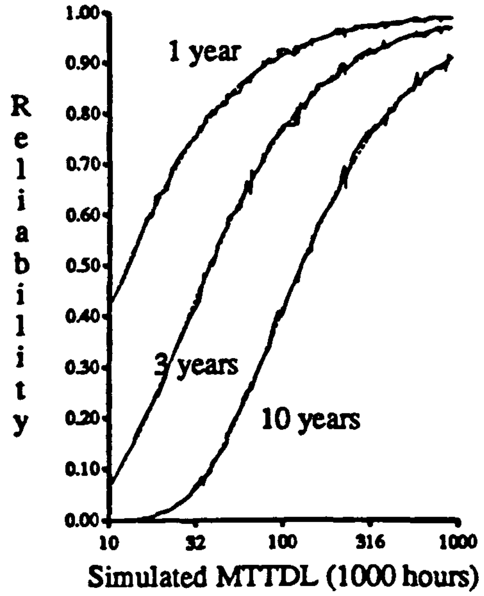


Figure 5.19: Simulated 1-, 3-, and 10-year Reliability. *This figure shows the 1-, 3-, and 10-year reliabilities computed during simulation against the MTTDL computed for the same simulations. For the purpose of comparison, the dotted and largely obscured lines show the 1-, 3-, and 10-year reliabilities for lifetimes that are exponentially distributed. The close match of these curves is good evidence that simulated disk-array lifetimes are exponentially distributed.*

10-year reliabilities of each of the 87 parameter sets used in previous figures beginning with Figure 5.14. These simulated estimates are plotted against their simulated $\underline{MTTDL}$. Also plotted with dotted lines are the 1-, 3-, and 10-year reliabilities of an exponential distribution. The fact that these dotted lines are nearly obscured constitutes strong evidence that simulated disk-array lifetimes are exponentially distributed.

5.5.3. Separating Disk Repair into Replacement Delivery and Recovery

This section demonstrates that the expression for $MTTDL_{Ortho}$, the mean lifetime of an orthogonal disk array, can be adapted for a more realistic, non-exponential disk-repair time composed of two parts. The first part of a disk repair process is a fixed delivery time at the end of which the failed disk is replaced.[3] Following this replacement, the second part of disk repair is an exponential disk-recovery time. For example, if a particular parameter set in the last section had an average disk-repair time of 24 hours, I might allocate 20 hours to a fixed-length delivery time and then set the average disk-recovery time to four hours. For each of these 87 parameter sets, I simulate a repair process whose replacement delivery time is 10%, 33%, 66%, 90%, and 99% of the total repair time. In each case, during a string repair all affected disks are replaced so that disk repair after string repair does not involve any replacement-delivery time. The simulator also assumes that if a disk fails while another failed disk's replacement is awaiting delivery, the second replacement can be added to the existing order and arrive with the first replacement disk. This is a good model for replacements delivered by a repairperson because service personnel are likely to bring a few extra disks when they visit customers. This optimization causes the average replacement-delivery time experienced by a failed disk repair process to be shorter than the actual fixed-length delivery time.

[3] Calling a fixed-length delivery time more realistic is perhaps a poor choice of words; in fact delivery does not take the same amount of time on each occasion, but it is substantially less variable than an exponential random variable. Recovery, on the other hand, will sometimes occur while the disk array is idle, and other times, it will occur during peak user load. If user requests are given priority, then recovery time will be highly variable, a much better match to an exponential random variable.

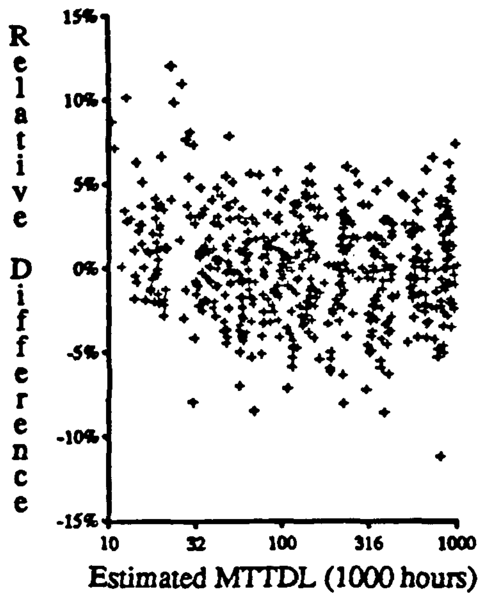


Figure 5.20: Partitioning Disk Repair into Replacement and Recovery. *This figure shows the agreement between simulation's estimate for MTTDL and $MTTDL_{Ortho}$ when disk repair is split into replacement delivery and disk recovery. For each of the 87 sample parameter sets, simulations have been run with replacement delivery time set to 0%, 10%, 33%, 66%, 90%, and 99% of the average disk-repair time used in previous sections. Because simulation satisfies all outstanding replacements at the end of a fixed-length delivery period, the average delivery time experienced by a failed disk is less than the actual delivery period.*

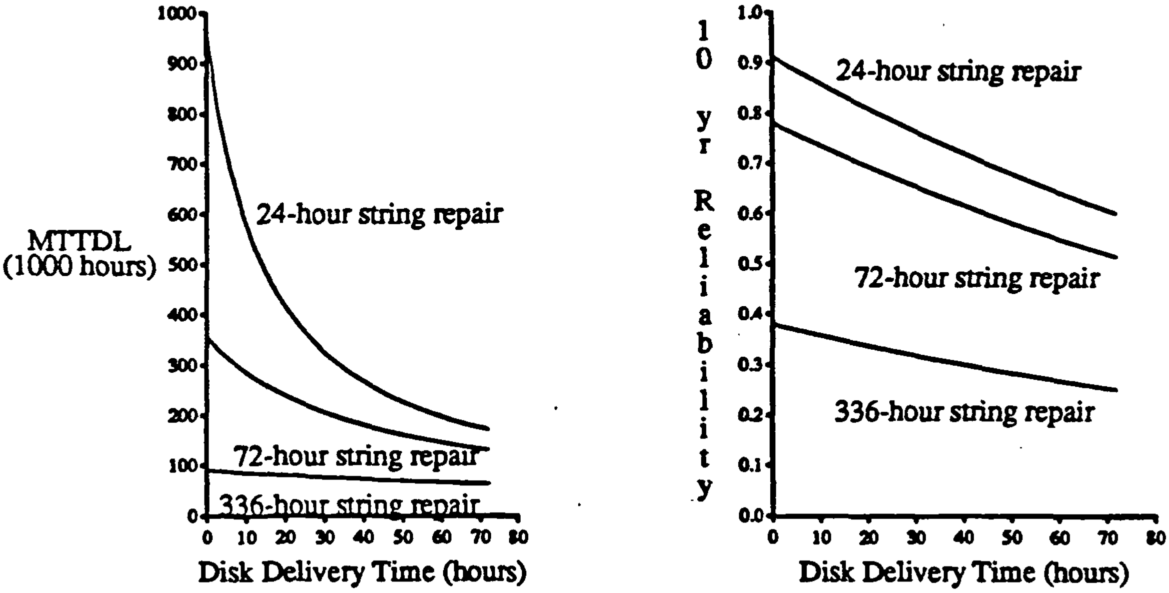
To account for replacement-delivery times that are shortened because disk failures happen close together, I examine each replacement delivery process. Most disk failures initiate an order for a replacement disk because there is no outstanding order. During the fixed-length period, D, until a replacement arrives, each of the other $G(N+1)-1$ disks may fail. Because each other disk fails independently with probability $1-e^{-D/MTTF_{disk}}$, the expected number of additional failures is $(G(N+1)-1)(1-e^{-D/MTTF_{disk}})$. If the number of disks in the array, $G(N+1)$, is large relative to the number of disks expected to fail during a delivery, then subsequent failures form a Poisson process, and their arrivals are uniformly distributed over the delivery period [Ross83 pp 37]. For this case the average time a failed disk waits until it is replaced is half the actual fixed-length delivery time. Including the delivery-initiating failure, the average delivery time is

$$\text{Average delivery time} = \frac{\text{Expected total delivery time}}{\text{Expected total failures}}$$

$$= \frac{D + (G(N+1)-1)(1-e^{-D/MTTF_{disk}})D/2}{1 + (G(N+1)-1)(1-e^{-D/MTTF_{disk}})} . \tag{5.19}$$

To incorporate these calculations into my estimate, $MTTDL_{Ortho}$, given in Equation 5.16, I set $MTTR_{disk-recovery}$ to the mean disk-recovery time and set $MTTR_{disk}$ to the average delivery time plus the mean disk-recovery time. Figure 5.20 shows the agreement between $MTTDL_{Ortho}$ and the simulated estimate of *MTTDL* for the 87 parameter sets at each of five partitions between replacement and recovery time. Although there is wider variation between simulated and estimated *MTTDL* than in Figure 5.18a, the vast majority of comparisons differ by less than ±5%.

5.5.4. Implications for the Design of Disk Arrays



Figures 5.21a and 5.21b: Repair Durations in Orthogonal Strawman. *The MTTDL and 10-year reliability of my strawman disk array are sensitive to their repair durations. Figure 5.21a shows MTTDL and Figure 5.21b shows 10-year reliability as they are effected by both average string-repair time and replacement-disk delivery time. The strawman disk array has an orthogonal organization of seven parity groups of 10 data disks plus a parity disk. Disks and strings both have mean lifetimes of 150,000 hours. Average disk-recovery time, excluding replacement-delivery time, is one hour.*

Applying the orthogonal estimate for the mean lifetime of a disk array to my strawman example, I show the importance of fast string repair and fast disk delivery in Figures 5.21a and 5.21b. The strawman disk array has an orthogonal organization with seven groups of ten data disks, a parity disk, and no on-line spare disks. Each disk has a mean lifetime of 150,000 hours and a mean recovery time of one hour. Each string has a mean lifetime of 150,000 hours and induces all disks on it to recover after its repair is complete.

Figures 5.21a and 5.21b demonstrate two significant characteristics of orthogonal disk arrays. First, string repair as slow as two weeks does not provide reliability at least as good as a single disk. Second, even if string repair is relatively fast, the delivery time of replacement disks must still be minimized to achieve high reliability. Fast repair processes can be expensive if they call for immediate attention of qualified service personnel. A small pool of on-line spare disks can effectively provide much faster repair without immediate attention from service personnel by reducing disk-delivery time to zero in most cases. The next section estimates the effect of a pool of on-line spares on the reliability of a disk array when string failures do not affect the integrity of disks' data. Then Section 5.7 estimates the reliability of a disk array that has on-line spares when string failures do affect data integrity.

5.6. Independent Disk Failures with On-line Spares

Figures 5.21a and 5.21b above suggest that on-line spare disks can significantly improve *MTTDL* and reliability. In this section I reduce average disk-repair time by employing a pool of on-line spare disks, which is not a continuation of the model in the last section because I do not include the effects of dependent disk failures. The combined effects of dependent disk failures and on-line spare disks are addressed in Section 5.7.

On-line spares reduce average disk-repair time because a failed disk can be replaced with a spare in the time it takes to change the software mappings for the location of the failed-disk's data. In this way disk-repair time is just the time it takes to recover a disk's contents by reading all other disks in a group, computing the exclusive-or, and writing these values to the recovering disk. Because there is no immediate need for a person to insert a new disk to replace a failed one, not only is repair and recovery time reduced, but opportunities for human error are eliminated. But, because on-line spares increase the cost of a disk array, their number will be limited, and disk failures will occasionally be exposed to longer periods of repair when the spare pool is exhausted.

My disk-array simulator can explicitly maintain a pool of spare disks. It uses a *threshold* parameter to decide when to issue an order for disks to replace spares now acting in place of recently failed disks. It also delivers enough disks to completely replenish the spare pool whenever a replacement order is filled.

5.6.1. Estimating Mean Array Lifetime

In all disk arrays protected with N+1 parity, data will be lost whenever two disks in one group fail before the first has been repaired. The state of the spare pool affects the repair rate, however. While there are spares, repair is fast and loss of data is unlikely, but while there are no spares, repair is slow and loss of data is much more likely. A complete Markov model for this involves a state for each combination of the number of groups recovering and the number of spares available. Such a model has many transition rates, each of which offers an opportunity for modeling error. Since closely-spaced double-disk failures and spare-pool depletion are both likely to be rare events of relatively short duration, I expect the distribution of time-to-data loss to be roughly exponential. With this expectation I can model reliability by estimating $MTTDL$. Since sources of data loss are rare, I estimate the reciprocal of $MTTDL$ by separately modeling each source of data loss and summing their rates of loss:

$$\frac{1}{MTTDL_{IndepSpares}} = \text{Independent disk–failure data–loss rate}$$
$$+ \text{Spares–exhausted data–loss rate}. \qquad (5.20)$$

The data-loss rate while on-line spares are available is given Equation 5.10 in Section 5.4.2 with $MTTR_{disk}$ set to the average time it takes to remap and recover a failed disk onto a

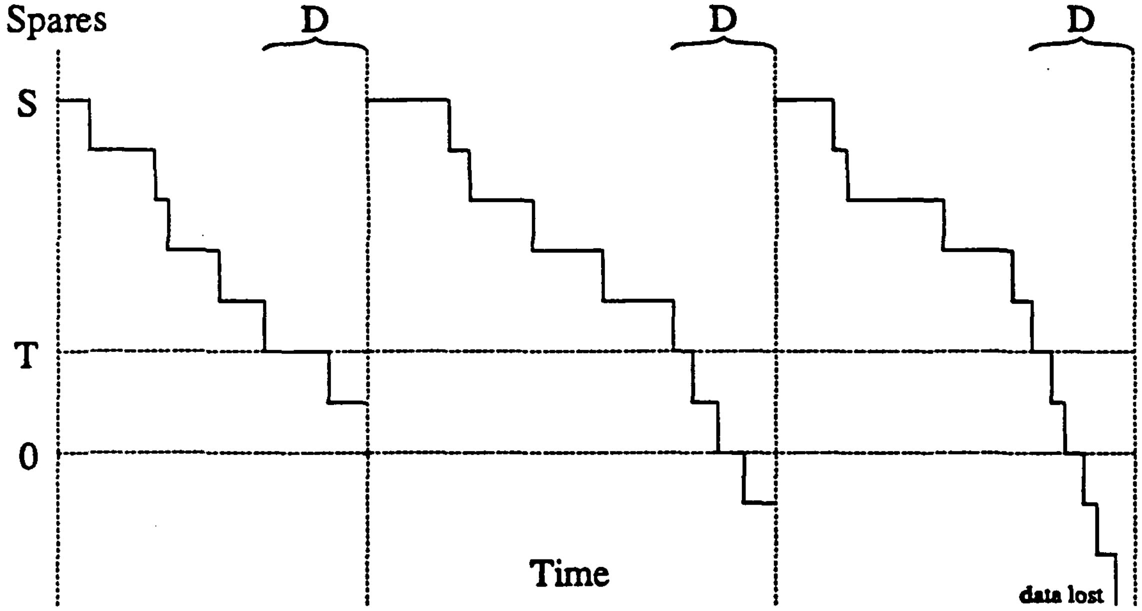


Figure 5.22: Example of Spare-Pool Depletion and Data Loss. *As disks fail the number of spares decreases from its maximum, S, to a threshold, T, at which time an order for more spares is issued. Orders take time D to be filled and result in a completely refilled spare pool. Until the order is filled disks may continue to fail. Additional failures consume the remaining T spares and then expose the array to a high level of vulnerability. This figure shows three opportunities for data loss during three successive orders. The first does not empty the spare pool, the second empties the spare pool then suffers and survives one additional failure, and the third suffers data loss because too many failures occur before the order is delivered.*

spare one, $MTTR_{disk-recovery}$. To estimate the data-loss rate arising because the spare pool periodically empties, I treat each period between the refilling of the spare pool as an independent opportunity to lose data. Figure 5.22 shows a sequence of examples where three opportunities for losing data occur during orders. The time until data loss caused by spare-pool exhaustion will be a geometric random variable with mean equal to the average time between spare-pool refilling divided by the probability of data loss during the delivery of an order.

$$\text{Spares–exhausted data–loss rate} = \frac{\text{Probability of data loss per order}}{\text{Average time between filled orders}} \tag{5.21}$$

Putting these terms together, the mean lifetime of a disk array suffering only independent disk failures and augmented with an on-line spare pool is $MTTDL_{IndepSpares}$:

$$MTTDL_{IndepSpares} = \tag{5.22}$$

$$\frac{1}{\dfrac{GN(N+1)MTTR_{disk}}{((2N+1)MTTR_{disk}+MTTF_{disk})MTTF_{disk}} + \dfrac{\text{P(data loss per order)}}{\text{Average time between filled orders}}},$$

where $MTTR_{disk}$ is the mean time for disk recovery excluding replacement-disk delivery time.

To compute the probability of data loss per order, I need the probability that at least one group will have suffered two or more failures in a disk array containing G parity groups of $N+1$ disks that has run out of spares and has suffered an additional q failures. For this calculation I neglect data losses caused by conflicts with recovering disks; that is, I assume failed disks that obtain an on-line spare are instantly recovered. With this model, when $q < 2$ data cannot be lost, and when $q > G$ data must be lost. When $2 \leq q \leq G$ the probability of data loss is 1 minus the probability that all failures will fall in distinct groups. The number of ways to assign q failures to distinct groups is the product of the number of ways of selecting q different groups and the number of ways of assigning q failures, one to a group. Then the probability that all failures will fall into distinct groups is the number of ways to assign q failures to different groups divided by the total number of ways of assigning q failures in $G(N+1)$ disks.

P(data loss | 2 ≤ q ≤ G unspared failures) = (5.23)

$$1-\frac{\binom{G}{q}(N+1)^q}{\binom{G(N+1)}{q}} = 1-\prod_{i=0}^{q-1}\frac{(G-i)(N+1)}{G(N+1)-i}$$

To derive the probability of data loss while an order for replacement disks is outstanding, I condition on the number of disk failures that occur while an order is being delivered:

$$\text{P(data loss per order)} = \sum_{q=0}^{G(N+1)+T} \text{P(data loss if } q \text{ failures in order)} \times \text{P(} q \text{ failures in order)}$$

$$= \sum_{q=0}^{G(N+1)+T} \text{P(data loss } | \ q \text{ failures)} \times \text{P(} q \text{ failures} < D \text{)}$$

Because data cannot be lost until the first $T+1$ disks have failed, and if the summation index, q, is changed to exclude the first T failures, this expression becomes:

$$= \sum_{q=T+2}^{G(N+1)+T} \text{P(data loss } | \ q \text{ failures)} \times \text{P(} q \text{ failures} < D \text{)}$$

$$= \sum_{q=2}^{G(N+1)} \text{P(data loss } | \ T+q \text{ failures)} \times \text{P(} T+q \text{ failures} < D \text{)}$$

Moreover, after the first $T+G+1$ disks have failed data must be lost, so this expression becomes:

$$= \sum_{q=2}^{G} \text{P(data loss } | \ q \text{ unspared failures)} \times \text{P(} T+q \text{ failures} < D \text{)}$$

$$+ \sum_{q=G+1}^{G(N+1)} \text{P(} T+q \text{ failures} < D \text{)}$$

$$= \sum_{q=2}^{G} \text{P(} T+q \text{ failures} < D \text{)} \times \left[1-\prod_{i=0}^{q-1}\frac{(G-i)(N+1)}{G(N+1)-i}\right]$$

$$+ \sum_{q=G+1}^{G(N+1)} \text{P(} T+q \text{ failures} < D \text{)}. \qquad (5.24)$$

To derive the probability of $T+1$ disk failures during the delivery of replacement disks, I use the fact that all operational disks have identical distributions for their remaining lifetimes. In particular, because each disk has the same failure rate, they all have the same probability of failing during the delivery of an order,

$$p = \text{P(each disk fails in order)} = 1 - e^{-\lambda D} \qquad (5.25)$$

where $\lambda = 1/MTTF_{disk}$. This means that the probability of exactly q failures in a fixed-length delivery time has a binomial distribution,

$$\mathrm{P}(\,q \text{ failures from } Q \text{ disks}\,) = \begin{bmatrix} Q \\ q \end{bmatrix} p^q (1-p)^{Q-q}$$

$$= \left[\prod_{j=1}^{q} \frac{Q-j-1}{j}\right] (1-e^{-\lambda D})^q \, e^{-\lambda D (Q-q)} . \qquad (5.26)$$

Because there are $G(N+1)+T$ disks operational when an order to replenish the spare pool is issued, the probability of data loss during an order becomes[4]

$$\mathrm{P}(\text{ data loss per order }) = \sum_{q=2}^{G} \begin{bmatrix} G(N+1)+T \\ T+q \end{bmatrix} (1-e^{-\lambda D})^{T+q} e^{-\lambda D (G(N+1)-q)} \left[1 - \prod_{i=0}^{q-1} \frac{(G-i)(N+1)}{G(N+1)-i} \right]$$

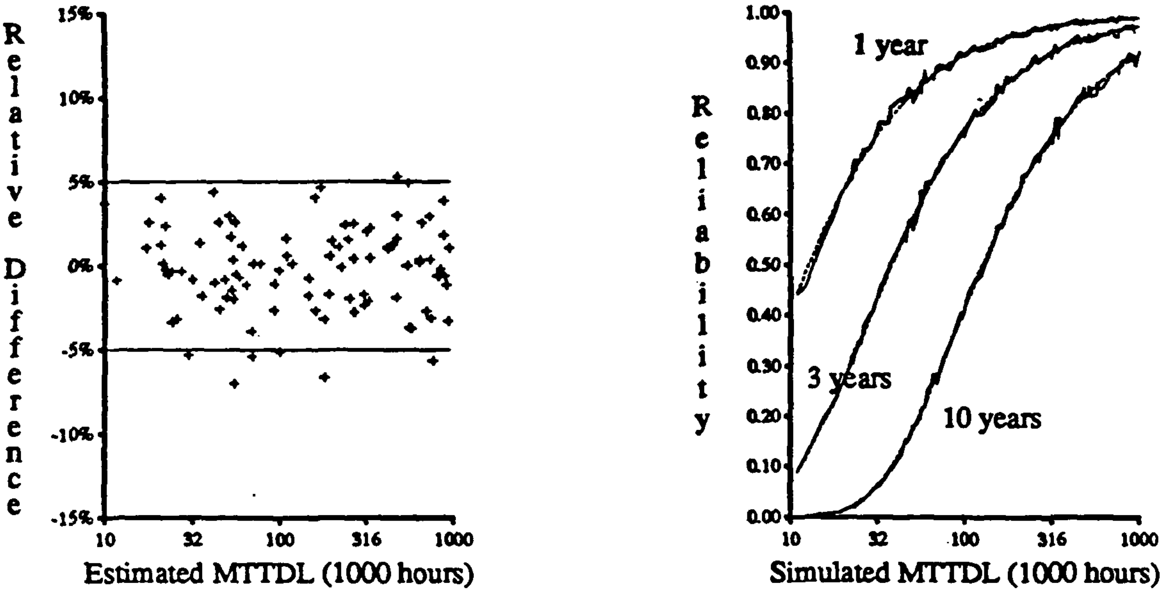
$$+ \sum_{q=G+1}^{G(N+1)} \begin{bmatrix} G(N+1)+T \\ T+q \end{bmatrix} (1-e^{-\lambda D})^{T+q} e^{-\lambda D (G(N+1)-q)} . \quad (5.27)$$

Finally, the average time between successive refillings of the spare pool is the time it takes to deliver replacement disks plus the expected time for the failure of the number of disks required to diminish the spare pool to its threshold. This latter is the expected time until the first $S-T$ disks have failed beginning with $G(N+1)+S$ operational disks.

$$\text{Average time between filled orders} = D + MTTF_{disk} \sum_{j=G(N+1)+T+1}^{G(N+1)+S} \frac{1}{j} \qquad (5.28)$$

To evaluate this estimate for the mean lifetime of a disk array that does not suffer string failures but does have an on-line spare pool, I selected 100 parameter sets at random from a large collection constrained to estimate mean lifetime between 10,000 and 1,000,000 hours. For each of these parameter sets, I simulated an estimate for mean time until data is lost so that I could compare this number to the $MTTDL_{IndepSpares}$ estimate. Figure 5.23a shows the result of that comparison. As in previous sections, some difference arises from the inherent variability in the simulation. Although more of the differences are outside the ± 5% bracket than in Figure 5.18a, these parameter sets are still well-modeled by the $MTTDL_{IndepSpares}$ estimate.

[4] This expression assumes $S > 0$. However, setting $S = 0$ and $T = -1$ yields an approximation to Equation 5.11 in Section 5.4.2 modified to separate replacement-disk delivery and disk recovery, as is done in Section 5.4.3.



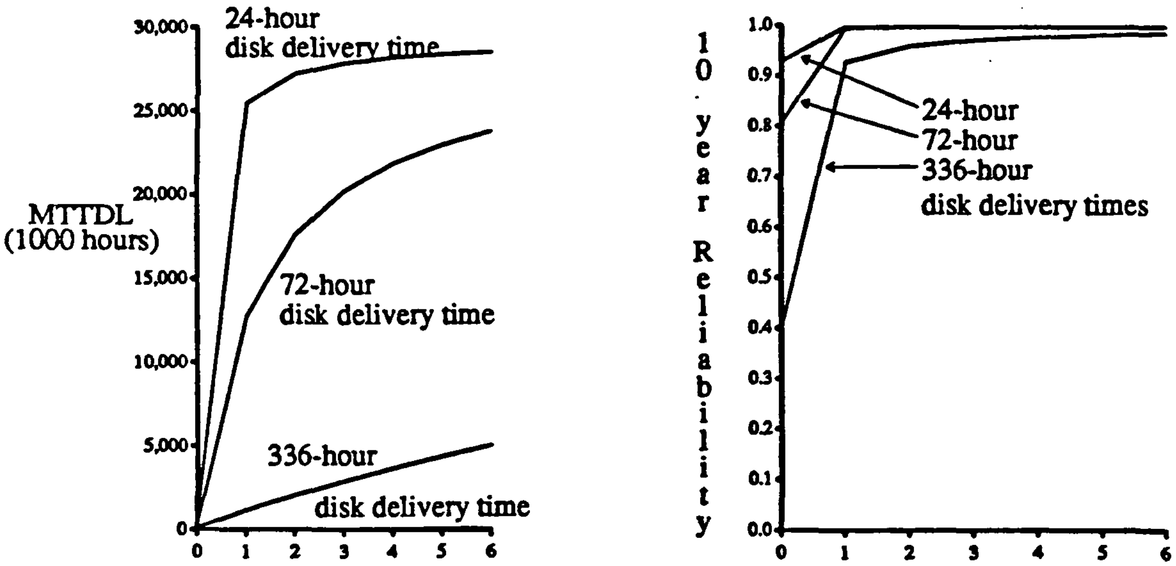
Figures 5.23a and 5.23b: Simulatation versus Estimatation with Spares. *On the left, Figure 5.23a shows the relative difference between the mean array lifetime estimates by simulation and by Equation 5.22 for 100 parameter sets selected at random. Recalling the variability inherent with simulation shown by Figure 5.6b, Figure 5.23a shows good agreement. On the right, Figure 5.23b shows the simulated 1-, 3-, and 10-year reliabilities in comparison to those appropriate to exponentially distributed lifetimes (largely obscured dotted lines). Again, this is solid evidence that simulated lifetimes have an exponential distribution.*

Based on the findings in previous sections, I expect that disk-array lifetimes under this model have an exponential distribution. Figure 5.23b shows the simulated 1-, 3-, and 10-year reliabilities against the corresponding simulated *MTTDL*. As in previous sections, the exponential 1-, 3-, and 10-year reliabilities are shown by the largely obscured dotted lines. This constitutes strong evidence that for this model as well as for previous models simulated disk-array lifetimes are exponentially distributed. To pursue this further, Appendix C presents an application of Pearson's chi-square goodness-of-fit test. As I expected, this test does not yield strong evidence against the hypothesis that simulated disk-array lifetimes have exponential distributions.

5.6.2. Implications for the Design of Disk Arrays

Investigating the effects of this estimate for the mean lifetime of a disk array, I apply $MTTDL_{IndepSpares}$ to my strawman example. Figures 5.24a and 5.24b show the *MTTDL* and the 10-year reliability as a function of the size of an on-line spare pool. In this figure, replacement disks are not ordered until all spares have been assigned to replace failed disks. This reorder policy will amortize the cost of delivering a set of new spare disks over the number of disks in this set. This can yield substantial savings beause field-service visits may cost between on tenth and one half of the cost of a disk [Ng90].

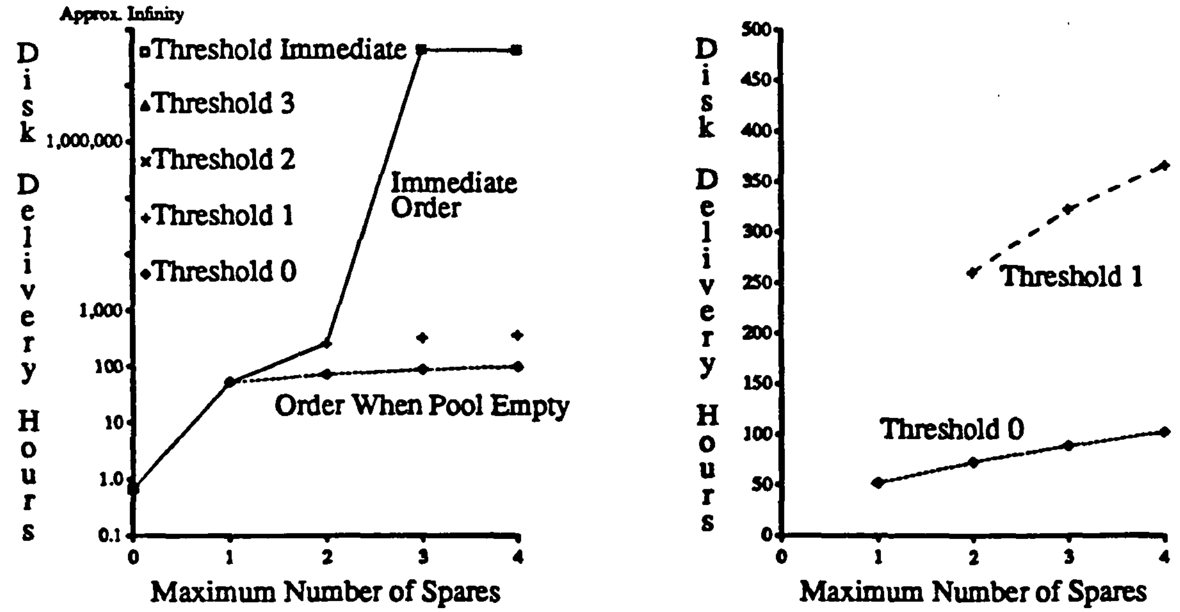
Although the improvement in *MTTDL* with increasing spare disks is much less than with longer disk-delivery times, the 10-year reliability exceeds 0.90 with only one spare disk even if the disk-delivery time is 336 hours (two weeks)! If a replacement disk is ordered as soon as a



Figures 5.24a and 5.24b: Evaluating Benefits of a Small Pool of Spare Disks. *A small pool of spare disks alleviates the effects of disk-delivery time in my strawman disk array if it is not subject to string failures. On the left, Figure 5.24a shows the MTTDL against the maximum (and initial) number of spare disks. On the right, Figure 5.24b shows the 10-year reliability for the strawman disk array against the same metric. Recall that this array has seven parity groups of 10 data disks and a parity disks. Each disk has a mean lifetime of 150,000 hours. When a disk fails and a spare is available, disk recovery takes one hour on average. Replacement disks for the spare pool are ordered when the last spare is assigned to replace a failed disk and are delivered in 24 (one day), 72 (three days) or 336 hours (two weeks).*

disk fails instead of after the last spare is assigned to a failure, then all three *MTTDL* curves achieve 29,000,000 hours with only four spare disks, and all three reliability curves exceed 0.99 with only two spare disks!

Figures 5.25a and 5.25b depict the relationship between the disk-delivery time, the replacement-order threshold, and the maximum number of spares for my strawman disk array. Because an order delivery involves a person meddling with the disk array, a high maximum number of spares and a low reorder threshold reduces the frequency that the disk array is exposed to human error. However, a large spare pool increases disk and support-hardware costs and a low threshold requires faster disk delivery to avoid lowered reliability. Both figures display the maximum number of hours that disk delivery can take without causing the 10-year reliability to drop under 0.995. Figure 5.25a shows that when the maximum number of spares is three or more and the order threshold is two or more, any reasonable disk delivery time will



Figures 5.25a and 5.25b: Pool Size, Reorder Threshold, versus Disk Delivery Time. *These two figures show maximum disk delivery time that yields a 10-year reliability greater than or equal to 0.995 in my strawman disk array. This array has seven parity groups of 10 data disks and a parity disk. Each disk has a mean lifetime of 150,000 hours and a mean recovery time of 1 hour. A low threshold leads to less frequent orders. Although this requires faster disk delivery or larger spare pools, it also reduces the frequency that error-prone humans tamper with the disk array.*

ensure that 10-year reliability exceeds 0.995. This reduces disk delivery costs, but does not minimize the frequency that orders are placed and deliveries arrive. Figure 5.25b concentrates on disk delivery times between a day and a few weeks. With a maximum of four spare disks in this array of 70 data disks, a disk delivery time of less than about 100 hours allows this 10-year reliability goal to be met with minimal reorder threshold of zero. For less pressure on disk delivery time, an reorder threshold of one is a good compromise between infrequent orders and inexpensive disk delivery.

These figures show the substantial benefits provided by even a small pool of on-line spares. However, this model fails to consider the negative consequences of string failures demonstrated in Section 5.5. The next section addresses this problem.

5.7. Dependent Disk Failures with On-line Spares

This section examines disk array reliability under the influence of dependent disk failures and with the benefit of on-line spare disks. Section 5.5 shows that dependent disk failures, although tolerable, can dramatically reduce the reliability that would be expected from the same disk array suffering only independent failures. On the other hand, Section 5.6 shows that a small number of on-line spares can dramatically improve the reliability of a disk array suffering only independent disk failures. Naturally, I would like to use on-line spare disks to overcome the limitations imposed by dependent disk failures.

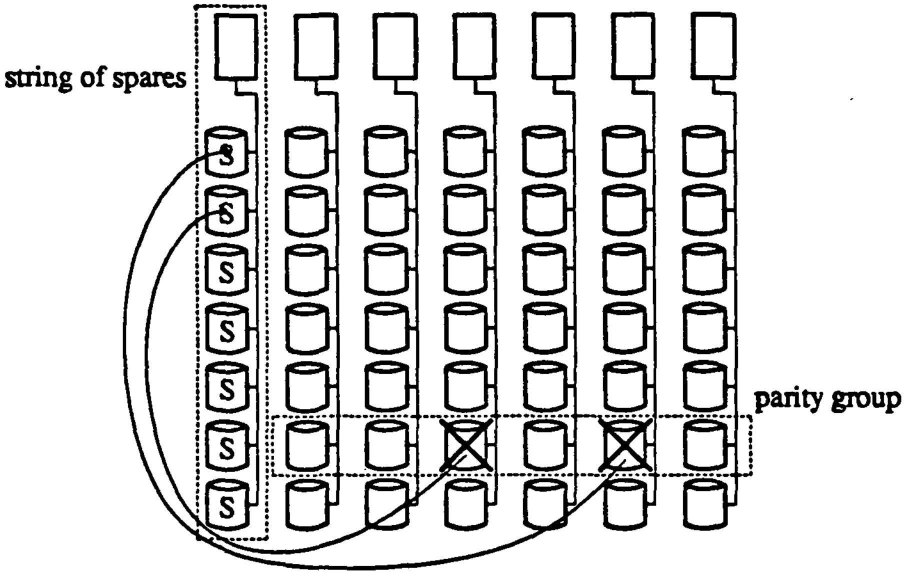


Figure 5.26: Two Failed and Spared Disks Example. *This example of an orthogonal disk array has suffered two disk failures at different times in the same parity group (N+1=6). In this case the second failure did not happen until after the first failure's assigned spare disk completed recovering the first failure's data. Because the two failure recoveries did not overlap, the disk array remains operational without loss of data. Notice that the two assigned spares are physically in the same string and logically in the same parity group. Until the failed disks are replaced and rebuilt (by copying from their respective spare disks or by another recovery operation), this array is not orthogonal. While it is not orthogonal, it is not protected against loss of data because the isolated failure of the string of spares may erase two disks in one parity group. This period of vulnerability can be reduced, but not eliminated, by manually removing the failed disks, moving two of the unassigned spare disks into the vacated positions, and copying or recoverying the assigned spare disks' contents onto the correctly-located disks. This operation to reduce vulnerability is an unnecessary, and possibly error-prone, human interaction with the disk array; I will not explore this method further in this work.*

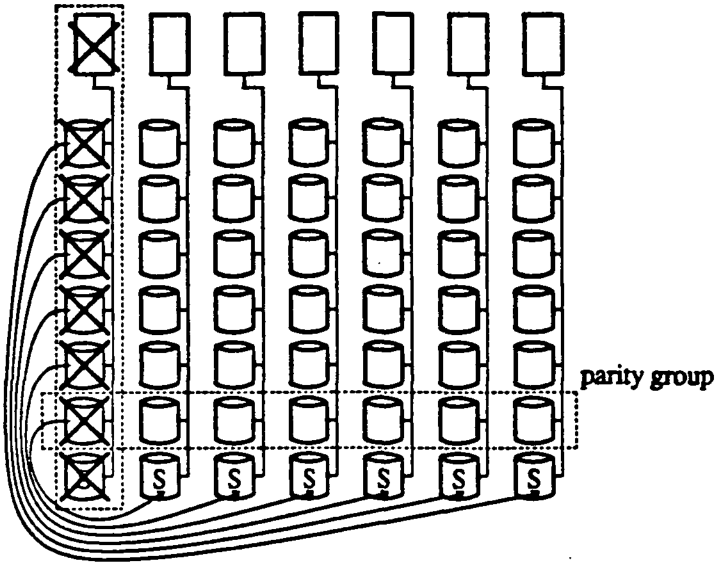


Figure 5.27: String Failed and Spared Example. *This example shows the effect on orthogonality of a string failure when on-line spare disks are allocated one to a string (unlike the case shown in Figure 5.26). Although there are enough spares to replace all data and parity disks affected by this string failure, until the failed string is repaired every other string has two disks that are logically in the same parity group.*

By including on-line spare disks into an orthogonal array, a few new issues are introduced because on-line disks must be attached to support hardware somewhere in the array. Figure 5.26 shows how an orthogonal array with spare disks may surrender its orthogonality while waiting for recently failed disks to be replaced. A second issue is the allocation of spare disks to strings. I may choose to allocate spare disks together in one string, as I have done in Figure 5.26, or spread them out over strings that also contain data or parity disks. For a small number of spares, these two choices yield comparable reliabilities. When the number of spares is equal to or larger than the number of disks attached to a string, however, a string containing only spare disks effectively replaces a failed string without disturbing the array's orthogonality. In contrast, if on-line spare disks are allocated one to each string then the failure of a string may negate orthogonality until the failed string is repaired and its disks's contents recovered. Figure 5.27 shows an example where a string has failed in an orthogonal array with spare disks allocated one to each string. In this case, there are enough spare disks to immediately replace the failed string, but each spare is assigned into a parity group already represented on the spare's string. Until the failed string is repaired, a second string failure will cause data to be lost even if no recoveries are in progress. Because of this increased vulnerability, I allocate spare disks to strings containing only spare disks for the rest of this chapter.

Because the lifetimes of simulated disk arrays were found to be distributed exponentially in Section 5.5, where the effects of string failures were modeled without spares, and in Section 5.6, where the effects of on-line spare disks were modeled without string failures, I also expected simulated lifetimes in arrays with on-line spares and string failures to be distributed exponentially. Figure 5.28 provides strong evidence for the correctness of this expectation. It shows the 1-, 3-, and 10-year reliabilities for a collection of 100 parameter sets each simulated with about eight different spare pool sizes. As in previous sections, the 1-, 3-, and 10-year reliabilities for an exponential random variable with a mean equal to the simulation's estimate

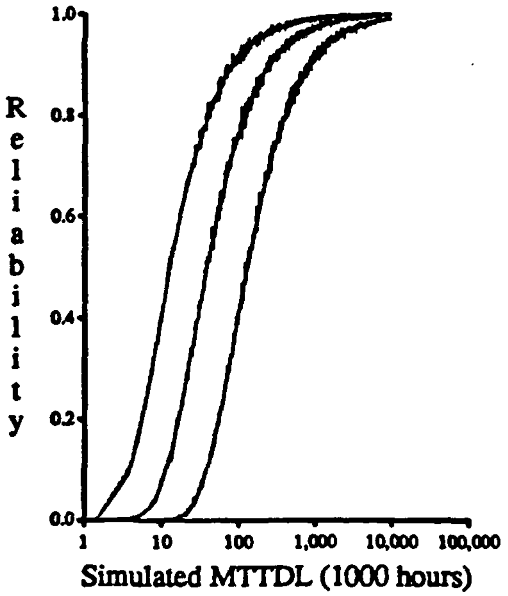


Figure 5.28: Simulated versus Exponential 1-, 3-, and 10-year Reliabilities. *This figure shows that simulated 1-, 3-, and 10-year reliability estimates closely approximate exponential 1-, 3-, and 10-year reliabilities. I collected these estimates for 100 parameter sets selected at random, each simulated with about eight different spare pool sizes. Each simulated reliability estimate is plotted as a function of its simulated mean lifetime estimate. Simulated reliability estimates are connected by a solid line and the corresponding exponential reliabilities are connected by a dotted line.*

(shown by the obscured dotted lines) approximate the simulation's estimates accurately. Also following the methods of previous sections, Appendix C presents the results of applying Pearson's chi-square goodness-of-fit to simulated lifetime data. Again, this test does not provide evidence against the hypothesis that the lifetimes of disk arrays in this section have exponential distributions.

I conclude that simulated lifetimes in arrays with on-line spares and string failures are distributed exponentially. This means that a complete reliability model depends only on the mean disk-array lifetime, *MTTDL*:

$$R(t) = e^{-t/MTTDL} \ . \tag{5.29}$$

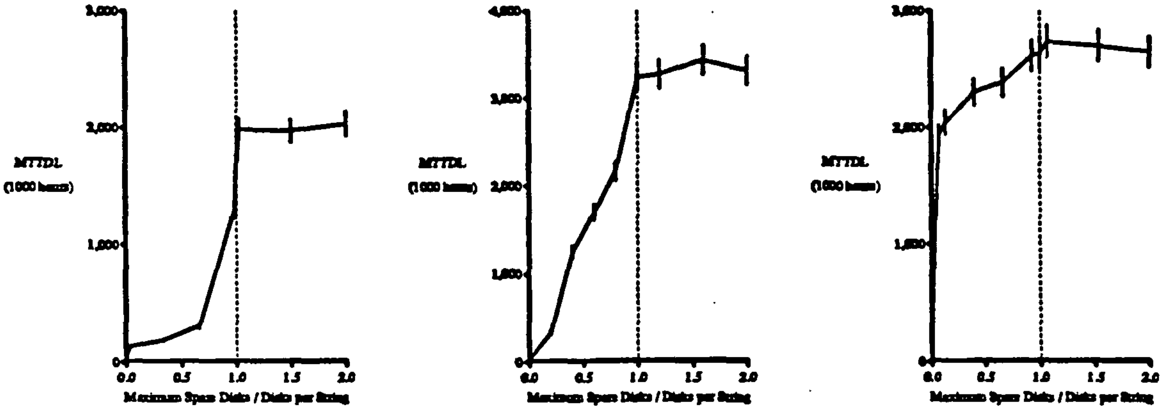
The following sections estimate *MTTDL*.

5.7.1. Modeling Mean Array Lifetime

A complete Markov model for dependent disk failures and on-line spare disks has an enormous number of states. As a result, not only would the task of specifying transition rates be error-prone, but tools like Sharpe would be unable to solve the models. For these reasons I have not developed a complete Markov model.

One alternative to developing a complete Markov model is to apply the approach used in Section 5.6. To do this, I would need to compute the expected time between instances where the disk array is fully populated and properly orthogonal and the probability that data is lost in each of these intervals. At this time, I have not found a model that accurately estimates the mean lifetime of a disk array for each configuration of the spare pool. Instead, I describe the effect that on-line spare disks have on the mean lifetime of a disk array for particular numbers of spare disks. This model, albiet an incomplete one, describes the general form of the mean time until data is lost as a function of the number of spare disks.

Figures 5.29a, 5.29b, and 5.29c show examples of a simulation's estimates for the mean lifetime of three different disk arrays as a function of the ratio between the maximum number of



Figures 5.29a, 5.29b, and 5.29c: Effect of Spares on Mean Lifetime. *As the vertical dotted lines show, these three simulation examples suggest that MTTDL does not benefit from more than one string of on-line spare disks. These figures also show that some configurations approach that peak with only a few spare disks while others do not approach that peak until the number of spare disks fills a string. These three figures are examples from the 100 randomly selected parameter sets used to verify models in this section. On the left, Figure 5.29a displays a configuration of 50 parity groups of three data disks and a parity disk. Because it is orthogonal, this disk array has 50 disks on each of its four strings. For this figure, mean disk lifetime is 50,000 hours, mean string lifetime is 100,000 hours, mean disk recovery time is one hour, mean string repair time is 72 hours, and replacement-disk delivery time is 72 hours. In the middle, Figure 5.29b displays a configuration of five parity groups of 16 data disks and a parity disk (17 strings with five disks on each). For this figure, mean disk lifetime is 50,000 hours, mean string lifetime is 1,000,000 hours, mean disk recovery time is 0.5 hours, mean string repair time is 24 hours, and replacement-disk delivery time is 72 hours. On the right, Figure 5.29c displays a configuration of 15 parity groups of 12 data disks and a parity disk (13 strings with 15 disks on each). For this figure, mean disk lifetime is 200,000 hours, mean string lifetime is 200,000 hours, mean disk recovery time is two hours, mean string repair time is two hours, and replacement-disk delivery time is 24 hours. In all three cases an order is issued whenever a disk fails; that is, the reorder threshold is one less than the maximum number of spare disks. Vertical bars display the 95% confidence interval computed for the data point of each simulation.*

spare disks in the array and the number of disks on a string. I selected these three examples to show that some configurations achieve their maximum mean lifetime with only a few spares and other configurations do not approach their maximum mean lifetime until they include many spares. In all three cases, however, increasing the number of spare disks beyond one string of spare disks does not significantly improve mean lifetime. This observation, that one string populated with on-line spare disks appears to be all that is needed to maximize the mean lifetime of a disk array, is the basis of this section's model, and is illustrated in Figure 5.30. It estimates *MTTDL* when there are zero, one, two, and an infinite number of strings populated with on-line spare disks. In most cases, one string of spare disks yields as high an *MTTDL* as two strings of spare disks, and in almost all cases two strings of spare disks yield as high an *MTTDL*

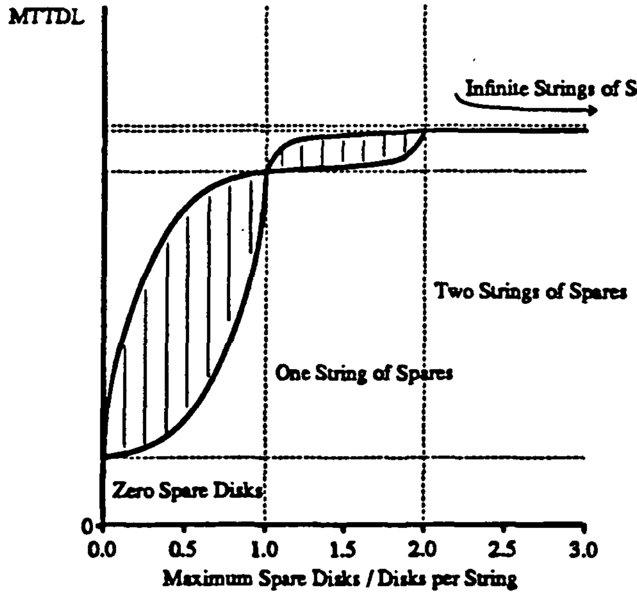
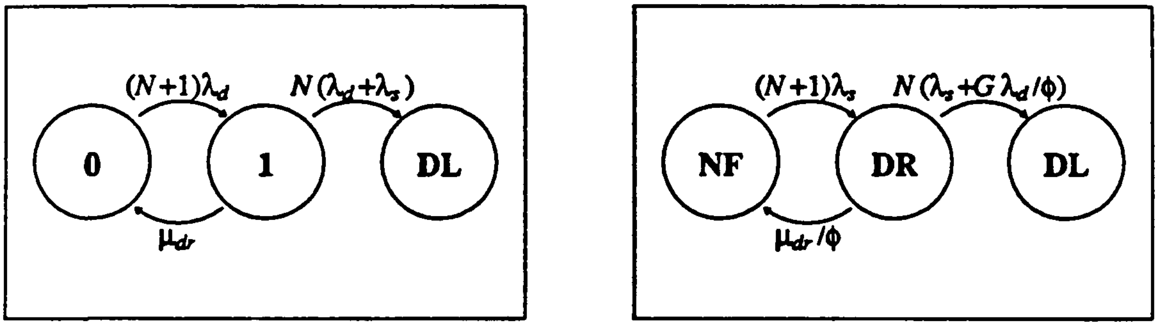


Figure 5.30: Generic Model for MTTDL. *This figure shows the general form of this section's simple model for the MTTDL of a disk array that suffers string failures as well as individual disk failures but has the benefit of on-line spare disks. This model estimates MTTDL when there are no on-line spare disks, when there is one string populated with on-line spare disks, when there are two strings populated with on-line spare disks, and when there are an infinite number of strings populated with on-line spare disks.*

as infinitely many strings of spare disks.

5.7.2. Infinite Spares Bound

In this section I present an upper bound on *MTTDL* (and, consequently, reliability) in a disk array that suffers both independent disk failures and string failures but that has the benefit of on-line spare disks. This upper bound models the case where there are infinitely many strings fully populated with spare disks. As in Section 5.5, the sources of data loss are secondary failures during an individual disk repair or an individual string repair. Because concurrent double failures are rare, I separately model these two cases and estimate the overall rate at



Figures 5.31a and 5.31b: Submodels for Orthogonal Disk Arrays with Infinite Spares. *These models are similar to those in Section 5.5.2, Figures 5.15a and 5.15b, which model data loss sources in an orthogonal array without on-line spare disks. In both cases the two sources of data loss in orthogonal disk arrays are additional failures during a disk repair and additional failures during a string repair. Figure 5.31a, on the left, is the same as Figure 5.15a except that disk replacement is immediate, so disk repair requires recovery only. It shows the submodel for data loss in a single parity group caused by a second failure during a (non-string-failed) disk recovery. Each of the* $N+1$ *disks in a parity group fails independently with* $MTTF_{disk} = 1/\lambda_d$ *and is recovered with* $MTTR_{disk-recovery} = 1/\mu_{dr}$. *While a disk is being repaired, the failure of any of the other N disks or their strings causes data loss. There are G instances of this sub-model contributing to the overall data loss rate because there are G parity groups in the disk array. Figure 5.31b, like Figure 5.15b, shows the submodel for data loss caused by a second failure during or soon after a string repair. Because an infinite number of spare disks and strings are available, string replacement is immediate. In this case the only period of vulnerability is the time required to recover of each of the disks on the replaced string. This means that after one of the* $N+1$ *strings, each with* $MTTF_{string} = 1/\lambda_s$, *fails, the array remains vulnerable to data loss on the next failure until all G disks on the replacement string have recovered. As in Figure 5.15b, the rate at which the slowest of G disk recoveries takes to complete is the reciprocal of the expected maximum of G disk recoveries,* $MTTR_{Gdisks} = \phi/\mu_{dr}$, *where* $\phi = 1/1+1/2+...+1/G$. *While at least one disk is still recovering, data will be lost with the failure of any of the other N strings or the failure of any other disk in a parity group that is still recovering. The average number of parity strings vulnerable while at least one disk is still recovering is* G/ϕ.

which data is lost as the sum of the rates of data loss arising from each of these component models. Figures 5.31a and 5.31b show sub-models for data losses initiated by individual disk failures and individual string failures, respectively. Applying the methods of Section 5.4.1 and 5.5.2, a bound on the mean lifetime of a disk array of G parity groups, each containing $N+1$ disks plus an infinite number of spare disks and strings, is:

$$MTTDL_{InfiniteSpares} = \frac{\dfrac{MTTF_{disk}{}^2}{GN(N+1)MTTR_{disk-recovery}}}{\dfrac{1+\alpha_F}{1+(2N+1)\varepsilon'_{dd}+N\varepsilon'_{ds}} + \dfrac{\alpha_F(1+\alpha_F\phi/G)}{1+GN\varepsilon'_{dd}+(2N+1)\varepsilon'_{ds}}} \qquad (5.30)$$

$$\text{where } \alpha_F = \frac{MTTF_{disk}}{MTTF_{string}},\ \varepsilon'_{dd} = \frac{MTTR_{disk-recovery}}{MTTF_{disk}},\ \varepsilon'_{ds} = \frac{MTTR_{disk-recovery}}{MTTF_{string}},$$

$$\text{and } \phi = \frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{G}.$$

Mean array lifetime closely approaches this bound with at most two strings of spare disks and more often with only one string. The next sections present estimates of mean array lifetime with one and two strings of spare disks and contrast these estimates to simulation and to this infinite spare disks bound.

The simple model shown in Figure 5.3 of Section 5.4.1 provides an alternative approximation for mean array lifetime when there are an infinite number of spares. Because each string failure induces each of the disks attached to it to recover, the rate at which each disk fails in the simpler model of Figure 5.3 can be increased by the rate at which each string fails. Mean lifetime and reliability are then given by Equation 5.11 in Section 5.4.2, with $MTTF_{disk}$ set to the reciprocal of this increased rate of failures and with $MTTR_{disk}$ set to the mean disk-recovery time. This approximation is pessimistic because it assumes the disk recoveries that are induced by a string failure occur at different times. Spreading these disk recoveries over time causes the array to be vulnerable to second-failed-string data losses for a larger fraction of time. Nevertheless, if the number of disks on a string is small or the mean string lifetime is long relative to the mean disk lifetime, this approximation is accurate. For example, it yields a mean lifetime within 10% of $MTTDL_{InfiniteSpares}$ in my strawman disk array[5] as long as mean string lifetime exceeds 250,000 hours.

5.7.3. One String of On-Line Spare Disks

In this section I present an estimate for the mean lifetime of an orthogonal disk array with one string of on-line spare disks. Once again I estimate the contribution to the overall rate at which data is lost from each source of loss separately and then sum these estimates. In this case I consider three submodels for data loss: the infinite-spare-disks model from Equation 5.30 in the last section, the independent-disk-failures-with-spare-disks model from Equation 5.22 in

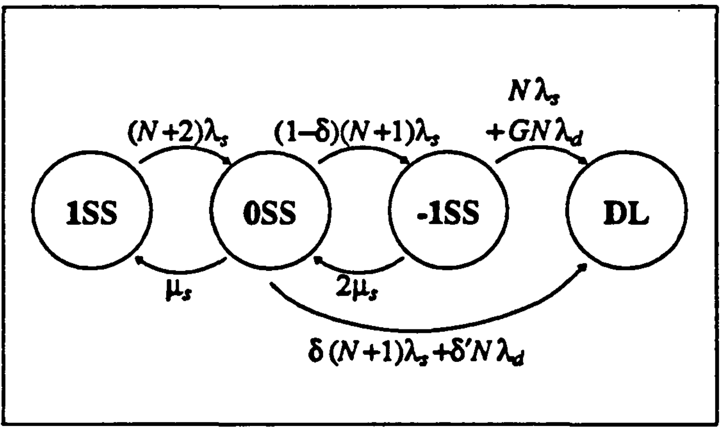


Figure 5.32: Submodel for Orthogonal Array with One String of Spares. *When there is only one string of spare disks, not all string failures will find a replacement string available immediately. This figure shows a Markov model for data losses that occur while a failed string is forced to wait for a replacement string. While there is one spare string available, state 1SS, each of the $N+2$ strings fails with $MTTF_{string} = 1/\lambda_s$ and is repaired or replaced with $MTTR_{string} = 1/\mu_s$. If one of the remaining $N+1$ strings fails, it causes data loss with probability δ because individual disk repairs were in progress. With probability $1-\delta$, a second string failure while the first is being repaired is survived. Similarily, data losses can be caused because other disks fail independently in any one of an average of δ' groups being repaired. While there are two strings being repaired, state $-1SS$, each at rate $MTTR_{string} = 1/mu_s$, any other string failure or individual disk failure on another string will cause data loss.*

[5] Recall that my strawman disk array has seven parity groups of 11 disks each. Each disk has an exponential lifetime with a mean of 150,000 hours; mean disk-recovery time is 1 hour.

Section 5.6, and a new model that accounts for data losses triggered by the failure of one or two more strings during a string repair. The mean lifetime of an orthogonal disk array with one string of spare disks, $MTTDL_{OneSpareString}$, is then:

$$\frac{1}{MTTDL_{OneSpareString}} = \text{Infinite–spares data–loss rate} + \text{Spares–exhausted data–loss rate}$$
$$+ \text{1Spare–string–repairing data–loss rate} \qquad (5.31)$$

Figure 5.32 describes the model for the data-loss rate arising from additional string failures during string repair. Using the method given by Equation 5.7 in Section 5.4.1 for deriving the mean time until data is lost beginning with an extra string of spare disks, the third component of the rate of data loss in this section is:

$$\text{1Spare–string–repairing data–loss rate} = \frac{\lambda_1(\lambda_2\lambda_3+\lambda_3\lambda_4+\lambda_4\mu_2)}{\lambda_1(\lambda_2+\lambda_3)+\lambda_3(\lambda_2+\lambda_4+\mu_1)+\mu_2(\lambda_1+\lambda_4+\mu_1)} \qquad (5.32)$$

$$\text{where } \lambda_1 = \frac{(N+2)}{MTTF_{string}},\ \lambda_2 = \frac{(N+1)(1-\delta)}{MTTF_{string}},\ \lambda_3 = \frac{N}{MTTF_{string}} + \frac{GN}{MTTF_{disk}},$$

$$\lambda_4 = \frac{(N+1)\delta}{MTTF_{string}} + \frac{N\delta'}{MTTF_{disk}},\ \mu_1 = \frac{1}{MTTR_{string}},\ \text{and}\ \mu_2 = \frac{2}{MTTR_{string}}.$$

In this model I use two parameters, δ and δ', to approximate more complex interactions between string failures and the replacement of failed data disks. Figure 5.33 shows a more accurate representation of this interaction. Because the number of states in this Markov model is dependent on the number of disks attached to a string, G, which is potentially large, it is much less convenient than the model in Figure 5.32. I present this more complex model to help me explain the values of δ and δ' in Equation 5.32.

The top row of states in Figure 5.33 represent an expanded version of state 1SS in Figure 5.32. This expansion models string failures that find the spare string with a full complement of spare disks and those that find one or more replacement disks on order. The second row of states in Figure 5.33 represent an expanded version of state 0SS from Figure 5.32. This expansion tracks the number of failed data disks in the array that do not have a spare disk recovering their contents or acting in their place. While in these states, the failure of a second string will

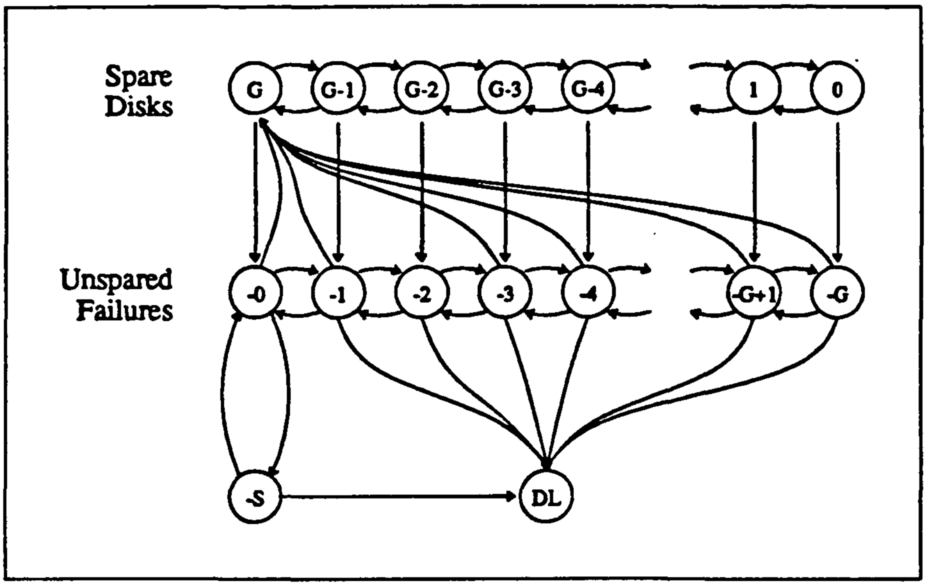


Figure 5.33: Detailed Submodel for Orthogonal Array with One String of Spares. *This Markov model includes more detail of the interaction between spare components and component failures. Its much greater level of detail is unnecessary except as an explanation for the transitions depicted in Figure 5.32. The top row of states represents the number of spare disks available on a single spare string. As independent disk failures are recovered by spare disks, the size of this spare pool decreases; it is refilled by deliveries of replacement disks. In this section, replacement disks are immediately reordered (threshold equal to one less than maximum number of spare disks). Although in simulation deliveries refill the entire spare disk pool on each arrival, I approximate refilling as separate concurrent orders for each replacement disk, each order delivered in an exponential time with mean equal to the average time between the assignment of a spare disk and the arrival of its replacement. When a string fails the system moves to the second row of states. In these states there are no spare disks. If the number of spare disks on the spare string is less than G, the number of disks on a string, when a string fails or if individual disks fail during a string repair, there will be one or more parity groups repairing a failed disk by first waiting for a replacement disk and then recovering the failed disk's contents onto its replacement. During this slow disk repair process, the completion of a string repair refills the spare pool after providing a replacement disk for all disks actively being repaired. On the other hand, a second string failure will cause loss of data unless there are no disks being actively repaired. Similarly, while parity groups are repairing slowly because no spare disks are available, additional independent disk failures in the same groups cause the loss of data.*

cause loss of data if any failed data disk has not been assigned a spare disk.[6] The parameter δ represents the probability that the array is in any of the states, $-1, -2, ..., -G$, when a second string fails during the repair of a first failed string. Similarily, the independent failure of a data

[6] The models of Figures 5.32 and 5.33 are only concerned with data losses originating with a string failure. While there are available spare strings and disks, the infinite spares model of the previous section represents losses of data caused by a second string failures during the recovery of the disks on the first failed string or by a second disk failure in one parity group during the recovery of a first failed disk. Similarly, the model in Section 5.6 represents losses of data caused by double-disk failures in one parity group while the spare disk pool has been emptied by independent disk failures.

disk in the same parity group as any already failed data disk that has not been assigned a spare disk will cause loss of data. The parameter δ' represents the expected number of unspared failed disks when a second string fails during the repair of a first failed string.

To estimate δ and δ', I assume that the threshold for ordering replacement disks is one less than the maximum number of spares; that is, an order for a replacement is issued immediately after each failure (unless there is already an outstanding order). This assumption is not unreasonable because Section 5.6 and Figure 5.25 show that immediate reorder maximizes mean lifetime and reliability. My estimate is based on the steady state proportion of time the system spends in each of the states on the second row of Figure 5.33, and ignores transitions out of this row.[7] In this case transitions from state $-i$ to $-(i+1)$ occur with rate $(G-i)(N+1)/MTTF_{disk}$ and transitions from state $-i$ to $-(i-1)$ occur with rate $i/\bar{D}$ where $\bar{D}$ is the average delivery time approximated by Equation 5.19 in Section 5.5.3. With these transition rates, the steady state proportion of time spent in state $-i$, π_i, is found by solving the system of equations, $i=0,1,2,\ldots,G-1$:

$$\pi_i \times (G-i)(N+1)/MTTF_{disk} = \pi_{i+1} \times (i+1)/\bar{D} \quad \text{and} \quad \sum_{i=0}^{G} \pi_i = 1 .$$

The solution for this system is

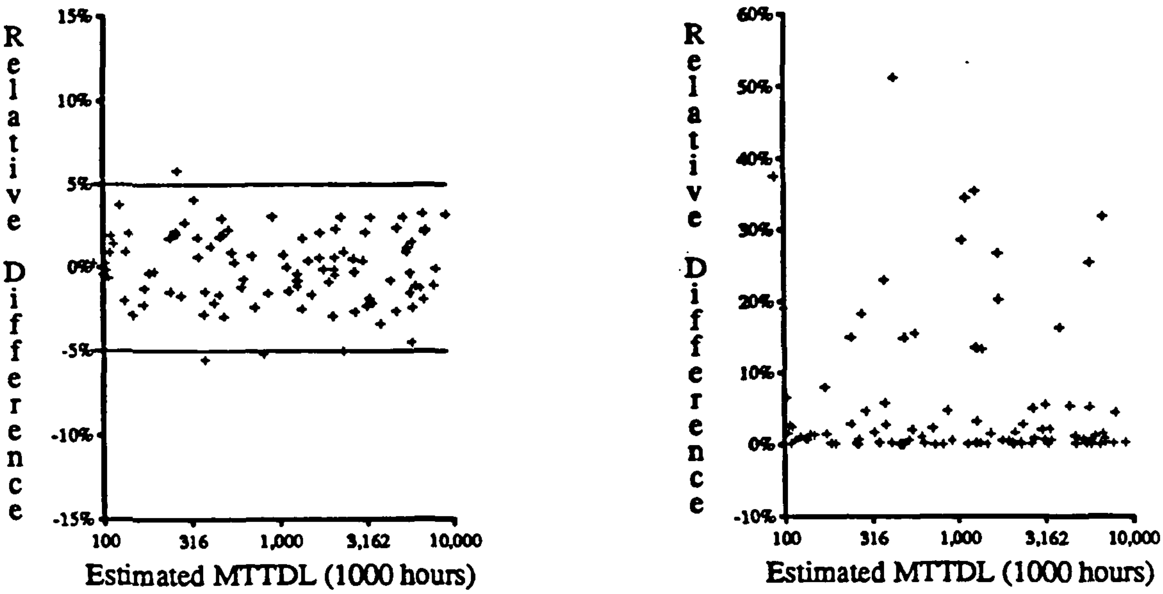
$$\pi_i = \binom{G}{i} \left[\frac{(N+1)\bar{D}}{MTTF_{disk}} \right]^i \pi_0 ,$$

$$\pi_0 = \left\{ \sum_{i=0}^{G} \binom{G}{i} \left[\frac{(N+1)\bar{D}}{MTTF_{disk}} \right]^i \right\}^{-1} ,$$

$$\delta = 1 - \pi_0 , \quad \text{and} \quad \delta' = \sum_{i=0}^{G} i \pi_i . \qquad (5.33)$$

Figure 5.34a shows that this model for mean time until data is lost with one string of spare disks agrees with the simulation's estimates within the inherent variation of simulation. As I have done in earlier sections, this comparison is made for 100 parameter sets selected at random

[7] This is not correct because string repairs should shift probability mass toward states with fewer unspared, failed disks. Fortunately, the error in this approximation is quite small for the range of parameter values that are conceivable in disk arrays.



Figures 5.34a and 5.34b: MTTDL with One String of Spares. *On the left, Figure 5.34a shows the relative difference between the mean lifetime of disk arrays with one string of spare disks estimated by simulation and by Equation 5.31. On the right, Figure 5.34b shows the relative difference between the estimate for mean lifetime with one string of spare disks given in Equation 5.31 and the estimate for mean lifetime with an infinite number of strings of spare disks given in Equation 5.30. Both figures employ the same 100 parameter sets selected at random from a large collection of conceivable parameter sets. One parameter set is not included in Figure 5.34b because its relative difference is 520%. This parameter set has unusually frequent and long string repairs. Its values are: three parity groups of 21 disks each, an* $MTTF_{disk}$ *of 100,000 hours, an* $MTTF_{string}$ *of 50,000 hours, an* $MTTR_{disk\text{-}recovery}$ *of 0.5 hours, an* $MTTR_{string}$ *of 168 hours, and a replacement-disk delivery time of 8 hours.*

from a large set of conceivable values. The selected parameter sets are not intended to represent typical choices; in fact, they are intended to stress reasonable choices for parameters to test that the models are accurate for a range of choices from poor to good. Therefore, I do not claim that the relative difference between the infinite-spares estimate for *MTTDL* in Equation 5.30 and this single-string-of-spare-disks estimate is representative of their relative difference in a set of good choices for disk array parameters. Nevertheless, Figure 5.34b shows their relative difference in this collection of 100 parameter sets. The infinite-spares estimate is more than 10% larger than the single-string-of-spare-disks estimate in 19 of the 100 parameter sets. This suggests that in some cases there is a potential for significant benefit from more than one string of spare disks.

The next section provides an estimate for mean lifetime in a disk array with two strings of spare disks and shows that the two-strings-of-spare-disks estimate is approximately equal to the infinite-spares estimate for all of these parameter sets.

5.7.4. Two Strings of On-Line Spare Disks

In this section, I develop a model for mean time until data is lost in a disk array with two strings of on-line spare disks. This model is a straightforward extension of the model in the last section. Again, I model the same three sources of data loss separately:

$$\frac{1}{MTTDL_{TwoSpareStrings}} = \text{Infinite–spares data–loss rate} + \text{Spares–exhausted data–loss rate}$$
$$+ \text{2Spare–string–repairing data–loss rate} \qquad (5.34)$$

And, again, the rate at which data is lost with infinite spares is given in Equation 5.30 of Section 5.7.2 and the rate at which data is lost arising from independent disk failures emptying the pool of spare disks is given in Equation 5.22 of Section 5.6. Although the rate arising from string failures exhausting the pool of spare disks is not the same as the equation derived in the last sec-

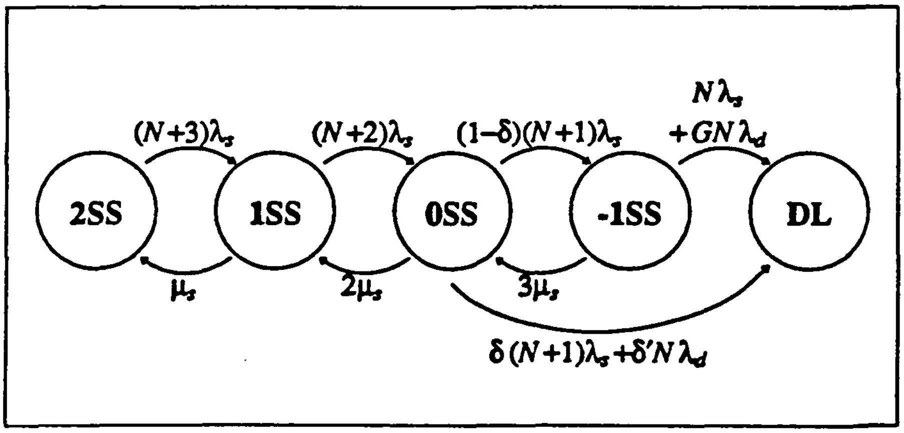


Figure 5.35: Submodel for Orthogonal Array with Two Strings of Spares. *This model represents data losses that occur because of vulnerabilities arising while two strings of disks are waiting for repair or replacement. This extension of the model in Figure 5.32 adds a state, 2SS, representing the disk array while it has two strings of spare disks available to replace failed disks and strings immediately. State transitions are almost the same as those in Figure 5.32. In particular, the parameters δ and δ′ have the same meanings and values.*

C-2

tion, it is quite similar. Figure 5.35 shows that the model I use in this section is a simple extension of the model in Figure 5.32. Applying the same methods, this model estimates:

$$\text{2Spare--string--repairing data--loss rate} = \tag{5.35}$$

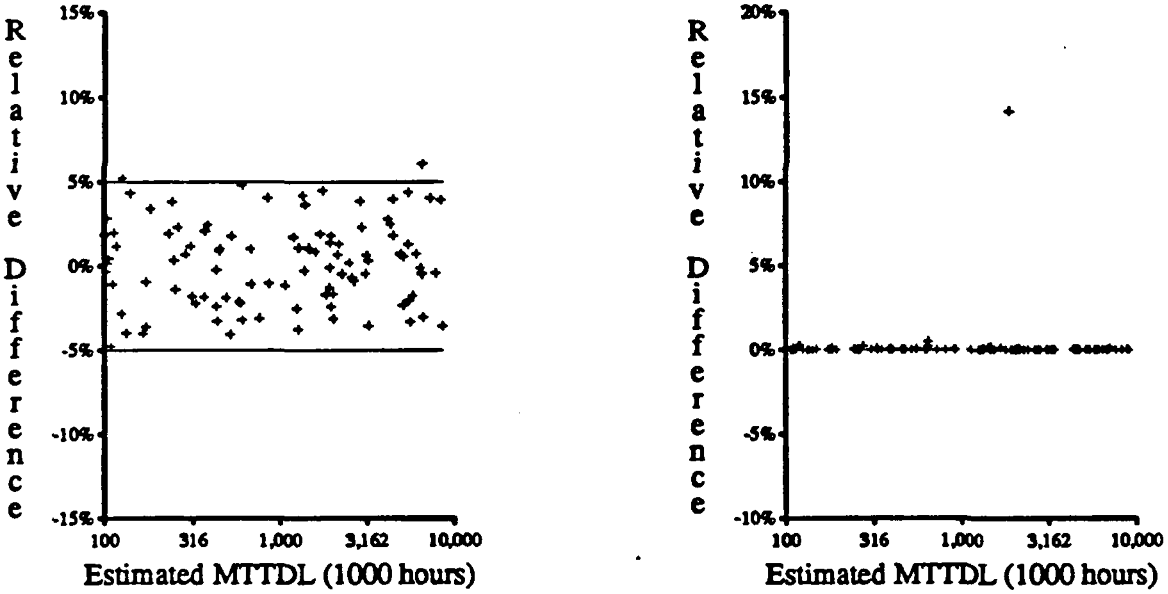
$$\frac{\lambda_1\lambda_2(\lambda_3\lambda_4+\lambda_4\lambda_5+\lambda_5\mu_3)}{\lambda_1\lambda_2(\lambda_3+\lambda_4)+\lambda_4(\lambda_1+\lambda_2)(\lambda_3+\lambda_5)+\lambda_4(\mu_1(\lambda_3+\lambda_5)+\mu_2(\lambda_1+\mu_1))+\mu_3(\lambda_1(\lambda_2+\mu_2)+\lambda_5(\lambda_1+\lambda_2+\mu_1))+\mu_1\mu_2\mu_3}$$

$$\text{where } \lambda_1 = \frac{(N+3)}{MTTF_{string}},\ \lambda_2 = \frac{(N+2)}{MTTF_{string}},\ \lambda_3 = \frac{(N+1)(1-\delta)}{MTTF_{string}},$$

$$\lambda_4 = \frac{N}{MTTF_{string}} + \frac{GN}{MTTF_{disk}},\ \lambda_5 = \frac{(N+1)\delta}{MTTF_{string}} + \frac{N\delta'}{MTTF_{disk}},$$

$$\mu_1 = \frac{1}{MTTR_{string}},\ \mu_2 = \frac{2}{MTTR_{string}},\ \text{and}\ \mu_3 = \frac{3}{MTTR_{string}}.$$

The two parameters, δ and δ', have the same values as they had in the last section. Equation 5.33 gives these values.



Figures 5.36a and 5.36b: MTTDL with Two Strings of Spares. *On the left, Figure 5.36a shows the relative difference between the mean lifetime of disk arrays with two strings of spare disks estimated by simulation and by Equation 5.34. On the right, Figure 5.36b shows the relative difference between the estimate for mean lifetime with two strings of spare disks given in Equation 5.34 and the estimate for mean lifetime with an infinite number of strings of spare disks given in Equation 5.30. Both figures employ the same 100 parameter sets also used in Figure 5.34. The parameter set excluded in Figure 5.34b is included here. It is the only parameter set with a relative difference larger than 1%.*

Figure 5.36a shows that this estimate for mean lifetime in a disk array with two strings of spare disks is in good agreement with simulation's estimates. Figure 5.36b shows that, except for one of the 100 parameter sets evaluated, two strings of spare disks yields a mean lifetime within 1% of the mean lifetime in an array with an infinite number of spare disks and strings. Even the one parameter set that has a relative difference larger than 1% only differs by 15%. Based on the data in this section and the last section, the mean lifetime of an array with an infinite number of spare disks and strings is frequently achieved with one string of spare disks and is almost always achieved with two strings of spare disks.

5.7.5. Implications for the Design of Disk Arrays

In this section I present analysis of four issues important to the design of my strawman disk array when it employs on-line spare disks:

(1) using spare disks with large parity groups to achieve higher reliability than is provided by arrays with a higher fraction of redundant disks,

(2) reducing disk-recovery time to dramatically improve mean disk-array lifetime in arrays with one or two strings of on-line spare disks,

(3) determining limits on reliability benefits provided by adding redundancy to disk-support hardware, and

(4) examining reliability when strings of spare disks are partially populated and replacement-disk reordering is not done immediately after each failure.

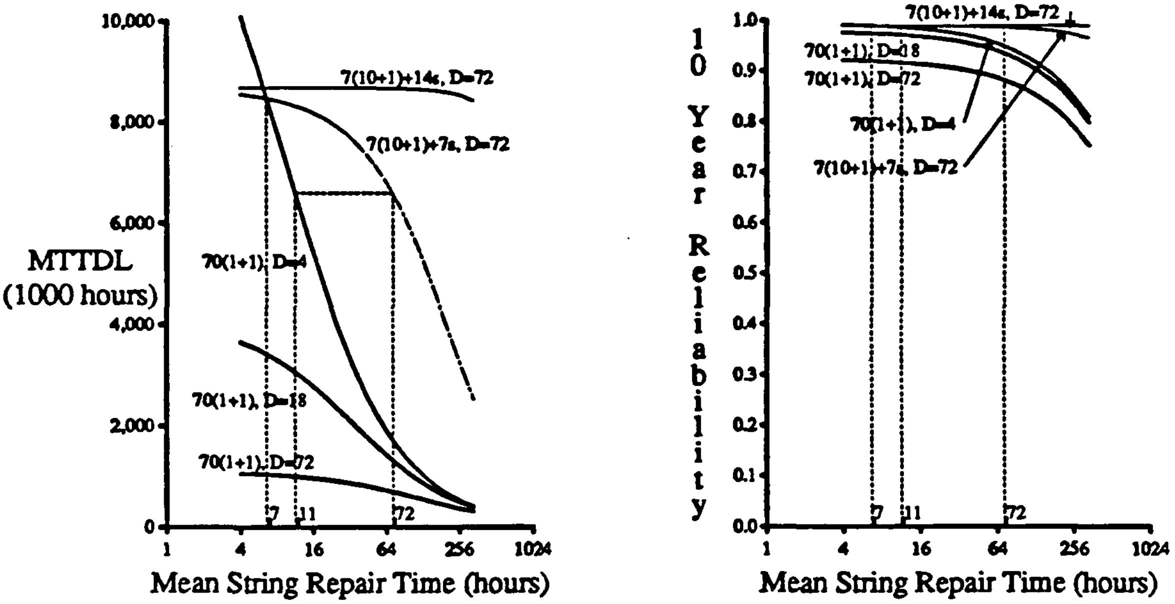
I evaluate the first three of these issues using the models developed in the last sections. Because the fourth of these issues does not meet the assumptions of these models, I explore it with simulation results.

All four of these issues are explored using the context of the strawman disk array I introduced in Table 5.1 of the introduction to this chapter. Unless I explicitly vary a parameter, my strawman disk array has seven parity groups with 10 data disks and a parity disk in each. Each disk and each string has an expontentially distributed lifetime with a mean of 150,000 hours. Disk recovery and string repair also have exponentially distribued durations with means one hour and 72 hours, respectively. Replacement-disk delivery time is the minimum of 72 hours or the time until an already ordered replacement disk arrives.

5.7.5.1. Higher Overhead for Redundancy May Not Improve Reliability

More redundancy should yield higher reliability. For example, the simple disk array lifetime model given in Equations 5.10 and 5.11 of Section 5.4.2 shows that *MTTDL* is inversely proportional to the number of disks in a parity group; larger parity groups have lower overhead and lower reliability. In this model, a disk array with mirrored disks is more reliabile than a disk array with N+1-parity redundancy. But the array with mirroring contains up to twice as many disks so its higher reliability is achieved at a substantial cost. This section shows that a disk array with N+1-parity redundancy and on-line spare disks can provide higher reliability at lower cost than a mirrored disk array with the same amount of user data.

Figure 5.37 shows that my strawman disk array with on-line spare disks achieves better reliability than a comparable mirrored disk array. With only one string of spare disks, a N+1-parity disk array with a 72-hour replacement-disk delivery time is more reliable than a mirrored disk array with either a 72- or an 18-hour replacement-disk delivery time. Even if the mirrored disk's replacement-disk delivery time is reduced to 4 hours it is still less reliable than my strawman disk array with a 72 hour replacement-disk delivery time and one string of spare disks unless mean string-repair time in both is less than seven hours. Additionally, my strawman disk array with one string of spare disks, a 72 hour replacement-disk delivery time, and a 72 hour mean string-repair time is as reliable as a mirrored disk array with a four hour replacement-disk delivery time and a 11 hour mean string-repair time. Because reducing disk-replacement and string-repair time requires increased availability of expensive human service, the cost advantage of my strawman disk array is even better than is suggested by the comparison of its 84 disks to



Figures 5.37a and 5.37b: N+1 Parity versus Mirrored Reliability. *Figure 5.37a, on the left, and Figure 5.37b, on the right, show the effect of string repair time and replacement-disk delivery time on mean lifetime and reliability, respectively, in N+1-parity and mirrored disk arrays. My strawman disk array contains seven parity groups each with 10 data disks and a parity disk (7(10+1)). Each disk and each string has an exponentially distributed lifetime with mean 150,000 hours and disk recovery time is exponentially distributed with mean one hour when a spare disk is available. The figures show two examples of my strawman disk array: one with one string of spare disks (+7s) and the other with two strings of spare disks (+14s). In both cases, replacement-disk delivery takes 72 hours (D=72). A comparable mirrored disk array has 70 parity groups each with one data and one duplicate disk (70(1+1)). Because each string contains seven disks, the mirrored disk array has 20 strings. These figures show three examples of a comparable mirrored disk array with replacement-disk delivery times of (D=) 4, 18, and 72 hours. In all arrays string-repair time is exponentially distributed with its mean displayed on the x-axis.*

the mirrored disk array's 140 disks!

The reliability advantages of N+1 parity are even better if two strings of spare disks are included. In this case my strawman disk array still has a 91 disks to 140 disks cost advantage over the mirrored disk array and its reliability is insensitive to mean string repair times as large as two weeks. Unless the mirrored disk array has a replacement-disk delivery time of four hours or less and a mean string-repair time of less than seven hours, my strawman disk array with a replacement-disk delivery time of 72 hours has better reliability.

Figure 5.37b shows the probability that these N+1-parity and mirrored disk arrays survive 10 years of operation without data loss. All configuration have a better than 75% chance of surviving 10 years and, if the mean string-repair time is less than 37 hours, all configurations have a better than 90% chance of surviving 10 years. These 10-year reliabilities are substantially higher than the 56% chance that a single disk with a 150,000 hour mean lifetime survives 10 years.

Because four of the five configurations shown in Figure 5.37b have almost the same 10-year reliability, there seems little reason other than cost to prefer one configuration over others. In particular, the relatively large differences in mean lifetime shown in Figure 5.37a do not appear in Figure 5.37b. These differences in mean lifetime are significant, however, if I consider the fraction of disk arrays that suffer data loss in 10 years. Equation 5.4 in Section 5.1 shows that doubling the mean lifetime of a disk array will halve the expected number of "angry" customers even though the probability that an individual disk array survives all 10 years without data loss is only increased a small amount. For example, 2.2% of all mirrored disk arrays with a replacement-disk delivery time of four hours and a mean string-repair time of 25 hours will lose data in 10 years, but only 1.0% of all N+1-parity disk arrays with two strings of spare disks, a replacement-disk delivery time of 72 hours, and a mean string-repair time of 72 hours, will lose data in 10 years.

These figures show the superior reliability of N+1-parity disk arrays with spare disks in comparison to mirrored disk arrays without spare disks. This is a reasonable comparison because N+1-parity disk arrays are less expensive than mirrored disk arrays even with spare disks. If comparable numbers of spare disks are added to mirrored disk arrays, increasing the cost differential, then mirrored disks achieve superior reliability.

5.7.5.2. Higher Reliability Through Faster Disk Recovery

To recover the contents of a failed disk in an N+1-parity disk array, all remaining disks in the failed disk's parity group must be entirely read and the failed disk's replacement must be entirely written. Before a block can be written to the failed disk's replacement, the corresponding blocks from each of the rest of the disks in the parity group must be have been read and

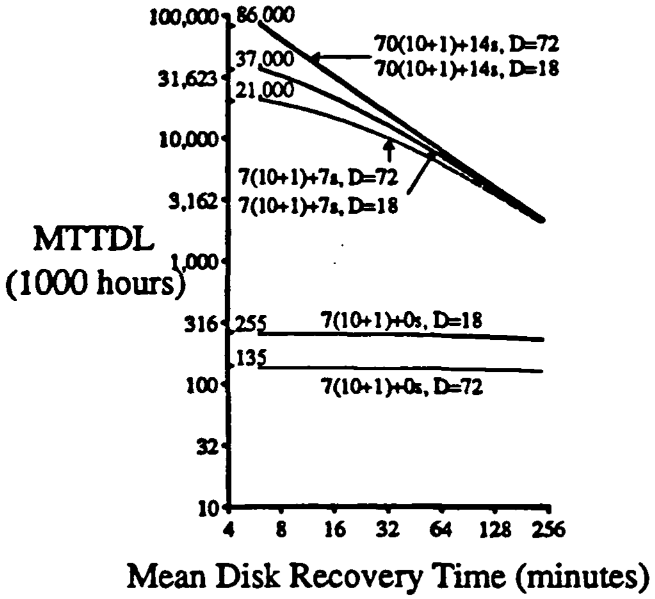


Figure 5.38: Disk Recovery Time versus On-line Spares. *This figure shows the effect on the mean lifetime of an N+1-parity disk array of faster disk recovery. My strawman disk array usually assumes an average disk recovery time of one hour. In this figure the disk recovery time is varied from about six minutes to four hours. It presents seven variations on my strawman disk array: arrays with zero, one, and two strings of spare disks are shown with replacement-disk delivery times of 18 hours and 72 hours*

their collective parity (exclusive-or) must be computed. This collective parity is exactly the failed disk's missing data, so the collective parity is then written to the failed disk's replacement disk.

If the array controller or host computer managing a failed disk's recovery has sufficient control, transfer, and exclusive-or bandwidth to read all remaining disks and write the replacement disk in parallel, a failed disk's recovery can be completed in about six minutes [Sierra90 pp 224]. This maximum recovery rate is rarely attained because many systems are not designed with sufficient bandwidth. Even if high speed recovery is possible, it would block all user accesses into the entire parity group for the duration of the recovery. In many computer systems, the unavailability of user data for many minutes or tens of minutes is tantamount to data loss because (1) the unavailable data may be out-of-date before it again supports user accesses or (2) the financial penalties derived from stalling accesses until recovery is complete are unacceptably high. Systems with these kind of high availability requirements may demand that user accesses be served during disk recovery. This will cause a failed disk's recovery be slowed down. Because the failed disk's data can be recovered block-by-block in any order, user accesses for any data in the effected parity group can be serviced, albiet at reduced performance [Muntz90].

Figure 5.38 shows the effect of changing the mean disk-recovery time on the mean disk-array lifetime. If there are no spares in an N+1-parity disk array, then the array's vulnerability to data loss is largely determined by replacement-disk delivery time. In this case, disk recovery can be slowed to one or four hours on average to accomodate user accesses without significant effect on reliability. If, instead, there are two strings of spare disks in the array, the array's vulnerability to data loss is largely determined by disk-recovery time. In this case, increasing the mean disk-recovery time by a factor of 10 from six minutes to an hour reduces mean lifetime by a factor of 10 which, in turn, increases the expected fraction of disk arrays that will lose data by the same factor of 10. With just one string of spare disks this effect is less pronounced, but it is still important for high reliability to minimize disk-recovery time.

5.7.5.3. Higher Array Reliability Through Higher String Reliability

As I mentioned in Figure 5.10 of Section 5.5, the conventional approach for avoiding low reliability in disk-support hardware is to employ more reliable, and more expensive, parts. For even higher string reliability, at even higher costs, support-hardware components can be made redundant. Because the fraction of a disk array's cost that is attributable to support-hardware is not likely to be large, it is reasonable to evaluate the contribution to array reliability that results from increased string reliability via higher quality parts or redundancy.

Figure 5.39 shows the effect of varying string reliability on the mean lifetime of my strawman disk array. If the unenhanced mean string lifetime is low – in this example, less than 100,000 hours – then doubling it doubles the mean lifetime. This effect is more pronounced in arrays that have one or more strings of spare disks. When mean string lifetime approaches or exceeds 1,000,000 hours, however, increasing it further provides little benefit for array

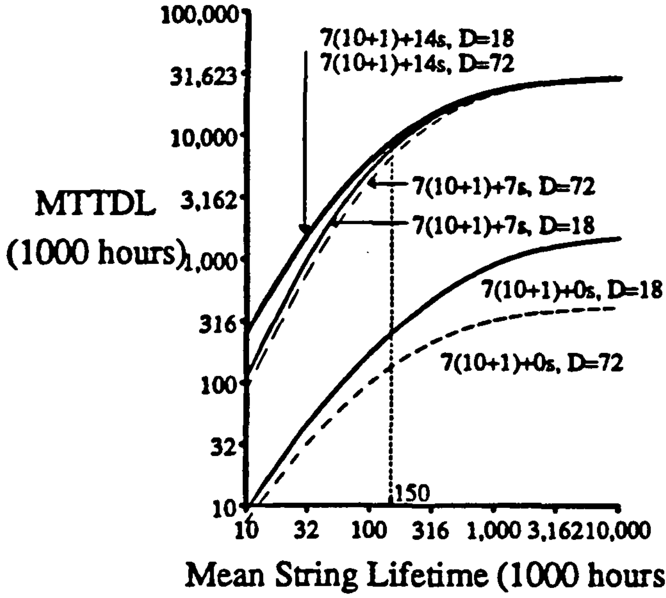


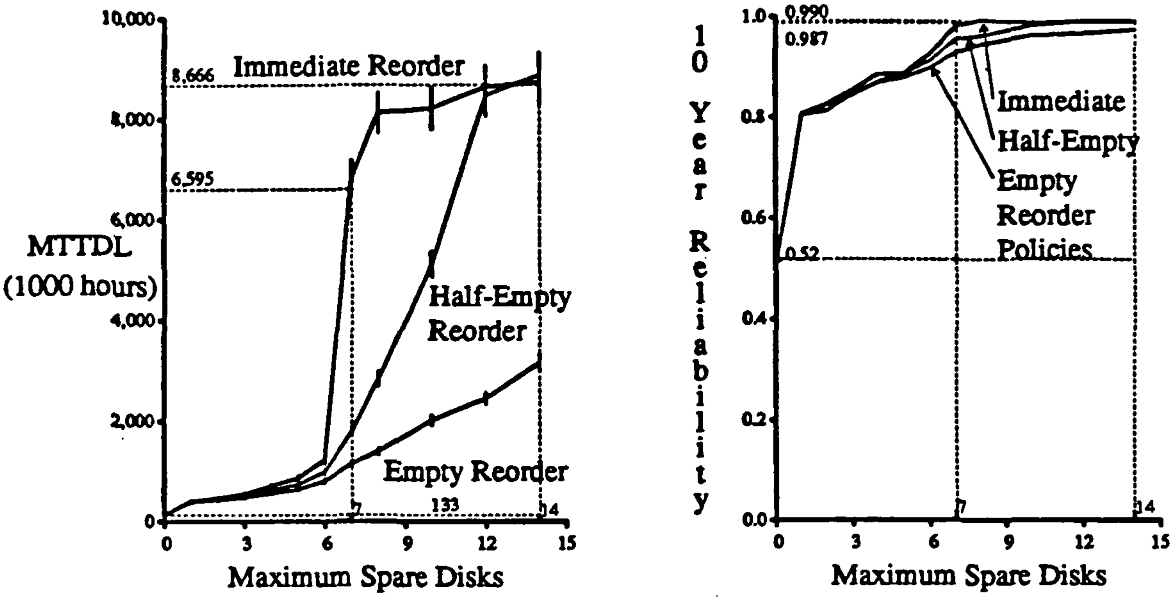
Figure 5.39: Diminishing Effect of More Reliable Strings on Array Reliability. *This figure shows the effect of string reliability on the mean lifetime of my strawman disk array. Seven variations for the disk array are shown. These are the same configurations presented in Figure 5.38 of the last section. In this figure, however, mean disk-recovery time is again fixed at one hour and mean string lifetime is varied instead. The dotted vertical line shows the default mean string reliability in my strawman disk array.*

reliability. Unless the unenhanced string reliability is very low and the cost of increasing mean string lifetime by a factor of 100 or 1,000 increases array cost by less than 10%, adding a string of spare disks is a more effective method of increasing reliability than is increasing mean string lifetime.

5.7.5.4. Partially Populated Spare Strings and Low Reorder Thresholds

In contrast to figures in the last three sections, this section presents simulation data instead of model estimates. It investigates aspects of the reliability of my strawman disk array that do not meet the assumptions of my models. In particular, this section explores two design alternatives:

(1) One or two extra strings partially populated with spare disks are desirable because disks are expensive.



Figures 5.40a and 5.40b: Partially Populated Spare Strings and Low Reorder Thresholds. *This figure shows simulated mean lifetime, on the left in Figure 5.40a, and 10-year reliability, on the right in Figure 5.40b, for my strawman disk array as a function of the maximum number of spare disks. When the maximum number of spare disks is less than 7 or between 7 and 14, there is a string partially populated with spare disks. Three variations are presented based on the reorder threshold: first, a threshold of one less than the maximum number of spare disks (immediate reorder), second, a threshold that is the integer part of half of the maximum number of spare disks (half-empty reorder), and third, a threshold of zero (empty reorder). Each estimated MTTDL generated by simulation is marked by a vertical bar showing the 95% confidence interval. Note that these curves may not be strictly monotonic increasing because of this variance in simulated estimates. Dotted lines show modeling estimates for zero, one, and two strings of spare disks.*

(2) a reorder threshold that does not cause an order to be issued immediately after every disk failure is desirable because it reduces the frequency that service personnel interacts with the disk array and because it delays the purchase of new disks.

Figures 5.40a and 5.40b show simulated estimates of mean lifetime and 10-year reliability, respectively, for my strawman disk array. Figure 5.40b shows that incorporating a single spare disk increases the chance that an individual disk array will survive 10 years without data loss from 52% to 80% and that incorporating one string of spare disks increases the chance of surviving 10 years without data loss to over 90% for all three reorder thresholds. Toward the basic goal of providing better reliability than a single disk drive, which has a 56% chance of surviving 10 years without failure, these results indicate that a small pool of spare disks with any reorder threshold is all that is necessary.

Figure 5.40a addresses high reliability in my strawman disk array. This figure shows that an immediate reorder policy achieves maximum reliability levels with substantially fewer spare disks than the delayed reorder policies. It also shows that high reliability is not achieved with any less than a fully populated string of spare disks.

Intuitively, I understand this figure by estimating the average number of spare disks in the spare pool. Because replacement-disk delivery time is much shorter than the expected time until the next disk failure, the average number of spare disks on hand is about half way between the maximum number of spare disks and one more than the threshold. This means that with immediate reorder, the array can nearly always immediately replace all disks on a failed string. This also explains why a half-empty reorder policy achieves the reliability of one string of spare disks with an immediate reorder policy once it has about one and a half strings of spare disks. This intuition is not satisfactory for explaining the reliability of the empty reorder policy because it incorrectly suggests that two strings of spare disks would be sufficient to achieve the reliability of one string of spare disks with an immediate reorder policy, which is inaccurate.

The benefit of a delayed reorder policy is largely derived from a reduction in the frequency that service personnel interacts with the disk array. The average rate of these interactions is the average rate of disk failures divided by the number of disks that must fail before replacement disks are reordered.

Average human interactions per hour = (5.36)

$$\frac{G\,(N+2)+(S+T+1)/2}{S-T}\left[\frac{1}{MTTF_{disk}}+\frac{1}{MTTF_{string}}\right].$$

When my strawman disk array has one string of spare disks and an immediate reorder policy ($S=7, T=6$), this rate is 182/150,000. When it has one and a half strings of spare disks and a half empty reorder policy ($S=11, T=6$), this rate is 186/(5×150,000), nearly five times lower without loss of reliability! Finally, when it has two strings of spare disks and an empty reorder policy

(S=14,T=0), this rate is 184/(14×150,000), 14 times lower than when the array has one string of spare disks and an immediate reorder policy. Unfortunately, the reliability of this third case is also substantially lower.

The results in Figure 5.40a indicate that my strawman disk array needs at least one string of spare disks to achieve high reliability. With four more spare disks and a policy of reordering spare disks when the pool is half empty, the frequency of human interaction with the array can be reduced by a factor of five without sacrificing reliability.

5.8. Summary and Conclusions

5.8.1. Summary of Reliability Models for Redundant Disk Arrays

In this section I restate the equations for each of the models for disk-array reliability found in this chapter. In all cases, the reliability of a disk array's data over a period of time of length t is:

$$R(t) = e^{-t/MTTDL_{estimate}}$$

where $MTTDL_{estimate}$ is an appropriate estimate of the mean time until a disk array suffers a failure that causes data to be destroyed or unavailable for a long period of time (a data loss). The reliability of a disk array over a period of length t is the probability that no failure causes data loss during that time period. The form of this equation is equivalent to stating that the time until data is lost in a disk array has an exponential distribution. This is not so much an assumption as an approximation that I demonstrated in prior sections to be appropriate because of the nature of disk array component failures [Arthurs81].

In all models I assume that disk lifetimes have an exponential distribution. Chapter 4 presents data suggesting that this is a reasonable assumption. I also assume that the lifetimes of non-disk components of a disk array's support hardware are exponentially distributed. This is a common assumption for electronic components.

All of the models in this chapter assume that each disk array has redundancy capable of recovering the contents of any single disk erasure. I assume that the time required to recover the contents of a failed disk from data on other disks has an exponential distribution. This assumption may not be accurate because the time required to recover a disk in an idle system should be short and deterministic. However, if disk recovery has lower priority than user data accesses, then, depending on system load, recovery time will be generally short and occasionally quite long – much like an exponential random variable.

In a redundant disk array that does not suffer dependent, or coupled, failure modes that damage multiple disks at the same time, the mean lifetime estimate from Equations 5.10 and 5.11 of Section 5.4.2 is:

$$MTTDL_{Indep} = \frac{(2N+1)MTTF_{disk}MTTR_{disk}+MTTF_{disk}^{2}}{GN(N+1)MTTR_{disk}} \approx \frac{MTTF_{disk}^{2}}{GN(N+1)MTTR_{disk}} .$$

In this expression G is the number of parity groups in the array, $N+1$ is the number of disks (including the parity disk) in each parity group, $MTTF_{disk}$ is the mean lifetime of each disk, and $MTTR_{disk}$ is the mean time required to repair a failed disk. Repair involves the replacement and recovery of the failed disk and is assumed to take an exponentially distributed amount of time.

If there are an infinite number of on-line spare disks then replacement time is zero and $MTTR_{disk}$ becomes $MTTR_{disk-recovery}$, the mean time required to recover a disk's contents. If replacement time is the minimum of a deterministic delivery period, D, and the time until a yet-undelivered replacement arrives, then $MTTR_{disk}$ is the sum of $MTTR_{disk-recovery}$ and the average delivery time, $\bar{D}$, given in Equation 5.19 of Section 5.5.3:

$$\text{Average delivery time} = \bar{D} = \frac{D+(G(N+1)-1)(1-e^{-D/MTTF_{disk}})D/2}{1+(G(N+1)-1)(1-e^{-D/MTTF_{disk}})}$$

If a redundant disk array that does not suffer dependent failure modes is augmented with a less than infinite number of spare disks, then the mean lifetime of its data is given by Equations 5.22, 5.27, and 5.28 in Section 5.6.1.

$$MTTDL_{IndepSpares} = \frac{1}{\frac{1}{MTTDL_{Indep}} + \frac{\text{P(data loss per order)}}{\text{Average time between filled orders}}} ,$$

where $MTTR_{disk}$ in $MTTDL_{Indep}$ should be set to $MTTR_{disk-recovery}$. The probability of data loss per replacement-disk order and the average time between each replacement-disk order arrival also depend on the maximum number of spare disks in the spare pool, S, and the number of spare disks in the spare pool when an order is issued for replacements, T. These expressions are

$$\text{P(data loss per order)} = \sum_{q=2}^{G} \begin{bmatrix} G(N+1)+T \\ T+q \end{bmatrix} (1-e^{-\lambda D})^{T+q} e^{-\lambda D(G(N+1)-q)} \left[1-\prod_{i=0}^{q-1} \frac{(G-i)(N+1)}{G(N+1)-i} \right]$$

$$+\sum_{q=G+1}^{G(N+1)} \left[\binom{G(N+1)+T}{T+q} \right] (1-e^{-\lambda D})^{T+q} e^{-\lambda D(G(N+1)-q)}$$

and

$$\text{Average time between filled orders} = D + \frac{1}{\lambda} \sum_{j=G(N+1)+T+1}^{G(N+1)+S} \frac{1}{j} ,$$

where λ is a space-saving notation for $1/MTTF_{disk}$.

Considering a redundant disk array without on-line spare disks, but suffering dependent failures caused by support-hardware failures, Section 5.5 suggests that support-hardware failures should be isolated into "string" groups orthogonally from parity groups as shown in Figure 5.11. In this case, the mean time until data is lost is given in Equation 5.16 as

$$MTTDL_{Ortho} =$$

$$\frac{\dfrac{MTTF_{disk}^{\,2}}{GN(N+1)MTTR_{disk}}}{\dfrac{1+\alpha_F}{1+(2N+1)\varepsilon_{dd}+N\varepsilon_{ds}} + \alpha_F \dfrac{1+\alpha_F\phi/G+(1+\alpha_F/G)(\alpha_R/\alpha_{Rd}+GN\varepsilon_{sd}+(N+1)\phi\varepsilon_{ss})}{((N+1)/MTTF_{string}+(2N+1)\varepsilon_{ss}+GN\varepsilon_{sd})\Psi(N+1)+\Psi(N)}}$$

$$\text{where } \alpha_F = \frac{MTTF_{disk}}{MTTF_{string}},\ \alpha_R = \frac{MTTR_{disk}}{MTTR_{string}},\ \alpha_{Rd} = \frac{MTTR_{disk}}{MTTR_{disk-recovery}},$$

$$\varepsilon_{dd} = \frac{MTTR_{disk}}{MTTF_{disk}},\ \varepsilon_{ss} = \frac{MTTR_{string}}{MTTF_{string}},\ \varepsilon_{sd} = \frac{MTTR_{string}}{MTTF_{disk}},\ \varepsilon_{ds} = \frac{MTTR_{disk}}{MTTF_{string}},$$

$$\Psi(g) = \alpha_{Rd} + GN\varepsilon_{dd} + g\,\phi\varepsilon_{ds} \quad \text{and} \quad \phi = \frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{G} .$$

In this expression $MTTR_{disk}$ is the sum of $MTTR_{disk-recovery}$ and the average delivery time, $\bar{D}$, given in Equation 5.19 and repeated above.

Finally, Section 5.7 presents a bound and two estimates for the mean time until data is lost in a disk array that suffers dependent failure modes and has been augmented with a pool of spare disks. Mean time until data loss is bounded by assuming an infinite pool of spare disks. Equation 5.30 in Section 5.7.2 reports:

$$MTTDL_{InfiniteSpares} = \frac{\dfrac{MTTF_{disk}^{\,2}}{GN(N+1)MTTR_{disk-recovery}}}{\dfrac{1+\alpha_F}{1+(2N+1)\varepsilon'_{dd}+N\varepsilon'_{ds}} + \dfrac{\alpha_F(1+\alpha_F\phi/G)}{1+GN\varepsilon'_{dd}+(2N+1)\varepsilon'_{ds}}}$$

$$\text{where } \varepsilon'_{dd} = \frac{\varepsilon_{dd}}{\alpha_{Rd}}, \text{ and } \varepsilon'_{ds} = \frac{\varepsilon_{ds}}{\alpha_{Rd}}.$$

If the array contains one string fully populated with spare disks, then mean time until data is lost is estimated by Equations 5.31, 5.32 and 5.33 in Section 5.7.3:

$$MTTDL_{OneSpareString} = \frac{1}{\dfrac{1}{MTTDL_{InfiniteSpares}} + \dfrac{1}{MTTDL_{IndepSpares}} + \dfrac{\lambda_1(\lambda_2+\lambda_3)+\lambda_3(\lambda_2+\lambda_4+\mu_1)+\mu_2(\lambda_1+\lambda_4+\mu_1)}{\lambda_1(\lambda_2\lambda_3+\lambda_3\lambda_4+\lambda_4\mu_2)}}$$

$$\text{where } \lambda_1 = \frac{(N+2)}{MTTF_{string}},\ \lambda_2 = \frac{(N+1)(1-\delta)}{MTTF_{string}},\ \lambda_3 = \frac{N}{MTTF_{string}} + \frac{GN}{MTTF_{disk}},$$

$$\lambda_4 = \frac{(N+1)\delta}{MTTF_{string}} + \frac{N\delta'}{MTTF_{disk}},\ \mu_1 = \frac{1}{MTTR_{string}},\ \mu_2 = \frac{2}{MTTR_{string}},$$

$$\delta = 1 - \left\{ \sum_{i=0}^{G} \binom{G}{i} \left[\frac{(N+1)\bar{D}}{MTTF_{disk}} \right]^i \right\}^{-1}, \text{ and } \delta' = \sum_{i=0}^{G} i \binom{G}{i} \left[\frac{(N+1)\bar{D}}{MTTF_{disk}} \right]^i (1-\delta).$$

This estimate, and the next, assume that a replacement disk is ordered immediately after each disk fails.

Finally, if the array contains two strings fully populated with spare disks, the mean time until data is lost is estimated by Equations 5.32, 5.33, 5.34, and 5.35 in Section 5.7.4:

$$\frac{1}{MTTDL_{TwoSpareStrings}} = \frac{1}{MTTDL_{InfiniteSpares}} + \frac{1}{MTTDL_{IndepSpares}} +$$

$$\frac{\lambda_1\lambda_2(\lambda_3\lambda_4+\lambda_4\lambda_5+\lambda_5\mu_3)}{\lambda_1\lambda_2(\lambda_3+\lambda_4)+\lambda_4(\lambda_1+\lambda_2)(\lambda_3+\lambda_5)+\lambda_4(\mu_1(\lambda_3+\lambda_5)+\mu_2(\lambda_1+\mu_1))+\mu_3(\lambda_1(\lambda_2+\mu_2)+\lambda_5(\lambda_1+\lambda_2+\mu_1))+\mu_1\mu_2\mu_3}$$

$$\text{where } \lambda_1 = \frac{(N+3)}{MTTF_{string}},\ \lambda_2 = \frac{(N+2)}{MTTF_{string}},\ \lambda_3 = \frac{(N+1)(1-\delta)}{MTTF_{string}},$$

$$\lambda_4 = \frac{N}{MTTF_{string}} + \frac{GN}{MTTF_{disk}},\ \lambda_5 = \frac{(N+1)\delta}{MTTF_{string}} + \frac{N\delta'}{MTTF_{disk}},$$

$$\mu_1 = \frac{1}{MTTR_{string}},\ \mu_2 = \frac{2}{MTTR_{string}}, \text{ and } \mu_3 = \frac{3}{MTTR_{string}}.$$

Although these last two models do not estimate mean array lifetime when the maximum number of spare disks is not exactly enough to populate one or two strings, Section 5.7.5 shows how they can be used to make disk-array design decisions. In particular, one or two spare disks frequently make a big difference to the chance of surviving 10 years without data loss, but it is likely that a fully populated string of spares is needed to achieve close to the minimum fraction of disk arrays that will loose data in 10 years.

5.8.2. Design of A Strawman Redundant Disk Array

Throughout this chapter I have used a strawman disk array to exemplify my models and explore array design issues. Table 5.1 in this chapter's introduction presents the strawman disk array as an attractive alternative to IBM's top-end disk subsystem, the IBM 3390. Table 5.2 summarizes estimates for the strawman disk array's mean time until data is lost and its 1-, 3-, and 10-year reliability that were presented in Sections 5.4.4, 5.5.4, 5.6.2, and 5.7.5. This table shows that without redundancy the strawman disk array has virtually no chance of surviving three or more years without data loss. This is the primary reason for including redundancy in a

MODEL	MTTDL	RELIABILITY			OVER-
	(hours)	1 year	3 year	10 year	HEAD
No Redundancy					
One Disk	150,000	0.94	0.84	0.56	0%
Seventy Disks	2,143	0.02	0.00	0.00	0%
Independent Disk Failures Only, (N+1 = 11, D = 72, $MTTR_{disk-recovery}$ = 1)					
0 Spares	411,444	0.98	0.94	0.81	10%
1 Spares, 0 Thresh.	12,734,300	0.9993	0.9979	0.9931	11%
2 Spares, 0 Thresh.	17,568,200	0.9995	0.9985	0.9950	13%
2 Spares, 1 Thresh.	28,758,300	0.9997	0.9990	0.9970	13%
∞ Spares	29,224,900	0.9997	0.9991	0.9970	∞
Independent and Dependent Disk Failures, ($MTTF_{string}$ = 150,000, $MTTR_{string}$ = 72)					
0 Spares	133,235	0.94	0.82	0.52	10%
7 Spares, 6 Thresh.	6,594,890	0.999	0.996	0.987	20%
14 Spares, 13 Thresh.	8,665,860	0.999	0.997	0.990	30%
∞ Spares	8,673,790	0.999	0.997	0.990	∞

Table 5.2: Summary of Reliability Estimates for Strawman Disk Array. *This figure summarizes mean time until data is lost and reliability estimates from each of the models in this chapter applied to my strawman disk array first presented in Table 5.1 of the introduction to this chapter. The last column shows the overhead cost of redundancy as a percentage of the non-redundant disk array cost. In this table 'Spares' is the maximum number of spares and 'Thresh.' is the reorder threshold. This disk array has 70 data disks organized into an orthogonal array of seven parity groups with 10 data disks and a parity disk in each group. Disks have exponentially distributed lifetimes with a mean of 150,000 hours. Strings have exponentially distributed lifetimes with the same mean. Disk recoveries and string repairs have exponentially distributed durations with means one hour and 72 hours, respectively. Replacement-disk delivery time is the minimum of a fixed 72 hour period, D, or the time until an already issued order arrives.*

disk array. Table 5.2 also shows that, in addition to overcoming this basic threat to data reliability, redundancy can provide high reliability with low overhead costs.

In the data of Table 5.2, I have used 10% overhead for the parity redundancy. This level of protection more than compensates for threats to reliability from independent failures alone and nearly compensates for the threats to reliability from independent and dependent disk failures without requiring on-line spare disks. If the only threats to data reliability are from independent disk failures, high reliability is achieved with only one on-line spare disk. In this case two on-line spare disks are approximately as useful as an infinite number of on-line spares. Where dependent disk failures also threaten data reliability, Section 5.7.5.4 shows that just one on-line spare disk achieves higher reliability than provided by a single disk and high reliability is provided by one string fully populated with on-line spare disks. In this case, two strings populated with on-line spare disks provides as high reliability as an infinite number of strings populated with on-line spare disks.

Section 5.7.5 examines the design of my strawman disk array in greater detail. It shows that with one string of on-line spare disks, the strawman disk array achieves higher reliability than a comparable collection of mirrored disks at lower cost and with less expensive repair processes. Section 5.7.5.2 then looks at the effect of varying disk recovery time that would result from varying the priority of recovery relative to normal user accesses. It finds that without on-line spare disks, reliability is insensitive to changes in the disk recovery rate, but with on-line spare disks, slowing the disk recovery rate substantially decreases the mean time until data is lost. The next section, 5.7.5.3, shows that adding on-line spare disks is generally more effective for improving the mean time until data is lost than is improving individual string reliability. Finally, Section 5.7.5.4 examines the possibility of reducing costs and opportunities for human error by delaying replacement-disk reordering until the spare pool is half full. It finds that with one and half strings of on-line spare disks, the mean time until data is lost in an array with a half full reorder policy is comparable to an array with an immediate reorder policy and one string of on-line spare disks. This means that the frequency of replenishing the spare pool can be reduced by a factor of five at a cost increase of less than 5% of the cost of the non-redundant array.

The net implication of this chapter's reliability models is that my strawman disk array can be made more reliable than a mirrored IBM 3390 disk subsystem at a cost less than one IBM 3390!

5.8.3. Conclusions

This chapter evaluates the reliability of disk arrays that employ N+1-parity redundancy to tolerate catastrophic failures. Because arrays of small diameter disks contain many more components than the large diameter disks they replace, their non-redundant reliability is unacceptably low. This is the *primary need for redundancy; to insure that disk arrays are at least as reliable as the single disks they replace.* Secondarily, many owners of computer systems have much higher reliability requirments for their storage systems. These customers have traditionally doubled their expenditures for magnetic disks and duplicated all of their data. Redundant disk arrays offer the opportunity to provide such customers the high reliability they seek at a much lower cost.

In this chapter I present models for disk array reliability. These models are analytic expressions based on Markov models of each source of data loss. They account for dependent disk failures, such as support-hardware failures that effect multiple disks, as well as independent disk failures. They also incorporate the benefits of on-line spare disks. These models have been validated against a detailed disk-array lifetime simulator for a wide variety of parameter selections. Agreement in most cases is within the simulator's 95% confidence interval.

The models I present in this chapter show that a redundant disk array can easily be designed to provide higher reliability than a single disk. Moreover, with a small overhead for parity and spare disks, a redundant disk array can achieve very high reliability. For some configurations including my strawman configuration, a N+1-parity disk array with on-line spare disks achieves higher reliability than the more expensive mirrored disk array.

As more and more reliability is required of more and more general purpose computer systems, reliability-cost tradeoffs will become critical. The models and design implications developed in this chapter will enable secondary storage system designers to achieve reliability goals with cost-effective redundant disk array solutions.

CHAPTER 6

Conclusions

It is my thesis that the burgeoning demand for reliable, parallel secondary storage can and will be met by redundant disk arrays. In support of my thesis, this dissertation makes three principal contributions. First, I argue that redundant arrays of small-diameter (inexpensive) disks are the heir apparent of secondary storage. Second, I provide a broad understanding of alternative redundant data encodings and of the relationship between parallelism, redundancy, and performance in disk arrays. Finally, I make possible the three-way optimization of cost, performance, and reliability for I/O systems designers by estimating analytically the reliability of redundant disk arrays.

The ascent of redundant arrays of magnetic disks is the inevitable result of three trends in computer technology. First, magnetic disk technology continues to dominate secondary storage systems, and the premier and most populous classes of magnetic disks are becoming physically smaller and lower in capacity. Capitalizing on this trend, disk arrays satisfy growing demands for capacity with increasing numbers of the technologically superior and more cost-competitive disks with smaller diameters. Second, the improvement in computational power of each new computer generation over that of its predecessor far surpasses the performance improvement of each new generation of magnetic disks over that of its predecessor. Disk arrays counter this growing gap by exploiting the parallelism inherent in their large number of component disks. Third, as society becomes irreversibly more dependent upon computerized mechanisms, its need for trustworthy systems intensifies. Disk arrays fulfill these needs cost-effectively by broadening the flexibility available to systems designers for balancing the cost of redundant data against the reliability induced by that redundancy.

Disk arrays achieve a high level of performance when many disks operate in parallel. Although there are some applications specially designed to redistribute data among disks dynamically so that user accesses evenly load all disks, the vast majority of systems do not have effective provisions for adjusting their allocations of data according to disk utilization. A more general approach to overcoming this problem in disk arrays employs disk striping, a process by which each user file is broken into relatively small pieces that are automatically allocated evenly among multiple disks. In this way, a large number of small accesses will apply a uniformly distributed load on the array's disks, and a single large access will transfer data from all disks in parallel. Although the performance of striped disk arrays is sensitive to the size of the pieces into which each file is subdivided, reasonable choices, such as the size of a disk track, deliver a large fraction of the disk array's potential performance.

Both performance and reliability are affected by encoding data in a redundant disk array. The simplest and least expensive encoding, called N+1-parity, stores data equivalent to the capacity of N data disks and its bitwise exclusive-or into a parity group of N+1 disks. Because catastrophic disk failures are self-identifying, a code that is single-error detecting, such as parity, provides single-erasure correction. In its simplest form, the disks of a parity group act as a single, virtual disk. But this sacrifices the potential to execute more than one small access on different disks of the parity group in parallel. Greater access rates can be achieved if each disk in a parity group is operated independently. In this case the group's parity information should be distributed evenly across all disks so that each bears an equal portion of the redundancy-maintenance overhead. For relatively long sequential accesses, this overhead is minimal, but for random writes of a small amount of data, this overhead can be as large as three times the amount of work done in a non-redundant disk array. Fortunately, new research into file systems with large caches promises to group random writes into large sequential ones, effectively avoiding redundancy-maintenance overheads.

In arrays with very large numbers of disks or in disk arrays with very high reliability requirements, the single-erasure correction provided by N+1-parity may not be sufficient; double-erasure correcting codes may be necessary instead. For very large disk arrays, a two dimensional N+1-parity encoding promises extremely high reliability. The penalty to performance of a two-dimensional parity encoding is a minimal redundancy-maintenance overhead that is at most five times the amount of work done in a non-redundant disk array. In smaller disk arrays with very high reliability requirements, non-binary codes can satisfy reliability requirements with only two disks of redundant information and the same minimal redundancy-maintenance overhead of two-dimensional parity.

Users' confidence that their data will not be lost rests on secondary storage more than on most other parts of a computer system. This confidence, in other words, is the true measure of data reliability. Its importance is comparable to the importance of cost and performance in secondary storage systems and should be an integral part of the system's design. Expanding the analysis of cost-performance tradeoffs to incorporate data reliability requires simple reliability estimates that operate on architectural parameters. In this dissertation, I develop models of the reliability of disk arrays that employ an N+1-parity encoding for redundant data. Based on Markov models of component failure and repair, my models take into account dependent failure modes arising from failures in shared support hardware. These dependent failure modes defeat data redundancy unless the set of disks affected by a dependent failure, called a string, is orthogonal to the parity groups that overlap it. With these models, designers of disk arrays can bypass the tedious task of programming a reliability simulator and concentrate on their product's development decisions.

To provide a concrete example of the use of these models, I analyze the design of a "strawman" disk array that is an attractive alternative to IBM's top-end disk, the 3390. Summary results for this comparison are reported in Tables 1.1 and 5.2. In short, the strawman disk array exceeds the access and transfer throughput of the 3390 by up to a factor of 6 and 8, respectively, while vastly exceeding the reliability of the 3390 and promising a comparable or superior cost. In fact, with the addition of at most two strings of on-line spare disks, my strawman disk array with N+1-parity encoding can achieve higher reliability than conventional and more expensive mirrored disk arrays.

The number of products offered in the disk array marketplace has begun to swell during the course of my research, and many other organizations have exploratory projects underway. This trend, coupled with the strong interest in disk arrays expressed by users whose needs emphasize business and scientific computing, is already fulfilling some of the tenets of my thesis. As a result of these encouraging signs, I am confident that future computer systems will boast reliable, parallel secondary storage based on redundant disk arrays.

Now that I have reviewed this dissertation, let me speculate about topics of future research. Because redundant disk arrays are a relatively new topic for research, their interaction with operating systems, network architectures, and archival systems is not yet fully understood. Additionally, there is ample opportunity for researchers to optimize the performance and reliability of these arrays. The issues that follow are a sampling of my personal concerns and interests.

One of the forces causing rapid increases in the computational power of new computer systems is multiprocessing. Collectively, the processors in a massively parallel machine create a large demand on secondary storage. One way to satisfy this demand, the approach taken by Berkeley's RAID prototypes, is to build a high performance server for secondary storage that employs redundant disk arrays and to locate it on a fast, central network. An alternative model, particularly appropriate where each node in the parallel machine is a general purpose processing system, is to connect a few disks to each node. This superimposition of a disk array onto a multiprocessor has advantages and disadvantages. On the positive side, it provides large aggregate computational power for controlling the disk array and data transfer bandwidth for accessing the disk array. On the negative side, it has a non-homogeneous distribution of data relative to a particular processor; if each processor's pattern of data access does not favor the disks attached to it, then significant computational and communication resources may be spent dynamically redistributing data. Hence, methods for effectively integrating I/O into massively parallel machines deserves investigation.

The distribution of function among client operating system, disk array controller, and disk-embedded controller is an issue worthy of research. On one hand, operating systems or user programs may exploit detailed knowledge of and control over a disk array to globally optimize performance. On the other hand, trustworthiness of secondary storage is maximized by isolating the control of disk arrays behind a simple, explicit interface. From the viewpoint of market acceptance, this issue is more complex because a disk array may be judged inadequate if its host software poorly "optimizes" the array's performance. An important example of this problem is the location, capacity, and function of data caching. It is quite likely that there will be caches in each disk, each array controller, and each host operating system. How then should this storage be controlled and in what ratios should its capacity be allocated?

With redundant disk arrays, secondary storage designers are well-positioned to satisfy rapidly growing demands for performance and reliability. But what about tertiary storage? The devices currently employed by tertiary storage systems are significantly slower and usually more manual than those in secondary storage. As a starting point, Stonebraker and Schloss have explored one way to adapt disaster recovery procedures to much larger on-line storage systems [Stonebraker90b]; further questions might be: will backup procedures be needed less often or must they become better integrated into the normal operation of the computer system?

There remain many issues related to reliability on my agenda. Muntz and Lui have begun examining the interaction between reliability and performance during the recovery of a failed disk's data [Muntz90]. This function may be amenable to optimization both at the level of the disk-embedded controller and at the level of the operating system's disk scheduling. More relevant to the specific material in this dissertation, however, is the need for more redundant disk array modeling. Partially populated strings of spares, non-immediate reorder policies, double-erasure-correcting codes, spare pools shared over multiple arrays, and more detailed hardware and software failure modes are all candidates for additional modeling.

Bibliography

[Abu-Sufah86] Walid Abu-Sufah, Harlan E. Husmann, David J. Kuck, "On Input/Output Speedup in Tightly Coupled Multiprocessors," *IEEE Transactions on Computers*, Volume C-35 (6), June 1986, pp 520-530.

[ADS85] Adaptive Data Systems, Inc., *SCSI Guidebook, Second Edition*, Adaptive Data Systems, Pomona CA, June 1985.

[Ahearn72] G. R. Ahearn, Y. Dishon, R. N. Snively, "Design Innovations of the IBM 3830 and 2835 Storage Control Units," *IBM Journal of Research and Development*, January 1972, pp 11-18.

[Allan83] I. D. Allan, "The Role of the Intelligent Peripheral Interface in Systems Architecture," *American Federation of Information Processing Societies (AFIPS) Joint Computer Conference Proceedings*, 1983.

[Amdahl67] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," *American Federation of Information Processing Societies (AFIPS) Joint Computer Conference Proceedings*, Volume 30, Atlantic City NJ, April 1967, pp 483- 485.

[AnonEtAl85] Anon et al., "A Measure of Transaction Processing Power," *Datamation*, Volume 31 (7), April 1985, pp 112-118, (also Tandem Technical Report 85.2, Tandem Corporation, 1985).

[ANSI86] *American National Standard for Information Systems - Small Computer System Interface (SCSI)*, ANSI X3.131-1986, New York NY, June 1986.

[ANSI87] *American National Standard for Information Systems – Intelligent Peripheral Interface – Device-Generic Command Set for Magnetic and Optical Disk Drives*, ANSI X3.132-1987, New York NY, December 1986.

[Arazi88] Benjamin Arazi, *A Commonsense Approach to the Theory of Error Correcting Codes*, MIT Press Series in Computer Systems, Herb Schwetman (Ed.), MIT Press, 1988.

[Arnould89] Emmanuel A. Arnould, Francois J. Bitz, Eric C. Cooper, H. T. Kung, Robert D. Sansom, Peter A. Steenkiste, "The Design of Nectar: A Network Backplane for Heterogeneous Multicomputers," *Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, Boston MA, April 1989, pp 205-216.

[Arthurs81] E. Arthurs, B. W. Stuck, "A Theoretical Reliability Analysis of a Single Machine with One Cold Standby Machine and One Repairman," *Proceedings of the 8th International Symposium on Computer Performance Modeling, Measurement, and Evaluation (PERFORMANCE '81)*, F. J. Kylstra (Ed.), North-Holland, November 1981, pp 479-488.

[Arulpragasam80] J. A. Arulpragasam, R. S. Swarz, "A Design for State Preservation on Storage Unit Failure," *The 10th Annual International Symposium on Fault-Tolerant Computing (FTCS-10)*, IEEE Computer Society, October 1980, pp 47-52.

[Astrahan57] M. M. Astrahan, B. Housman, J. F. Jacobs, R. P. Mayer, W. H. Thomas, "Logical Design of the Digital Computer in the SAGE System," *IBM Journal of Research and Development*, Volume 1 (1), January 1957, pp 76-83.

[ATC90] *Product Description, RAID+ Series Model RX*, Array Technology Corporation, Revision 1.0, Array Technology Corporation, Boulder CO, February 1990.

[Avizienis78] Algirdas Avižienis, "Fault-Tolerance: The Survival Attribute of Digital Systems," *Proceedings of the IEEE*, Volume 66 (10), October 1978, pp 1109-1125.

[Bailey91] Michael J. Bailey, "Scientific Visualization for a Large, Remotely-Distributed User Community," *Graphicon Conference Proceedings*, February 1991, pp 11-26.

[Balanson88] Richard Balanson, Presentation to the IBM Fellowship Conference, San Jose CA, November 1988.

[Bates89] K. H. Bates, "Performance Aspects of the HSC Controller," *Digital Technical Journal*, Volume 8, February 1989.

[Bell85] C. Gordon Bell, "Multis: A New Class of Multiprocessor Computers," *Science*, Volume 228, April 1985, pp 462-467.

[Bell89] C. Gordon Bell, "The Future of High Performance Computers in Science and Engineering," *Communications of the ACM*, Volume 32 (9), September 1989, pp 1091-1101.

[Beretvas78] T. Beretvas, "Performance Tuning in OS/VS2 MVS," *IBM Systems Journal*, Volume 17 (3), 1978, pp 290-313.

[Bhat72] U. N. Bhat, *Elements of Applied Stochastic Processes*, John Wiley & Sons, 1972.

[Billmers84] M. A. Billmers, M. W. Swartwout, "AI-SPEAR: Computer System Failure Analysis Tool," *Proceedings of the Sixth European Conference on Artificial Intelligence*, Pisa Italy, September 1984, pp 65-75.

[Bitton88] Dina Bitton, Jim Gray, "Disk Shadowing," *Proceedings of the 14th International Conference on Very Large Data Bases (VLDB)*, 1988, pp 331-338.

[Bodega89] National Science Foundation Workshop on Next Generation Secondary Storage Architecture, Bodega Bay CA, May 1989 (unpublished).

[Bohl81] Marilyn Bohl, *Introduction to IBM Direct Access Storage Devices*, Science Research Associates, Chicago, 1981.

[Boral83] H. Boral, D. DeWitt, "Database Machines: An Idea Whose Time Has Passed?," *Database Machines*, H. O. Leilich, M. Missikoff (Eds.), Springer-Verlag, September 1983.

[Bortz90] Alfred B. Bortz (Ed.), "Report of the National Science Foundation Workshop on Advanced Data Storage Technology for Computer Systems," Carnegie Mellon University, January 1990.

[Brady89] J. Brady, personal communications, July 1989.

[Breuer76] Melvin A. Breuer, Arthur D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Potomac MD, 1976.

[Brown72] D. T. Brown, R. L. Gibson, C. A. Thorn, "Channel and Direct Access Device Architecture," *IBM Systems Journal*, Volume 11 (3), 1972, pp 186-199.

[Bucher80] Ingrid Y. Bucher, Ann H. Hayes, "I/O Performance Measurement on CRAY-I and CDC 7600 Computers," *Proceedings of the 16th Computer Performance Evaluation Users Group (CPEUG)*, NBS 500-65, October 1980, pp 245-254.

[Burger84] Robert M. Burger, Ralph K. Calvin III, William C. Holton, Larry W. Sumney, "The Impact of ICs on Computer Technology," *IEEE Computer*, October 1984, pp 88-95.

[Burks46] A. W. Burks, H. H. Goldstine, J. von Neumann, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," *Papers of John von Neumann*, W. Aspray, A. Burks (Eds.), MIT Press, 1987, pp 97-146.

[Buzen75] Jeffrey P. Buzen, "I/O Subsystem Architecture," *Proceedings of the IEEE*, Volume 63 (6), June 1975.

[Buzen82] Jeffrey P. Buzen, Anneliese von Mayrhauser, "BEST/1 Analysis of the IBM 3880-13 Cached Storage Controller," *International Conference on Management and Performance Evaluation of Computer Systems (CMG) XIII Proceedings*, Computer Measurement Group, Chicago IL, 1982, pp 156-173.

[Buzen86] Jeffrey P. Buzen, Annie W.C. Shum, "I/O Architecture in MVS/370 and MVS/XA," *CMG Transactions*, Volume 54, Computer Measurement Group, Chicago IL, Fall 1986.

[Buzen87] Jeffrey P. Buzen, A. W. Shum, "A Unified Operational Treatment of RPS Reconnect Delays," *Proceedings of the 1987 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Performance Evaluation Review*, Volume 15 (1), May 1987.

[CDC88] Control Data Corporation, *Product Specification for WREN IV SCSI Model 94171-344*, Control Data OEM Product Sales, Minneapolis MN, January 1988.

[Chen89] Peter M. Chen, "An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890," University of California Technical Report UCB/CSD 89/506, Berkeley CA, May 1989, Master's Thesis.

[Chen90a] Peter M. Chen, Garth A. Gibson, Randy H. Katz, David A. Patterson, "An Evaluation of Redundant Arrays of Disks Using an Amdahl 5890," *Proceedings of the 1990 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Boulder CO, May 1990.

[Chen90b] Peter M. Chen, David A. Patterson, "Maximizing Performance in a Striped Disk Array," *Proceedings of the 17th Annual International Symposium of Computer Architecture (SIGARCH)*, Seattle WA, May 1990, pp 322-331.

[Chervenak90] Ann L. Chervenak, "Performance Measurements of the First RAID Prototype," University of California, Technical Report UCB/CSD 90/574, Berkeley CA, May 1990, Master's Thesis.

[Chervenak91] Ann L. Chervenak, Randy H. Katz, "Performance of a RAID Prototype," *Proceedings of the 1991 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, May 1991.

[Codd60] E. F. Codd, "Multiprogram Scheduling," *Communications of the ACM*, Volume 3 (6), June 1960.

[Computer90] "Gigabit Network Testbeds," *IEEE Computer*, Volume 23 (9), September 1990, Special Report.

[Conner3100] Conner Peripherals Inc., *CP3100 Engineering Product Specification*, Preliminary Revision III, San Jose CA.

[Cormier83] R. L. Cormier, R. J. Dugan, R. R. Guyette, "System/370 Extended Architecture: The Channel Subsystem," *IBM Journal of Research Development*, Volume 27 (3), May 1983.

[Cray88] Cray Research Inc., *CTF77 Reference Manual*, Release B, SR-0018, Mendota Heights MN, February 1988.

[Denning67] P. J. Denning, "Effects of Scheduling on File Memory Operations," *American Federation of Information Processing Societies (AFIPS) Joint Computer Conference Proceedings*, Volume 30, 1967.

[Denning70] P. J. Denning, "Virtual Memory," *ACM Computing Surveys*, Volume 2, September 1970, pp 153-190.

[Denning80] P. J. Denning, "Working Sets Past and Present," *IEEE Transactions on Software Engineering*, Volume SE-6 (1), January 1980, pp 64-84.

[Deodhar83] Shirish Deodhar, E. J. Weldon, "High Speed Interleaved Reed-Solomon Error Detection and Correction System," *Journal of the Society of Photo-optical Instrumentation Engineers (SPIE)*, Volume 421, 1983.

[Devlin90] Phil Devlin, Presentation to the Disk Array Forum, September 1990.

[Dishon88] Yitzhak Dishon, T. S. Liu, ''Disk Dual Copy Methods and Their Performance,'' *The 18th Annual International Symposium on Fault-Tolerant Computing (FTCS-18)*, IEEE Computer Society Press, 1988, pp 314-319.

[Ditzel80] David R. Ditzel, ''Program Measurements on a High-Level Language Computer,'' *IEEE Computer*, August 1980, pp 62-72.

[Dolotta76] T. A. Dolotta, M. I. Bernstein, R. S. Dickson Jr., N. A. France, B. A. Rosenblatt, D. M. Smith, T. B. Steel Jr., *Data Processing in 1980-1985*, Wiley, New York, 1976.

[Effelsberg84] Wolfgang Effelsberg, Theo Haerder, ''Principles of Database Buffer Management,'' *ACM Transactions on Database Systems*, Volume 9 (4), December 1984, pp 560-595.

[Eggers89] Susan J. Eggers, *Simulation Analysis of Data Sharing in Shared Memory Multiprocessors*, PhD Dissertation, University of California, Technical Report UCB/CSD 89/501, April 1989.

[EET89] *Electronic Engineering Times*, June 12, 1989, pp 51.

[Emlich89] Larry W. Emlich, Herman D. Polich, ''VAXsimPLUS, A Fault Manager Implementation,'' *Digital Technical Journal*, Volume 8, February 1989.

[Feierbach79] G. Feierbach, D. Stevenson, ''The Illiac-IV,'' *Infotech State of the Art Report on Supercomputers*, Maidenhead, England, data also appears in *Computer Structures: Principles and Examples*, D. P. Siewiorek, C. G. Bell, A. Newell (Eds.), McGraw-Hill, 1982, pp 268-269, and in John L. Hennessy, David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 1990, pp 554-555.

[Feiertag71] R. J. Feiertag, E. I. Organick, ''The Multics Input/Output System,'' *Proceedings of the Third ACM Symposium on Operating System Principles (SOSP)*, 1971, pp 35-41.

[Frank87] F. D. Frank, ''Advances in Head Technology,'' Challenges in Winchester Technology course notes, Santa Clara University, December 1987.

[Friedman83] M. B. Friedman, ''DASD Access Patterns,'' *International Conference on Management and Performance Evaluation of Computer Systems (CMG) XIV Proceedings*, Computer Measurement Group, Chicago IL, 1983.

[Fujitsu2351] Fujitsu Corporation, *M2351A: Mini-Disk Drive Engineering Specifications*, B03P-4655-0002A, November 1984.

[Fujitsu2361] Fujitsu Corporation, *M2361A: Mini-Disk Drive Engineering Specifications*, B03P-4825-0001A, February 1987.

[Garcia-Molina84] Hector Garcia-Molina, Richard J. Lipton, Jacobo Valdes, "A Massive Memory Machine," *IEEE Transactions on Computers*, Volume C-33 (5), May 1984, pp 391-399.

[Garcia-Molina88] Hector Garcia-Molina, Kenneth Salem, "The Impact of Disk Striping on Reliability," *IEEE Data Engineering Bulletin*, Volume 1 (2), 1988.

[Geist82] Robert M. Geist, Kishor S. Trivedi, "Optimal Design of Multilevel Storage Hierarchies," *IEEE Transactions on Computers*, Volume C-31 (3), March 1982, pp 249-260.

[Geist87] Robert M. Geist, Stephen Daniel, "A Continuum of Disk Scheduling Algorithms," *ACM Transactions on Computer Systems*, Volume 5 (1), February 1987, pp 77-92.

[Geist90] Robert Geist, Kishor Trivedi, "Reliability Estimation of Fault-Tolerant Systems: Tools and Techniques," *IEEE Computer*, Volume 23 (7), July 1990, pp 52-61.

[Gelb89] Jack P. Gelb, "An Overview of System-Managed Storage," *IBM Systems Journal*, Volume 28 (1), 1989, pp 77-103.

[Gelsinger89] Patrick P. Gelsinger, Paola A. Gargini, Gerhard H. Parker, Albert Y. C. Yu, "Microprocessors Circa 2000," *IEEE Spectrum*, October 1989, pp 43-74.

[Gibson87] Garth A. Gibson, "Estimating Performance of Single Bus, Shared Memory Multiprocessors," University of California, Technical Report UCB/CSD 87/355, May 1987, Master's Thesis.

[Gibson89a] Garth A. Gibson, "SpurBus Specification," University of California, Technical Report UCB/CSD 88/480, March 1989.

[Gibson89b] Garth A. Gibson, Lisa Hellerstein, Richard M. Karp, Randy H. Katz, David A. Patterson, "Coding Techniques for Handling Failures in Large Disk Arrays," *Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, Boston MA, April 1989, pp 123-132.

[Gibson89c] Garth A. Gibson, "Reliability and Performance in Redundant Arrays of Magnetic Disks," *International Conference on Management and Performance Evaluation of Computer Systems (CMG) XX Proceedings*, Computer Measurement Group, Chicago IL, December 1989.

[Goldstein87] S. Goldstein, "Storage Performance – An Eight-Year Outlook," IBM Technical Report 03.308-1, Second Edition, IBM Santa Teresa Lab, San Jose, October 1987.

[Gray90] Jim Gray, Bob Horst, Mark Walker, "Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput," *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB)*, August 1990, pp 148-161.

[Glover88] Neal Glover, "Prologue - The Coming Revolution in Error Correction Technology," *ENDL's 1988 Disk/Test Conference*, also in *Practical Error Correction Design For Engineers, Second Edition*, Neal Glover, Trent Dudley, Data Systems Technology, Broomfield CO, 1988.

[Grossman85] C. P. Grossman, "Cache-DASD Storage Design for Improving System Performance," *IBM Systems Journal*, Volume 24 (3/4), 1985, pp 316-334.

[Gustafson89] John L. Gustafson, "Bridging the Gap Between Amdahl's Law and Sandia-Laboratory's Result," *Communications of the ACM*, Volume 32, August 1989, pp 1015-1016.

[Harker81] J. M. Harker, D. W. Brede, R. E. Pattison, G. R. Santana, L. G. Taft, "A Quarter Century of Disk File Innovation," *IBM Journal of Research and Development*, Volume 25 (5), September 1981, pp 677-689.

[Hill86] M. D. Hill, S. J. Eggers, J. R. Larus, G. S. Taylor, G. Adams, B. K. Bose, G. A. Gibson, P. M. Hansen, J. Keller, S. I. Kong, C. G. Lee, D. Lee, J. M. Pendleton, S. A. Ritchie, D. A. Wood, B. G. Zorn, P. N. Hilfinger, D. Hodges, R. H. Katz, J. Ousterhout, D. A. Patterson, "Design Decisions in SPUR," *IEEE Computer*, Volume 19 (11), November 1986, also in *Computers for Artificial Intelligence Processing*, B. W. Wah, C. V. Ramamoorthy (Eds.), John Wiley & Sons, 1990, pp 273-299.

[Hill87] Mark Donald Hill, *Aspects of Cache Memory and Instruction Buffer Performance*, PhD Dissertation, University of California, Technical Report UCB/CSD 87/381, November 1987.

[Hoagland85] A. S. Hoagland, "Information Storage Technology: A Look at the Future," *IEEE Computer*, Volume 18, July 1985, pp 60-67.

[Hoagland88] A. S. Hoagland, "Overview of Magnetic Storage Technology Trends," *Systems Design and Networks Conference Proceedings, Mass Storage Trends and Systems Integration*, Kenneth Majithia (Ed.), IEEE Bay Area Council, Santa Clara CA, April 1988, pp 1-7.

[Hoagland89] A. S. Hoagland, "Magnetic and Optical Data Storage Trends," presented at *NSF Workshop on Next Generation Secondary Storage Architecture*, Bodega Bay CA, May 1989 (unpublished).

[Hodges77] David A. Hodges, "Microelectronic Memories," *Scientific American*, Volume 237 (3), September 1977, pp 130-145.

[Hodges90] David A. Hodges, personal communications, November 1990.

[Houtekamer85] G. Houtekamer, "The Local Disk Controller," *Proceedings of the 1985 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, August 1985.

[IBM80] IBM Corporation, *Disk Storage Technology*, GA 26-1665, February 1980.

[IBM3380] IBM Corporation, *IBM 3380 Direct Access Storage Introduction*, Manual GC26-4491-0, September 1987.

[IBM3390] IBM Corporation, *IBM 3390 Direct Access Storage Introduction*, Manual GC26-4573-0, August 1989.

[IBM0661] IBM Corporation, *IBM 0661 Disk Drive Product Description Model 371, First Edition*, Low End Storage Products, 504/114-2, July 1989.

[IBMAS400] IBM Corporation, *AS/400 Programming: Backup and Recovery Guide*, SC21-8079-0, June 1988.

[Jeong89] Deog-Kyoon Jeong, David A. Wood, Garth A. Gibson, Susan J. Eggers, David A. Hodges, Randy H. Katz, David A. Patterson, "A VLSI Chip Set for a Multiprocessor Workstation - Part II: A Memory Managment Unit and Cache Controller," *IEEE Journal of Solid-State Circuits,* Volume 24 (6), December 1989.

[Jilke86] W. Jilke, "Disk Array Mass Storage Systems: The New Opportunity," Amperif Corporation, September 1986.

[Johnson84] O. G. Johnson, "Three-Dimensional Wave Equation Computations on Vector Computers," *Proceedings of the IEEE,* Volume 72 (1), January 1984.

[Kannan78] Krishnamurthi Kannan, "The Design of a Mass Memory for a Database Computer," *Proceedings of the Fifth Annual Symposium on Computer Architecture (SIGARCH),* Palo Alto CA, April 1978.

[Kaplan58] E. L. Kaplan, Paul Meier, "Nonparametric Estimation from Incomplete Observations," *American Statistical Association Journal,* Volume 53, June 1958, pp 457-481.

[Katz89a] Randy H. Katz, John K. Ousterhout, David A. Patterson, Peter M. Chen, Ann L. Chervenak, Rich Drewes, Garth A. Gibson, Ed K. Lee, Ken C. Lutz, Ethan L. Miller, Mendel Rosenblum, "A Project on High Performance I/O Subsystems," *ACM Computer Architecture News,* Volume 17 (5), September 1989, pp 24-31.

[Katz89b] Randy H. Katz, G. A. Gibson, D. A. Patterson, "Disk System Architectures for High Performance Computing," *Proceedings of the IEEE,* Volume 77 (12), December 1989, pp 1842-1858.

[Katz90] Randy H. Katz, David W. Gordon, James A. Tuttle, "Storage System Metrics for Evaluating Disk Array Organizations," University of California Technical Report UCB/CSD 90/611, Berkeley CA, December 1990.

[Katzman77] J. A. Katzman, "System Architecture for Nonstop Computing," *14th IEEE Computer Society International Conference (COMPCON 77),* February 1977.

[Kim85] Michelle Y. Kim, A. M. Patel, "Error-Correcting Codes for Interleaved Disks with Minimal Redundancy," IBM Computer Science Research Report, RC11185 (50403), May 1985.

[Kim86] Michelle Y. Kim, "Synchronized Disk Interleaving," *IEEE Transactions on Computers*, Volume C-35 (11), November 1986.

[Kim87a] Michelle Y. Kim, Anil Nigam, George Paul, Robert J. Flynn, "Disk Interleaving and Very Large Fast Fourier Transforms," *International Journal of Supercomputer Applications*, Volume 1 (3), MIT Press, Fall 1987, pp 75-96.

[Kim87b] Michelle Y. Kim, *Synchronously Interleaved Disk Systems with Their Application to the Very Large Fast Fourier Transform*, PhD Dissertation, Polytechnic University, Janurary 1987.

[King87] Richard P. King, "Disk Arm Movement in Anticipation of Future Requests," IBM Computer Science Research Report, December 1985.

[Kirk90] R. E. Kirk, *Statistics, An Introduction, Third Edition*, Holt, Rinehart and Winston, 1990.

[Klietz88] A. Klietz, J. Turner, T.C. Jacobson, "TurboNFS: Fast Shared Access for Cray Disk Storage," *Proceedings of the Cray User Group Convention*, April 1988.

[Knuth71] D. E. Knuth, "An Empirical Study of FORTRAN Programs," *Software - Practice & Experience*, Volume 1 (2), 1971, pp 105-134.

[Koch87] Philip D. L. Koch, "Disk File Allocation Based on the Buddy System," *ACM Transactions on Computer Systems*, Volume 5 (4), November 1987, pp 352-370.

[Kryder89] Mark H. Kryder, "Data Storage in 2000 - Trends in Data Storage Technologies," *IEEE Transactions on Magnetics*, Volume 25 (6), November 1989.

[Kung86] H. T. Kung, "Memory Requirements for Balanced Computer Architecture," *Proceedings of the 13th Annual International Symposium of Computer Architecture (SIGARCH)*, June 1986.

[Lary89] Richard Lary, "VAXsimPLUS," presented at *NSF Workshop on Next Generation Secondary Storage Architecture*, Bodega Bay CA, May 1989 (unpublished).

[Lawless82] J. F. Lawless, *Statistical Models and Methods for Lifetime Data*, Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons, 1982.

[Lazowska86] Edward D. Lazowska, John Zahorjan, David R. Cheriton, Willy Zwaenepoel, "File Access Performance of Diskless Workstations," *ACM Transactions on Computer Systems*, Volume 4 (3), August 1986, pp 238-268.

[Lee90] Edward K. Lee, "Software and Performance Issues in the Implementation of a RAID Prototype," University of California, Technical Report UCB/CSD 90/573, Berkeley CA, May 1990, Master's Thesis.

[Lee91] Edward K. Lee, Randy H. Katz, "Performance Consequences of Parity Placement in Disk Arrays," *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, Palo Alto CA, April 1991.

[Lin88] Ting-Ting Yao Lin, *Design and Evaluation of an On-Line Predictive Diagnostic System*, PhD Dissertation, Carnegie Mellon University, April 1988.

[Lin90] Ting-Ting Y. Lin, Daniel P. Siewiorek, "Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis," *IEEE Transactions on Reliability*, Volume 39 (4), October 1990.

[Liptay68] J. S. Liptay, "Structural Aspects of the System/360 Model 85, Part II: The Cache," *IBM Systems Journal*, Volume 7 (1), 1968, pp 15-21.

[Livny87] M. Livny, S. Khoshafian, H. Boral, "Multi-disk Management Algorithms," *Proceedings of the 1987 ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, May 1987.

[Lomas91] Luis Lomas, personal communications, January 1991.

[Lynch72] William C. Lynch, "Do Disk Arms Move?" *Performance Evaluation Review*, Volume 1 (4), December 1972, pp 3-16.

[MacWilliams77] Florence Jessie MacWilliams, Neil James Alexander Sloane, *The Theory of Error-Correcting Codes*, North-Holland Mathematical Library, Volume 16, Elsevier Science Publishing Company, New York NY, 1977.

[Massiglia86] Paul Massiglia, *Digital Large System Mass Storage Handbook*, Digital Equipment Corporation, 1986.

[Matloff87] Norman S. Matloff, "A Multiple-Disk System for Both Fault Tolerance and Improved Performance," *IEEE Transaction on Reliability*, Volume R-36 (2), June 1987, pp 199-201.

[McKusick83] Marshall Kirk McKusick, William N. Joy, Samual J. Leffler, Robert S. Fabry, "A Fast File System for UNIX," *ACM Transactions on Computer Systems*, Volume 2 (3), August 1984, pp 181-197.

[McNutt86] Bruce McNutt, "An Empirical Study of Variations in DASD Volume Activity," *International Conference on Management and Performance Evaluation of Computer Systems (CMG) XVII Proceedings*, Computer Measurement Group, Chicago IL, 1986, pp 274-283.

[Menon90] Jai Menon, Jim Kasson, "Methods for Improved Update Performance of Disk Arrays," IBM Technical Report RJ6928 (66034), November 1990.

[Miller91] Ethan L. Miller, "Input/Output Behavior of Supercomputing Applications," University of California, Technical Report UCB/CSD 91/616, January 1991, Master's Thesis.

[Mitoma89] Mike Mitoma, personal communications, 1989.

[Mitoma90] Mike Mitoma, personal communications, 1990.

[Moore75] Gordon E. Moore, "Progress in Digital Integrated Electronics," *IEEE Digest International Electron Devices Meeting*, 1975, pp 11.

[Moore79] Gordon E. Moore, "Are We Really Ready for VLSI~2?," *IEEE Digest International Solid-State Circuits Conference*, 1979, pp 54-55.

[Muntz90] Richard R. Muntz, John C. S. Lui, "Performance Analysis of Disk Arrays Under Failure," *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB)*, Dennis McLeod, Ron Sacks-Davis, Hans Schek (Eds.), Morgan Kaufmann Publishers, August 1990, pp 162-173.

[Myers86] Glenford J. Myers, Albert Y. C. Yu, David L. House, "Microprocessor Technology Trends," *Proceedings of the IEEE*, Volume 74 (12), December 1986.

[Nelson88] Mike N. Nelson, Brent B. Welch, John K. Ousterhout, "Caching in the Sprite Network File System," *ACM Transactions on Computer Systems*, Volume 6 (1), February 1988.

[Nelson90] Victor P. Nelson, "Fault-Tolerant Computing: Fundamental Concepts," *IEEE Computer*, Volume 23 (7), July 1990, pp 19-25.

[Ng90] Spencer W. Ng, "Sparing for Redundant Disk Arrays," IBM Almaden Research Lab presentation, July 1990.

[Ng91] Spencer W. Ng, "Improving Disk Performance Via Latency Reduction," *IEEE Transactions on Computers*, Volume 40 (1), January 1991, pp 22-30.

[Ousterhout85] John K. Ousterhout, Herve Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, J. G. Thompson, "A Trace-Driven Analysis of the UNIX 4.2 BSD File System," *Proceedings of the Tenth ACM Symposium on Operating System Principles (SOSP), ACM Operating Systems Review*, Volume 19 (5), December 1985, pp 15-24.

[Ousterhout88] John K. Ousterhout, Andrew R. Cherenson, Frederick Douglis, Michael N. Nelson, Brent B. Welch, "The Sprite Network Operating System," *IEEE Computer*, February 1988.

[Ousterhout89] John K. Ousterhout, Fred Douglis, "Beating the I/O Bottleneck: A Case for Log-Structured File Systems," *ACM Operating Systems Review*, Volume 23 (1), January 1989, pp 11-28.

[Park86] A. Park, K. Balasubramanian, "Providing Fault Tolerance in Parallel Secondary Storage Systems," Princeton Technical Report CS-TR-057-86, November 1986.

[Patel82] A. M. Patel, "Error and Failure-Control for a Large-Size Bubble Memory," *IEEE Transactions on Magnetics*, Volume MAG-18 (6), November 1982.

[Patterson87] David A. Patterson, Garth A. Gibson, Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," University of California, Technical Report UCB/CSD 87/391, Berkeley CA, December 1987, preprint of [Patterson88].

[Patterson88] David A. Patterson, Garth A. Gibson, Randy H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proceedings of the 1988 ACM Conference on Management of Data (SIGMOD)*, Chicago IL, June 1988, pp 109-116.

[Patterson90] John L. Hennessy, David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 1990.

[Peters87] R. Peters, personal communications regarding the University of California at Berkeley's Computing Center, September 1987.

[Peterson72] W. Wesley Peterson, E. J. Weldon, Jr., *Error-Correcting Codes, Second Edition*, MIT Press, 1972.

[Poston88] A. Poston, "A High Performance File System for UNIX," NASA NAS document, June 1988.

[Powell77] Michael L. Powell, "The DEMOS File System," *Proceedings of the Sixth ACM Symposium on Operating System Principles (SOSP)*, November 1977, pp 33-42.

[Pugh71] E. W. Pugh, "Storage Hierarchies: Gaps, Cliffs, and Trends," *IEEE Transactions on Magnetics*, Volume MAG-7, December 1971, pp 810-814.

[Rao89] Thammavarapu R. N. Rao, Eiji Fujiwara, *Error-Control Coding for Computer Systems*, Prentice Hall Series in Computer Engineering, Edward J. McCluskey (Ed.), Prentice Hall, 1989.

[Reddy89] A. L. Narasimha Reddy, Prithviraj Banerjee, "Evaluation of Multiple-Disk I/O Systems," *IEEE Transactions on Computers*, December 1989.

[Rosenblum90] Mendel Rosenblum, John K. Ousterhout, "The LFS Storage Manager," *Proceedings of the Summer 1990 USENIX Technical Conference*, Anaheim CA, June 1990.

[Ross85] S. M. Ross, *Introduction to Probability Models, Third Edition*, Academic Press, Harcourt Brace Jovanovich, 1985.

[Sahner86] R. A. Sahner, K. S. Trivedi, "Sharpe: Symbolic Hierarchical Automated Reliability and Performance Evaluator, Introduction and Guide for Users," Department of Computer Science, Duke University, September 1986.

[Sahner87] R. A. Sahner, K. S. Trivedi, "Reliability Modeling using SHARPE," *IEEE Transactions on Reliability*, Volume R-36 (2), June 1987, pp 186-193.

[Salem86] K. Salem, H. Garcia-Molina, "Disk Striping," *Proceedings of the 2nd IEEE International Conference on Data Engineering*, 1986.

[Seagate90] Seagate Corp., *World Class Data Storage*, Seagate Publication 1000-006, 1990.

[Scheffler73] L. J. Scheffler, "Optimal Folding of a Paging Drum in a Three Level Memory System," *Proceedings of the Fourth ACM Symposium on Operating System Principles (SOSP), Operating Systems Review*, Volume 7 (4), 1973, pp 58-65.

[Schulze89] Martin E. Schulze, Garth A. Gibson, Randy H. Katz, David A. Patterson, "How Reliable is a RAID?" *Proceedings of the 1989 IEEE Computer Society International Conference (COMPCON 89)*, San Francisco CA, Spring 1989.

[Scranton83] R. A. Scranton, D. A. Thompson, D. W. Hunter, "The Access Time Myth," IBM Technical Report RC10197, 1983.

[Seltzer90a] Margo I. Seltzer, Peter M. Chen, John K. Ousterhout, "Disk Scheduling Revisited," *Proceedings of the Winter 1990 USENIX Technical Conference*, Washington DC, January 1990.

[Seltzer90b] Margo I. Seltzer, Michael Stonebraker, "Transaction Support in Read Optimized and Write Optimized File Systems," *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB)*, Dennis McLeod, Ron Sacks-Davis, Hans Schek (Eds.), Morgan Kaufmann Publishers, August 1990, pp 174-185.

[Serell62] R. Serrell, M. M. Astrahan, G. W. Patterson, I. B. Pyne, "The Evolution of Computing Machines and Systems," *Proceedings of the Institute of Radio Engineers (IRE)*, Volume 50 (5), May 1962, pp 1039-1058.

[Shannon48] Claude E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, Volume 27 (3), July 1948, pp 379-423, and (4) October 1948, pp 623-656.

[Shao88] Howard M. Shao, Irving S. Reed, "On the VLSI Design of a Pipeline Reed-Solomon Decoder Using Systolic Arrays," *IEEE Transactions on Computers*, Volume 37 (10), October 1988.

[Shebell85] J. P. Shebell, "Symptom Directed Diagnosis Foundations and Practices," *IEEE International Conference on Computer Design (ICCD): VLSI in Computers*, IEEE Computer Society Press, Piscataway NJ, July 1985, pp 290-293.

[Sierra90] Hugh M. Sierra, *An Introduction to Direct Access Storage Devices*, Academic Press, 1990.

[Siewiorek82] D. P. Siewiorek, R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, 1982.

[Slotnik70] D. L. Slotnik, "Logic per Track Devices," *Advances in Computers*, Volume 10, Frantz Alt (Ed.), Academic Press, 1970, pp 291-296, also *ACM Transactions on Database Systems*, Volume 1 (3), September 1976.

[Smith85] Alan Jay Smith, "Disk Cache - Miss Ratio Analysis and Design Considerations," *ACM Transactions on Computer Systems*, Volume 3 (3), August 1985.

[Sprott73] D. A. Sprott, "Normal Likelihoods and Relation to a Large Sample Theory of Estimation," *Biometrika*, Volume 60 (3), 1973, pp 457-465.

[Staelin90] Carl Staelin, Hector Garcia-Molina, "Clustering Active Disk Data To Improve Disk Performance," Princeton Technical Report CS-TR-283-90, September 1990.

[Stonebraker81] Michael R. Stonebraker, "Operating System Support for Database Management," *Communications of the ACM*, Volume 24 (7), July 1981.

[Stonebraker88] M. R. Stonebraker, R. H. Katz, D. A. Patterson, J. K. Ousterhout, "The Design of XPRS," *Proceedings of the 14th International Conference on Very Large Data Bases (VLDB)*, Los Angeles CA, August 1988.

[Stonebraker90a] M. R. Stonebraker, et. al., "The Implementation of POSTGRES," *IEEE Transactions on Knowledge and Data Engineering*, March 1990.

[Stonebraker90b] M. R. Stonebraker, G. A. Schloss, "Distributed RAID – A New Multiple Copy Algorithm," *Proceedings of the 6th IEEE International Conference on Data Engineering*, April 1990.

[Tang69] D. T. Tang, R. T. Chien, "Coding for Error Control," *IBM Systems Journal*, Volume 8 (1), 1969, pp 48-86, also in *The Theory and Practice of Reliable System Design*, D.P. Siewiorek, R.S. Swarz (Eds.), Digital Press, 1982, pp 639-669.

[Taylor86] G. S. Taylor, P. N. Hilfinger, J. R. Larus, D. A. Patterson, B. G. Zorn, "Evaluation of the SPUR Lisp Architecture," *Proceedings of the 13th Annual International Symposium of Computer Architecture (SIGARCH)*, June 1986.

[Tendolkar85] N. N. Tendolkar, D. C. Bossen, M. Y. Hsiao, "Design for On Line Diagnosis of Failures in Large Computer Systems," *IEEE International Conference on Computer Design (ICCD): VLSI in Computers*, IEEE Computer Society Press, Piscataway NJ, July 1985, pp 286-289.

[Thisquen88] J. Thisquen, "Seek Time Measurements," Amdahl Peripheral Products Division Technical Report, May 1988.

[Thornton64] James E. Thornton, "Parallel Operation in the Control Data 6600," *American Federation of Information Processing Societies (AFIPS) Joint Computer Conference Proceedings*, Volume 26 (Part 2), 1964, pp 33-40, also in *Computer Structures: Principles and Examples*, D. P. Siewiorek, C. G. Bell, A. Newell (Eds.), McGraw-Hill, 1982, pp 730-742.

[TMC87] Thinking Machines Corporation, *Connection Machine Model CM-2 Technical Summary*, Thinking Machines Technical Report HA87-4, Thinking Machines Corporation, April 1987.

[Toerey72] Toby J. Toerey, Tad B. Pinkerton, "A Comparative Analysis of Disk Scheduling Policies," *Communications of the ACM*, Volume 15 (3), March 1972, pp 177-184.

[Tsao83] M. M. Tsao, "Trend Analysis and Fault Prediction," Dept. of Electrical and Computer Engineering, Carnegie Mellon University Technical Report 130, Pittsburgh PA, May 1983.

[Warlaumont89] John Warlaumont, "X-ray Lithography: On the Path to Manufacturing," *Journal of Vacuum Science and Technology*, Volume 7 (6), Nov/Dec 1989.

[White84] Steven W. White, Noel R. Stader II, V. Thomas Thyne, "A VLSI-Based I/O Formatting Device," *IEEE Transactions on Computers*, Volume C-33 (2), February 1984.

[Wolff89] Ronald W. Wolff, *Stochastic Modeling and the Theory of Queues*, Prentice Hall, 1989.

[Wolff91] Ronald W. Wolff, personal communications, January 1991.

[Wolfram88] Stephen Wolfram, *Mathematica: A System for Doing Mathematics by Computer*, Addison-Wesley, 1988.

[Wood87] David A. Wood, Susan J. Eggers, Garth A. Gibson, "SPUR Memory System Architecture," University of California, Technical Report UCB/CSD 87/394, Berkeley CA, December 1987.

[Wood90a] David A. Wood, *The Design and Evaluation of In-Cache Address Translation*, PhD Dissertation, University of California, Technical Report UCB/CSD 90/565, March 1990.

[Wood90b] Roger Wood, "Magnetic Megabits," *IEEE Spectrum*, Volume 27 (5), May 1990.

[Yeack-Scranton90] C. E. Yeack-Scranton, "DRAM for DASD Applications?" Computer Mechanics seminar presentation, University of California, Berkeley CA, February 1990.

APPENDIX A

Reliability Simulation

A.1. Introducing RELI

An important tool in my disk array reliability analysis has been a detailed, event-driven simulator (apatheticly called RELI). RELI simulates a disk array by explicitly modeling a collection of parity groups imposed on a set of strings of disks. Disks and strings are modeled as operational or under repair; RELI does not model the workload on these components, nor does it model a host computer's operation (or the host's hardware and software failures).

The array begins fully stocked with on-line spares and operational data disks (all disks in a parity group are referred to as data disks because this simulator only need distinguish between spares and non-spares). Disk and string lifetimes are stochastically assigned, as are disk and string repair durations. The pool of on-line spares is replenished by the arrival of newly ordered disks explicitly ordered when the spare pool size falls below a specific threshold, or implicitly ordered by the arrival of a string repairperson.

The simulation proceeds from event to event. Events are primarily disk failures, string failures, string repairs, disk recoveries, or new disk arrivals. Simulation ends when at least one parity group has more than one (data) disk failed at the same time.

The simulator is instrumented to collect a variety of statistics, the most important being the array lifetime and its 1-, 3-, and 10-year reliabilities, but also metrics such as fraction of time with one or more disk in recovery, the average number of on-line spares, the average arrival rate of new disks, and the average recovery time.

Because lifetimes and repair durations are stochastically assigned, two invocations of RELI with all parameters identical except the random number generator seed will yield different array lifetime statistics. To provide confidence that the statistics reported are good estimations of their true averages across all sequences of disk and string lifetimes and repairs, RELI repeatedly simulates array lifetimes for each of many different random number sequences before reporting average statistics.

A.2. Parity Group Organization in a Set of Strings

Each disk in an array is connected to a particular string, and is either an on-line spare or belongs to a particular parity group. RELI can either cluster the data disks from one parity group on the same string or spread the disks of each parity group out over as many strings as possible. The former is called a *string parallel* mapping and the latter is called a *string orthogonal* mapping. The orthogonal mapping yields higher array lifetimes because string failures often involve no more than one disk from each parity group.

An array can accomodate a maximum number of on-line spares (defined by the reorder threshold + the number requested in a reorder). In the string parallel mapping, spares will be clustered on the last strings. In the string orthogonal mapping, spares will be spread over all strings. There is a third mapping, *string orthogonal with separate spares*, which clusters spares on the last strings while spreading data disks across the remaining strings. This mapping encourages string failures to be recovered by a separate string of spares so that the distribution of parity groups across strings remains widely spread. Figures 5.26 and 5.27 show examples of these mappings.

A.3. Modeling Repair, Recovery, and Spare Pool Replenishing

The idea behind on-line spares is that physical disk *repair* (most likely done by immediate replacement) may take a long time, but the *recovery* of its data onto an available on-line spare should be much faster. Once a failed disk's contents have been recovered on a spare, that spare now replaces the repairing data disk and the associated parity group is no longer vulnerable to data loss if another disk were to fail.

However, the data replacing spare is frequently not located on the same string as the repairing disk it replaces. In particular, it may be located on a string with other data disks (or other spares replacing data disks) from the same parity group. If this mapping persists then this parity group is vulnerable to immediate data loss if the wrong string fails. For this reason, the spares contents should end up back on the string holding the disk it is replacing. Figure 5.26 shows an example of this effect.

RELI explicitly models an operator visit (by default, at random in the 24 hours after the initial data disk failure). Whenever an operator visits the array, all spares replacing data disks are moved from their current string locations to the locations of the data disks they replace. At this point the replacing spare has become the replaced data disk and there is an empty slot on a string for a spare.

When a string fails (because of power, cabling, controller, or cooling component failures), it is possible that some or all of its disks are also damaged. Certainly their data is not available through that string's datapath. Because string failure may involve substantial replacement or a repairperson visit, RELI attempts to find a spare for each disk on the failed string. Unfortunately, during a string repair, a replacing spare cannot be physically moved onto the repairing string so string failed disks are exposed to abnormal mappings for longer periods than simple failed disks.

Because personnel is available at the array whenever a spare order is delivered or a string repair completes, all replacing spares are moved to the replaced disk's proper position at these times. A string repairperson is assumed to bring enough spare disks to replace all string failed data disks. RELI further assumes that both string repair and order delivery personnel bring enough extra spares to fill all failed or empty disk slots. However, if an order is delivered while a string is waiting repair, RELI assumes that the delivery personnel is not equipped to complete the string repair.

If more than one parity group is recovering the contents of a failed data disk in that group onto a spare disk concurrently, then these recoveries proceed either serially or in parallel. By default RELI will serialize recoveries because many I/O systems have too little bandwidth to perform parallel transfers of the entire contents of many disks.

A.4. Verifying RELI

Each time I write a simulator for a complex system that has no measurable implementation and whose output is only statistics, I am struck by the problem of determining correctness. What is needed is some confidence that the simulator models a reasonable system, and that its outputs are correctly derived from that reasonable system.

Of course, at first simple test cases and hand verified traces suffice. But the most interesting and complex aspects of the system may only occur under duress, so another answer is needed for validating these operations. I view this as a question of correct programming and have applied multiple strategies. First, throughout the code I introduce assertions of invariants that are tested as the program runs. If an invariant assertion fails, the program aborts immediately. Whenever the simulation clock is advanced, a special set of invariants testing the state of all disks, parity groups, and strings is applied. A second strategy collects statistics whose values have known expected values; for example, the disk and string failure rates are computed and compared to the corresponding input parameters.

A more powerful approach, albeit substantially more expensive, is to implement the simulator twice can compare results. Because I implemented both simulators, the second uses a different, simpler structure: no global state or event lists are maintained. The second simulator examines an event trace written by RELI and verifies that the given event is legal at that time. Extensive invariant state checks are made in the second simulator; it runs between one and two orders of magnitude slower than RELI. RELI's event trace includes overall statistics at the end so that the second simulator can verify these to within 0.001%. This verification scheme was responsible for uncovering a large fraction of the bugs in RELI.

Only a small fraction of the data collected for this research came from RELI runs verified by the second simulator, but a substantial peppering of the input parameter space was verified without disagreement (after both progams stabilized).

A.5. Terminating RELI

The simulation of an individual array's lifetime is a sample of the underlying distribution of array lifetimes modeled by the simulator. Sample values will vary; in fact, because data loss depends on the occurrence of two disks in one group in recovery at the same time when recovery and even replacement is very much faster than failure, opportunities for data loss are rare. Processes based on rare events tend to have large variation in their samples. Disk array time until data loss has such large variation.

To get a good estimate of the underlying model's mean time to data loss (MTTDL), RELI samples the distribution (by simulating an array from birth to data loss) until a *stopping condition* is met. The purpose of the stopping condition is to make sure that the final statistics reported are good estimators of the underlying model's average statistics; that is, how many arrays must be simulated to data loss before the average time until data loss and other important metrics are trustworthy.

The approach used in RELI is to stop as soon as a particular confidence interval on MTTDL is no larger than a given fraction of the current MTTDL estimate. By default, RELI stops when the 90% confidence interval is no larger than [0.9*MTTDL, 1.1*MTTDL], however, all data reported in Chapter 5 continued simulation until the 95% confidence interval was no larger than [0.95*MTTDL, 1.05*MTTDL]. RELI's stopping condition does not test any other measured statistic largely because MTTDL is the most important and one of the most likely to be variable.

APPENDIX B

Effect on Reliability of Error in MTTDL

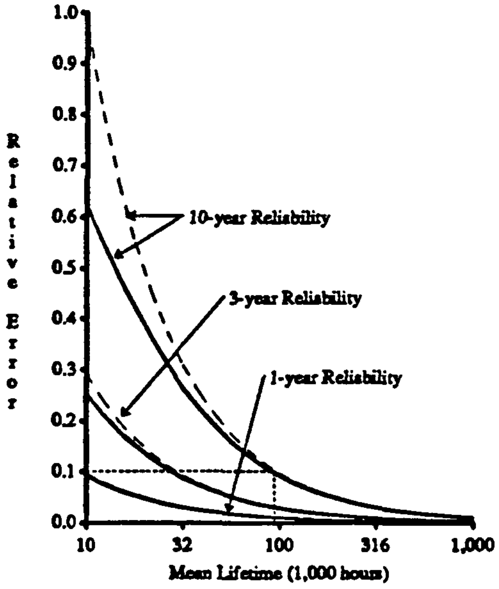


Figure B.1: Effect of 10% Error in Estimated MTTDL on Reliability. *The relative error in a system's 1-, 3-, and 10-year reliability resulting from a 10% underestimation of mean lifetime is shown by solid lines as a function of the true mean lifetime. For example, if the true mean lifetime of a system is 100,000, then underestimating it by 10% (to 90,000), will underestimate the system's 10-year reliability by less than 10%. This kind of error should be expected from simulated estimates for mean lifetime because their 95% confidence interval has a width that is 10% of its estimate. Overestimates for the mean lifetime of 10% overestimate the system's reliability by a similar amount. The dotted lines approximate of this error with the function* $t\beta/(1\pm\beta)M$ *where* 100β *is the percent error in the estimated mean and M is the true mean. This approximation for the error is no more than 10% optimistic for the relative error in the 10 year reliability when the true mean is larger than 50270 hours and is underestimated by 10%.*

APPENDIX C

Goodness-of-Fit Testing

This appendix strengthens Chapter 5's conclusion that the lifetimes of redundant disk arrays have an exponential distribution. In Chapter 5, Figures 5.7, 5.19, 5.23, and 5.28 show *that the 1-, 3-, and 10-year reliabilities of an exponential random variable concur on a graph* with these reliabilities estimated by simulating disk array lifetimes. Taken with the thoeretical expectation, given in Section 5.9, that disk array lifetimes should have an exponential distribution, these figures make an intuitive case for exponentially distributed lifetimes for disk arrays. This section presents statistical evidence for this result by applying ***Pearson's chi-square*** (χ^2) *goodness-of-fit* test [Kirk90 pp 533, Lawless82 pp 441] to the data generated by simulation.

There are many sets of data to which I have applied this goodness-of-fit test. Each of Section 5.4, 5.5, 5.6, and 5.7 has tested a model against about 100 different parameter sets, selected at random.[1] For each of these parameter sets, RELI has simulated about 1,000 to 1,500 disk arrays from installation to data loss. It is these collections of individual lifetimes (one collection per parameter set per section) that can be tested with the goodness-of-fit test because each lifetime in one collection is a separate sample for the same disk array.

[1] A parameter set is an assignment of a value to each variable in a particular disk array model. For example, in Figure 5.9 of Section 5.4, a particular parameter set for the strawman disk array is: G = 7 parity groups, N = 10 data disks in a parity group, $MTTF_{disk}$ = 150,000 hours for the mean lifetime of a disk, and $MTTR_{disk}$ = 72 hours for the mean repair time of a disk.

Hypothesis – Simulated Lifetimes are Individually Exponential								
χ^2	§ 5.4		§ 5.5		§ 5.6		§ 5.7	
p-value	#	%	#	%	#	%	#	%
0.01	0	0.00	1	0.01	0	0.00	11	0.01
0.05	**3**	**0.03**	**3**	**0.03**	**6**	**0.06**	**45**	**0.05**
0.10	11	0.12	7	0.07	12	0.12	93	0.11
0.25	22	0.23	21	0.21	31	0.32	229	0.27
0.50	54	0.57	46	0.46	57	0.58	464	0.54
0.75	75	0.80	77	0.78	76	0.78	668	0.78
1.00	94	1.00	99	1.00	98	1.00	857	1.00

Table C.1: Goodness-of-Fit Tests on Each Set of Simulated Lifetimes. *Pearson's chi-square goodness-of-fit test was applied to the 1,000 to 1,500 simulated lifetimes of each parameter set for all four sections. This table shows a cummulative distribution of these p-values. Each row shows the number of and fraction of tests whose p-values are less than 0.01, 0.05, 0.10, 0.25, 0.50, 0.75, and 1.00 for each section. The bold row show the number of and fraction of tests that give evidence against an exponential distribution for lifetimes with a 95% confidence level.*

The procedure for computing Pearson's statistic on a sample from an exponential distribution yields a random variable that has a χ^2 distribution. This means that if the lifetimes simulated for one parameter set have an exponential distribution, then computing the Pearson's statistic on a sample of these lifetimes is the same as sampling a single value from a χ^2 distribution. Because tables of and programs for the χ^2 distribution are available,[2] I then compute the probability that a single sample from this distribution would be at least as large as the simulation's sample statistic. If this probability, which is often called the test's *p-value*, is quite small, then it is unlikely that the simulation's sample statistic is a random sample from a χ^2 distribution, and, therefore, it is unlikely that the lifetimes of disk arrays with this particular parameter set have an exponential distribution.

[2] When this test involves alot of data, it is likely that the degrees of freedom for the χ^2 distribution exceed 20 or 30. Because most tables for this distribution do not contain entries for such high degrees of freedom, I have used the Wilson-Hilferty transformation [Lawless82 pp 513] on the χ^2 statistic. This transforms a χ^2 random variable to a standard normal random variable (having a normal distribution with a mean of zero and standard deviation of one).

I have computed the p-values for the simulation data of each parameter set in each section. Table C.1 shows the number (#) and fraction (%) of parameter sets whose p-values are less than 0.01, 0.05, 0.10, 0.25, 0.50, 0.75, and 1.00 for each section. Bold entries in this table show the number and fraction of parameter sets whose p-values are less than 0.05. For these 57 cases, the hypothesis that disk-array lifetimes have an exponential distribution would be rejected with a 95% confidence level. Before concluding that these parameter sets give rise to disk arrays whose lifetimes do not have an exponential distribution, let me review what it means to reject a hypothesis with a 95% confidence level.

For a single goodness-of-fit test, Pearson's statistic should take a value with the same distribution as a χ^2 random variable. The test rejects its hypothesis with a 95% confidence level if the probability that a single value from a χ^2 distribution exceeds the computed statistic is less than 0.05. This means that the hypothesis is falsely rejected with probability 0.05. So if the test is repeated many times on sets of data that all meet the hypothesis, then about 5% of these tests can be expected to yield false rejections. Because Table C.1 shows that the 57 cases that would be rejected with a 95% confidence level represent around 5% of their respective sections, it is plausible that these rejections are simply the expected false rejections.

I formalize this argument by stating a broader hypothesis: that for each section, the lifetimes of disk arrays constructed with all parameter sets have exponential distributions. If this

Hypothesis – Simulated Lifetimes are Collectively Exponential				
χ^2	§ 5.4	§ 5.5	§ 5.6	§ 5.7
p-value	0.39	0.82	0.27	0.53

Table C.2: Goodness-of-Fit Tests on All Simulated Lifetimes in Each Section. *Using the Pearson statistics computed for Table C.1, I test that in each section these statistics have a chi-square distribution. This table shows the p-value for each of these four tests. None of these tests yield evidence against the hypothesis that all lifetimes simulated in each section have exponential distributions.*

hypothesis is true then each Pearson statistic, computed for the simulated lifetimes of a particular parameter set, has a χ^2 distribution, as explained above. I test that the Pearson statistics have this distribution by applying (recursively, if you will) Pearson's chi-square goodness-of-fit test. This results in a single p-value for each section; shown in Table C.2, none of these four p-values are small enough to be rejected with a 95% confidence level, so the hypothesis that lifetimes for all parameter sets have exponential distributions is not rejected.

Based on these goodness-of-fit tests, I cannot find evidence against the hypothesis that the lifetimes of disk arrays modeled in Sections 5.4, 5.5, 5.6, and 5.7 have exponential distributions.

APPENDIX D

Comparison Data for Chapter 5

In this section I present tabular versions of data shown graphically in some of the figures of Chapter 5. For convenience, the major parameters are redefined here.

G is the number of parity groups in a disk array.

N is the number of disks in a parity group, excluding the parity disk.

Disk MTTF

is the mean time to failure of an individual disk, expressed in 1,000 hours.

Disk MTTR

is the mean time to repair an individual disk, expressed in hours.

String MTTF

is the mean time to failure of any component other than a disk in a string of support hardware, expressed in 1,000 hours.

String MTTR

is the mean time to repair of an individual string, expressed in hours.

Order Thres.

is the size of the on-line spare pool at which an order for more spares is issued.

Order Size

is the minimum number of new disks that are provided by an order for more spare disks.

More precisely, this is the difference between the order threshold and the maximum number of on-line spare disks that the array can hold.

Order Time

is the period of time before an issued order is delivered, expressed in hours.

MTTDL Markv

is the mean time to data loss that is estimated by Sharpe's solution of a Markov model, expressed in 1,000 hours.

MTTDL Est., Ortho, Ortho′

is the mean time to data loss that is estimated by the models presented in Chapter 5, expressed in 1,000 hours.

MTTDL Simul

is the mean time to data loss that is estimated by my RELI simulation software, expressed in 1,000 hours.

Table D.1 – Data for Section 5.4, Figures 5.6 and 5.7, part 1

Parity	Group	Disk		Mean Array Lifetimes	
Groups (G)	Size (N+1)	MTTF	MTTR	Estimate	Simulation
1	4	20	72.0	463.0	474.4
1	5	20	24.0	833.4	836.4
1	21	50	8.0	744.0	728.1
1	51	100	8.0	490.2	510.9
1	51	100	24.0	163.4	164.2
2	9	20	24.0	115.8	119.7
2	9	20	72.0	38.6	40.2
2	17	100	72.0	255.3	261.3
2	21	50	8.0	372.0	390.4
2	51	20	1.0	78.4	78.3
2	51	200	8.0	980.4	1012.0
2	51	200	24.0	326.8	325.7
2	51	500	72.0	680.9	670.9
3	5	20	8.0	833.3	833.6
3	9	20	8.0	231.5	236.2
3	13	20	8.0	106.8	110.9
3	13	100	72.0	296.8	305.4
3	17	100	72.0	170.2	179.3
3	21	20	0.5	634.9	625.5
4	9	50	72.0	120.6	128.8
4	9	100	72.0	482.3	492.4
4	13	20	2.0	320.5	328.0
4	13	50	24.0	166.9	169.9
4	13	50	72.0	55.6	57.2
4	13	100	72.0	222.6	215.3
4	17	50	24.0	95.7	97.9
4	21	50	8.0	186.0	182.9
4	21	50	72.0	20.7	21.3
4	21	200	24.0	992.1	998.3
4	51	100	24.0	40.9	42.2
4	51	200	8.0	490.2	485.7
4	51	200	72.0	54.5	59.0
5	5	20	72.0	55.6	55.5
5	9	50	72.0	96.5	96.8
5	9	100	72.0	385.8	386.9
5	13	20	8.0	64.1	63.2
5	17	20	2.0	147.1	146.2
5	17	20	24.0	12.3	12.4
5	21	50	2.0	595.2	604.0
5	21	50	72.0	16.5	17.1
5	51	20	2.0	15.7	15.8
10	2	20	24.0	833.4	810.6
10	5	20	72.0	27.8	27.8
10	5	100	72.0	694.5	684.2
10	9	20	8.0	69.4	67.3
10	9	20	24.0	23.1	24.0
10	9	50	8.0	434.0	437.9
10	13	20	1.0	256.4	258.4
10	17	50	8.0	114.9	117.5
10	51	20	1.0	15.7	16.5
10	51	50	1.0	98.0	98.7
10	51	50	8.0	12.3	13.3
10	51	100	24.0	16.3	16.6
10	51	200	24.0	65.4	64.7
15	5	20	2.0	666.7	654.1

Table D.1 – Data for Section 5.4, Figures 5.6 and 5.7, part 2

Parity	Group	Disk		Mean Array Lifetimes	
Groups (G)	Size (N+1)	MTTF	MTTR	Estimate	Simulation
15	13	50	8.0	133.5	129.0
20	2	50	72.0	868.1	888.7
20	4	20	8.0	208.3	203.0
20	4	50	24.0	434.1	437.1
20	4	100	72.0	578.8	584.6
20	9	100	8.0	868.1	854.1
20	9	200	72.0	385.8	386.4
20	17	50	0.5	919.1	905.4
20	17	100	24.0	76.6	76.2
20	17	100	72.0	25.5	26.2
20	21	20	1.0	47.6	49.2
20	21	20	2.0	23.8	23.7
20	21	50	0.5	595.2	592.5
20	21	50	1.0	297.6	319.4
20	21	200	8.0	595.2	597.8
20	51	500	24.0	204.3	203.0
50	2	20	8.0	500.0	491.6
50	2	20	24.0	166.7	163.9
50	4	50	24.0	173.6	174.5
50	4	100	72.0	231.5	219.4
50	5	20	0.5	800.0	792.7
50	9	50	8.0	86.8	87.7
50	13	100	24.0	53.4	53.6
50	13	200	8.0	641.0	663.2
50	17	20	0.5	58.8	56.1
50	17	20	1.0	29.4	30.1
50	17	50	2.0	91.9	91.9
50	21	50	0.5	238.1	242.0
50	21	100	8.0	59.5	58.3
100	4	20	0.5	666.7	649.2
100	4	50	8.0	260.4	263.7
100	4	50	72.0	28.9	27.6
100	9	20	0.5	111.1	110.7
100	9	100	8.0	173.6	174.5
100	9	200	72.0	77.2	77.8
100	17	50	2.0	46.0	46.7
100	17	200	2.0	735.3	702.5
100	21	50	0.5	119.0	111.0
100	21	50	1.0	59.5	60.4

Table D.1: Raw Data for Figures 5.6 and 5.6 in Section 5.4. *These 94 data sets each represent a comparison between the estimate of mean time to data loss (MTTDL) derived in Section 5.4 and the corresponding estimate generated by simulation. In this model, there are no dependent disk failures and no on-line spare disks. These 94 parameter sets were selected by first generating the cross-product of sets of conceivable values for each parameter, then selecting approximately 100 sets at random. The units for "Disk MTTR" is hours, and for "Disk MTTF," "MTTDL Estimate," and "MTTDL Simulation" it is 1,000 hours. The expression for "MTTDL Estimate" is found in Equations and in Section 5.4. Simulated MTTDL has a 95% confidence interval that extends ±5% of its value.*

Table D.2 – Data for Section 5.5, Figures 5.18 and 5.19, part 1

G	N+1	Disk		String		MTTDL			
		MTTF	MTTR	MTTF	MTTR	MARKV	ORTHO	ORTHO′	SIMUL
1	9	100	2.0	100	168	420.6	420.4	403.6	429.8
1	13	100	72.0	50	8	97.3	94.9	92.1	98.9
1	17	20	24.0	500	24	56.9	56.7	54.6	56.3
1	17	100	72.0	100	24	115.2	112.9	109.4	117.3
1	17	500	0.5	50	72	120.9	121.0	115.2	115.8
1	21	20	8.0	100	72	35.9	35.5	33.1	34.8
1	21	500	72.0	50	8	66.3	64.3	62.1	64.9
2	4	20	24.0	200	168	369.8	367.9	357.5	374.4
2	4	20	24.0	200	8	564.4	563.8	559.0	586.6
2	5	20	72.0	500	168	123.2	122.6	118.1	120.8
2	9	20	2.0	1000	168	550.1	549.3	507.3	564.1
2	9	20	2.0	50	72	78.0	77.8	72.3	76.7
2	9	50	24.0	200	2	465.7	464.9	460.7	457.8
2	9	100	24.0	100	168	214.2	212.9	203.0	208.5
2	9	200	8.0	50	24	612.3	615.3	609.2	643.0
2	17	50	72.0	1000	8	60.7	60.4	57.6	61.4
2	21	50	2.0	500	8	918.3	917.6	914.3	924.1
2	21	50	8.0	1000	2	336.2	336.0	333.7	342.7
2	21	50	24.0	1000	168	90.0	89.3	84.9	87.4
2	21	100	24.0	1000	24	383.1	382.2	378.0	379.6
3	5	20	72.0	100	2	67.8	67.2	64.7	66.0
3	5	200	2.0	50	168	431.4	431.7	415.1	404.8
3	13	20	1.0	200	168	59.9	59.8	46.0	58.4
3	13	200	72.0	1000	2	840.9	838.8	830.0	820.3
3	13	500	72.0	100	24	246.7	246.9	241.5	253.4
3	17	500	8.0	50	8	305.4	308.1	305.3	307.4
3	21	20	8.0	50	8	17.6	17.4	16.9	17.8
3	21	50	1.0	100	72	50.1	50.0	44.9	47.8
4	9	200	1.0	200	168	676.8	676.8	650.2	673.0
4	13	20	2.0	50	24	47.5	47.3	44.7	47.2
4	13	20	24.0	50	2	14.8	14.5	13.9	14.4
4	13	50	24.0	1000	24	147.4	147.0	144.9	145.1
4	13	100	1.0	50	24	207.3	207.3	202.7	207.8
4	13	100	8.0	100	72	142.6	141.9	135.6	149.1
4	17	20	0.5	100	24	68.9	68.8	63.9	70.0
4	17	20	1.0	50	24	32.1	32.0	29.5	33.0
4	17	50	0.5	200	24	332.2	332.0	321.8	321.8
4	17	200	72.0	500	24	258.7	256.4	251.6	255.9
4	17	500	72.0	100	72	96.2	94.7	90.5	93.8
4	21	50	8.0	1000	168	97.8	97.1	85.9	97.0
4	21	50	8.0	200	168	32.9	32.4	26.2	32.0
4	21	50	8.0	100	2	83.3	82.8	81.9	83.2
5	5	20	8.0	200	24	335.1	334.3	331.0	332.4
5	5	20	24.0	50	168	39.7	38.8	34.0	38.5
5	5	100	2.0	50	168	219.5	219.5	206.5	221.8
5	9	20	1.0	200	72	147.0	146.7	130.0	144.3
5	13	100	8.0	200	2	714.4	714.4	711.7	692.4
5	17	200	2.0	500	72	870.8	870.1	844.0	874.9
5	17	500	8.0	50	168	37.6	37.5	33.0	39.0
5	21	50	1.0	500	2	847.0	846.6	845.2	837.7
5	21	500	0.5	200	168	200.3	200.3	187.6	188.5

Table D.2 – Data for Section 5.5, Figures 5.18 and 5.19, part 2

G	N+1	Disk		String		MTTDL			
		MTTF	MTTR	MTTF	MTTR	MARKV	ORTHO	ORTHO′	SIMUL
10	5	50	72.0	500	168	127.8	126.0	120.7	125.3
10	5	100	2.0	200	168	580.8	580.3	541.2	581.0
10	9	50	24.0	50	24	35.2	34.4	32.9	35.6
10	9	100	8.0	500	24	868.5	866.2	857.9	837.0
10	17	20	8.0	100	8	12.0	11.8	11.4	11.8
10	17	50	0.5	500	168	79.8	79.6	52.3	77.1
10	17	50	1.0	1000	24	415.7	414.9	398.5	401.9
10	17	100	24.0	1000	168	90.3	88.9	80.2	94.1
10	21	100	8.0	1000	8	230.4	229.8	228.2	224.5
10	21	200	72.0	1000	168	77.5	75.4	70.1	75.9
10	21	500	1.0	50	168	21.0	21.0	17.4	21.3
15	2	20	1.0	200	168	824.6	824.0	735.8	781.2
15	2	20	72.0	200	8	155.8	154.9	152.6	156.2
15	2	20	72.0	100	72	120.0	118.3	114.9	120.7
15	4	20	0.5	200	24	770.1	769.3	736.6	789.0
15	4	50	2.0	100	24	857.7	857.4	841.3	876.6
15	4	500	72.0	100	168	603.1	616.6	601.1	602.9
15	5	50	1.0	50	72	114.3	114.2	104.1	118.6
15	9	200	2.0	200	168	222.2	221.8	199.4	221.6
15	9	200	8.0	100	168	97.6	97.1	86.5	94.1
15	13	20	2.0	200	168	19.4	19.1	8.8	19.0
15	17	20	1.0	100	8	34.5	34.3	32.4	34.7
15	17	50	8.0	200	2	49.6	49.2	48.6	51.8
15	17	100	24.0	50	2	18.3	17.6	16.8	18.4
15	17	200	24.0	500	2	222.7	221.0	218.5	226.3
15	21	20	1.0	1000	2	59.1	59.0	58.8	58.5
20	5	200	24.0	50	8	305.4	313.0	309.2	311.4
20	9	20	8.0	100	72	14.6	14.1	10.8	13.9
20	13	20	0.5	500	72	62.1	61.8	37.4	65.3
20	13	20	1.0	200	24	42.5	42.1	35.5	41.7
20	17	50	24.0	500	2	16.5	16.3	15.8	16.4
20	17	100	1.0	500	72	140.3	139.9	115.2	145.1
20	17	200	72.0	50	2	10.4	9.6	8.5	10.4
20	17	500	72.0	500	2	207.5	203.8	198.9	213.5
20	21	50	8.0	1000	72	27.8	27.4	24.0	28.9
20	21	500	24.0	50	2	32.6	32.1	30.8	33.7

Table D.2: Raw Data for Figures 5.18 and 5.19 in Section 5.5. *These 87 data sets each represent a comparison between the estimate of MTTDL derived in Section 5.5 and the corresponding estimate generated by simulation. In this model, there are dependent disk failures but no on-line spare disks. The caption to Table D.1 describes the parameter set selection method. Because the Sharpe software has limits on the number of states in the models it can solve, the 12 parameter sets with 50 parity groups could not be solved. The units for "Disk MTTR" and "String MTTR" is hours, and for "Disk MTTF," "String MTTF," and all four "MTTDL" columns, the units is 1,000 hours. The Markov model for "MTTDL Markv" is shown in Figure 5.12). and the expressions for "MTTDL Ortho" and "MTTDL* Ortho′" *are found in Equations 5.16 and 5.18, respectively. Simulated MTTDL has a 95% confidence interval that extends ±5% of its value.*

Table D.3 – Data for Section 5.6, Figure 5.23, part 1

Parity	Group	Disk		Order			MTTDL	
Groups (G)	Size (N+1)	MTTF	MTTR	Thres.	Size	Time	Estimated	Simulation
1	9	20	2.0	0	0	4	928.1	965.3
1	9	20	72.0	3	1	72	81.9	82.0
1	13	50	72.0	4	6	24	230.6	233.1
1	17	100	24.0	0	0	24	771.3	774.2
1	21	20	2.0	1	1	12	478.1	484.0
1	21	100	72.0	1	7	36	340.5	333.4
2	5	20	24.0	0	0	12	280.3	287.5
2	13	20	24.0	3	1	36	55.0	56.0
2	13	20	24.0	0	5	4	55.0	54.2
2	13	50	72.0	2	6	4	115.3	116.0
2	21	20	0.5	0	1	24	441.5	446.0
2	21	20	2.0	2	6	48	239.0	238.8
2	21	50	72.0	1	9	168	43.7	45.7
2	21	100	24.0	4	1	168	501.0	516.3
2	21	200	72.0	0	1	36	669.9	671.2
3	2	20	72.0	3	1	72	936.0	930.2
3	5	50	1.0	0	0	48	859.9	854.7
3	5	50	72.0	0	1	4	586.2	565.5
3	13	20	24.0	1	1	24	36.7	37.2
3	13	100	24.0	1	9	24	895.7	891.8
3	21	20	0.5	0	8	12	599.6	578.1
3	21	50	2.0	0	0	72	28.8	28.7
3	21	100	24.0	1	3	72	333.8	340.7
4	5	20	1.0	0	0	168	33.1	32.8
4	5	20	24.0	2	3	168	210.5	213.7
4	9	20	1.0	0	0	24	57.2	53.5
4	9	20	2.0	8	2	4	695.6	713.8
4	9	20	2.0	0	8	168	264.9	259.8
4	13	20	2.0	4	1	48	321.3	313.9
4	13	50	24.0	4	4	24	168.9	164.6
4	13	50	72.0	0	1	48	56.7	55.6
4	13	100	24.0	4	1	12	671.8	674.1
4	17	50	24.0	1	19	36	97.3	94.7
4	21	20	2.0	0	1	12	104.4	104.1
4	21	50	2.0	0	0	24	58.8	58.5
4	21	50	72.0	0	0	12	18.6	19.1
4	21	50	72.0	7	13	24	21.9	22.2
4	21	100	24.0	7	1	12	250.5	256.6
4	21	200	24.0	2	2	72	997.0	1007.8
4	21	200	72.0	0	0	4	317.7	312.4
5	5	20	24.0	8	12	72	168.5	175.6
5	5	20	72.0	4	16	72	57.4	58.9
5	5	20	72.0	1	1	168	56.5	56.7
5	5	50	72.0	0	1	48	348.9	356.9
5	13	20	0.5	0	0	4	114.9	116.8
5	13	100	72.0	3	1	72	181.3	190.1
5	17	20	24.0	0	1	24	12.2	12.1
5	17	100	72.0	1	7	4	104.6	99.5
5	21	50	2.0	0	0	24	47.3	46.1
5	21	50	24.0	1	3	12	50.6	50.2
10	2	20	72.0	1	1	48	280.7	281.9
10	9	20	0.5	0	0	4	124.6	124.8
10	9	20	24.0	1	1	12	23.6	23.6
10	13	50	2.0	1	1	36	783.4	759.7
10	13	50	72.0	0	0	4	21.8	22.7

Table D.3 – Data for Section 5.6, Figure 5.23, part 2

Parity	Group	Disk		Order			MTTDL	
Groups (G)	Size (N+1)	MTTF	MTTR	Thres.	Size	Time	Estimated	Simulation
10	13	500	0.5	0	0	168	975.5	944.5
10	17	100	24.0	6	14	24	154.4	153.3
10	21	20	0.5	4	1	48	190.2	178.4
10	21	20	2.0	1	1	36	37.3	36.6
10	21	50	24.0	4	1	36	25.3	24.5
10	21	50	24.0	1	4	4	25.3	25.2
10	21	100	2.0	0	8	72	890.7	889.0
20	5	20	2.0	4	1	48	500.4	528.3
20	5	20	2.0	4	4	36	500.4	508.8
20	5	50	24.0	0	1	4	261.4	265.5
20	9	50	24.0	4	1	4	72.9	69.2
20	9	100	72.0	0	10	48	97.3	96.3
20	13	20	2.0	2	3	48	61.0	60.6
20	13	50	0.5	0	0	36	24.0	23.9
20	13	50	2.0	1	1	168	72.1	69.4
20	17	50	1.0	3	1	36	459.5	464.8
20	17	200	2.0	0	0	48	153.2	150.5
20	17	500	72.0	0	1	168	572.6	572.6
20	21	50	2.0	0	1	24	74.4	74.5
20	21	200	72.0	0	8	168	63.9	64.7
20	21	200	72.0	1	4	24	67.1	66.4
50	2	50	24.0	1	1	168	954.6	944.0
50	2	50	72.0	3	1	168	348.7	350.3
50	5	20	0.5	8	2	36	800.2	757.5
50	5	20	2.0	7	1	72	200.2	196.9
50	5	50	72.0	1	1	168	31.3	29.7
50	9	20	0.5	0	2	4	189.6	183.7
50	9	20	2.0	1	19	36	51.4	50.4
50	9	100	2.0	1	4	168	577.3	607.2
50	13	20	0.5	0	20	48	23.2	23.8
50	13	100	24.0	2	18	4	53.7	55.4
50	13	100	72.0	7	1	4	18.1	18.3
50	13	200	2.0	0	0	24	204.3	205.5
50	17	20	0.5	2	6	12	57.9	59.4
50	17	100	1.0	2	6	12	735.5	716.2
50	17	100	72.0	10	10	168	10.5	10.9
50	17	500	24.0	2	2	48	767.2	790.3
50	21	20	0.5	0	1	4	22.5	22.5
50	21	50	2.0	0	8	24	44.0	43.6
50	21	100	0.5	1	1	48	281.4	273.8
50	21	200	1.0	0	1	36	494.0	484.9
50	21	200	2.0	0	1	168	46.7	47.9
50	21	200	2.0	1	1	24	927.4	944.5
50	21	200	72.0	0	8	4	26.8	26.0

Table D.3: Raw Data for Figure 5.23 in Section 5.6. *These 99 data sets each represent a comparison between the estimate of mean time to data loss (MTTDL) derived in Section 5.6 and the corresponding estimate generated by simulation. In this model, there are no dependent disk failures but there are on-line spare disks. These 99 parameter sets were selected by the method described in the caption to Table D.1. The units for "Disk MTTR" and "Order Time" is hours, and for "Disk MTTF," and both "MTTDL" columns, the units is 1,000 hours. Expressions for "MTTDL Estimated" are found in Equations 5.22, 5.27, and 5.28. Simulated MTTDL has a 95% confidence interval that extends ±5% of its value.*

Table D.4 – Data for Section 5.7, Figures 5.34 and 5.36, part 1

		Disk		String		Order			MTTDL	
G	N+1	MTTF	MTTR	MTTF	MTTR	THRES	SIZE	TIME	EST	SIMUL
1	5	50	24.0	1000	24	0	0	72	1233.3	1206.3
1	5	50	24.0	1000	24	0	1	72	4688.2	4810.9
1	5	50	24.0	1000	24	1	1	72	4744.6	4555.8
1	9	20	8.0	500	72	0	0	72	69.3	69.5
1	9	20	8.0	500	72	0	1	72	559.7	557.9
1	9	20	8.0	500	72	1	1	72	645.2	614.2
1	9	100	8.0	200	8	0	0	24	2329.4	2237.5
1	9	100	8.0	200	8	0	1	24	7708.9	7791.4
1	9	100	8.0	200	8	1	1	24	7728.1	7414.0
1	9	400	8.0	50	24	0	0	8	915.7	920.1
1	9	400	8.0	50	24	0	1	8	3415.0	3487.4
1	9	400	8.0	50	24	1	1	8	3438.7	3561.2
1	9	800	24.0	100	168	0	0	24	641.8	649.9
1	9	800	24.0	100	168	0	1	24	4359.3	4392.6
1	9	800	24.0	100	168	1	1	24	4590.1	4475.6
1	17	50	2.0	100	2	0	0	8	514.7	497.4
1	17	50	2.0	100	2	0	1	8	2038.3	2097.7
1	17	50	2.0	100	2	1	1	8	2045.6	2071.4
2	17	100	2.0	500	2	0	0	8	1431.3	1399.2
2	17	100	2.0	500	2	1	1	8	6430.5	6509.1
2	17	100	2.0	500	2	3	1	8	6432.2	6385.9
2	17	100	72.0	500	72	0	0	72	94.5	94.4
2	17	100	72.0	500	72	1	1	72	182.9	183.5
2	17	100	72.0	500	72	3	1	72	183.3	190.0
2	21	20	1.0	1000	2	0	0	72	7.8	7.4
2	21	20	1.0	1000	2	1	1	72	373.1	383.5
2	21	20	1.0	1000	2	3	1	72	458.6	469.6
2	21	50	1.0	100	24	0	0	72	26.0	26.0
2	21	50	1.0	100	24	1	1	72	1060.6	1052.6
2	21	50	1.0	100	24	3	1	72	1361.1	1345.9
2	21	50	2.0	1000	24	0	0	8	254.5	247.1
2	21	50	2.0	1000	24	1	1	8	1349.8	1383.3
2	21	50	2.0	1000	24	3	1	8	1352.8	1403.8
2	21	100	1.0	50	24	0	0	72	39.4	38.9
2	21	100	1.0	50	24	1	1	72	1109.0	1108.8
2	21	100	1.0	50	24	3	1	72	1487.4	1432.9
2	21	100	24.0	500	24	0	0	72	99.3	96.6
2	21	100	24.0	500	24	1	1	72	349.7	347.5
2	21	100	24.0	500	24	3	1	72	350.7	358.6
2	21	200	1.0	100	8	0	0	72	192.0	182.5
2	21	200	1.0	100	8	1	1	72	5661.8	5748.2
2	21	200	1.0	100	8	3	1	72	5956.0	6154.3

Table D.4 – Data for Section 5.7, Figures 5.34 and 5.36, part 2

		Disk		String		Order			MTTDL	
G	N+1	MTTF	MTTR	MTTF	MTTR	THRES	SIZE	TIME	EST	SIMUL
3	4	50	72.0	500	168	0	0	24	541.7	551.0
3	4	50	72.0	500	168	2	1	24	807.9	849.7
3	4	50	72.0	500	168	5	1	24	808.5	833.5
3	4	200	72.0	100	72	0	0	8	1402.7	1335.2
3	4	200	72.0	100	72	2	1	8	2090.4	2078.1
3	4	200	72.0	100	72	5	1	8	2092.5	2054.6
3	5	100	24.0	1000	72	0	0	8	3705.8	3674.3
3	5	100	24.0	1000	72	2	1	8	5770.6	6028.8
3	5	100	24.0	1000	72	5	1	8	5771.7	5697.3
3	13	200	8.0	100	168	0	0	24	131.4	130.1
3	13	200	8.0	100	168	2	1	24	1269.8	1284.0
3	13	200	8.0	100	168	5	1	24	1438.8	1378.2
3	13	200	72.0	500	72	0	0	72	324.4	327.5
3	13	200	72.0	500	72	2	1	72	632.6	637.0
3	13	200	72.0	500	72	5	1	72	633.7	647.7
3	21	100	0.5	50	168	0	0	8	16.6	16.0
3	21	100	0.5	50	168	2	1	8	345.5	339.3
3	21	100	0.5	50	168	5	1	8	1868.9	1784.6
3	21	100	8.0	100	168	0	0	8	33.9	34.0
3	21	100	8.0	100	168	2	1	8	240.7	244.2
3	21	100	8.0	100	168	5	1	8	276.2	269.8
4	2	20	24.0	100	24	0	0	24	743.0	747.5
4	2	20	24.0	100	24	3	1	24	1471.5	1466.0
4	2	20	24.0	100	24	7	1	24	1472.6	1476.5
4	5	20	0.5	200	72	0	0	8	307.0	310.5
4	5	20	0.5	200	72	3	1	8	7944.2	7948.7
4	5	20	0.5	200	72	7	1	8	8299.0	8332.1
4	5	20	2.0	500	2	0	0	72	69.1	65.7
4	5	20	2.0	500	2	3	1	72	2313.3	2428.6
4	5	20	2.0	500	2	7	1	72	2315.2	2284.7
4	5	20	8.0	200	72	0	0	24	118.7	120.9
4	5	20	8.0	200	72	3	1	24	517.2	505.4
4	5	20	8.0	200	72	7	1	24	520.7	530.4
4	13	200	2.0	200	2	0	0	72	426.6	449.3
4	13	200	2.0	200	2	3	1	72	9082.3	8793.2
4	13	200	2.0	200	2	7	1	72	9109.2	9434.6
4	17	20	1.0	200	24	0	0	24	13.2	13.0
4	17	20	1.0	200	24	3	1	24	292.0	284.1
4	17	20	1.0	200	24	7	1	24	305.6	303.4
4	17	50	2.0	50	24	0	0	72	14.0	14.0
4	17	50	2.0	50	24	3	1	72	276.9	281.6
4	17	50	2.0	50	24	7	1	72	327.2	323.2

Table D.4 – Data for Section 5.7, Figures 5.34 and 5.36, part 3

G	N+1	Disk		String		Order			MTTDL	
		MTTF	MTTR	MTTF	MTTR	THRES	SIZE	TIME	EST	SIMUL
4	17	100	72.0	1000	72	0	0	72	56.3	56.5
4	17	100	72.0	1000	72	3	1	72	108.7	107.7
4	17	100	72.0	1000	72	7	1	72	108.8	109.2
4	17	400	24.0	500	72	0	0	8	967.5	909.0
4	17	400	24.0	500	72	3	1	8	2094.6	2103.6
4	17	400	24.0	500	72	7	1	8	2098.3	2148.8
4	21	20	2.0	1000	24	0	0	24	9.7	9.8
4	21	20	2.0	1000	24	3	1	24	114.3	112.7
4	21	20	2.0	1000	24	7	1	24	114.9	120.5
4	21	50	2.0	500	8	0	0	72	20.2	19.4
4	21	50	2.0	500	8	3	1	72	611.9	619.2
4	21	50	2.0	500	8	7	1	72	618.5	631.5
4	21	800	8.0	200	24	0	0	72	558.4	581.7
4	21	800	8.0	200	24	3	1	72	2732.2	2804.9
4	21	800	8.0	200	24	7	1	72	2756.4	2775.1
4	21	800	24.0	1000	72	0	0	24	2101.2	2156.7
4	21	800	24.0	1000	72	3	1	24	5417.1	5353.1
4	21	800	24.0	1000	72	7	1	24	5426.6	5553.5
5	4	50	24.0	200	72	0	0	8	643.8	638.4
5	4	50	24.0	200	72	4	1	8	1140.1	1156.3
5	4	50	24.0	200	72	9	1	8	1141.2	1154.2
5	4	100	24.0	1000	168	0	0	72	1349.8	1310.0
5	4	100	24.0	1000	168	4	1	72	5761.8	5674.8
5	4	100	24.0	1000	168	9	1	72	5776.8	5522.8
5	4	400	24.0	50	168	0	0	8	586.2	583.2
5	4	400	24.0	50	168	4	1	8	2356.7	2334.8
5	4	400	24.0	50	168	9	1	8	2420.8	2431.3
5	5	200	8.0	50	72	0	0	8	591.3	611.8
5	5	200	8.0	50	72	4	1	8	3015.2	3003.2
5	5	200	8.0	50	72	9	1	8	3075.6	2956.7
5	5	800	72.0	100	168	0	0	8	946.9	946.9
5	5	800	72.0	100	168	4	1	8	1939.3	1956.4
5	5	800	72.0	100	168	9	1	8	1950.0	1983.1
5	9	400	8.0	100	72	0	0	24	587.7	590.0
5	9	400	8.0	100	72	4	1	24	3345.7	3245.1
5	9	400	8.0	100	72	9	1	24	3416.9	3405.4
5	13	50	2.0	500	168	0	0	8	127.0	133.1
5	13	50	2.0	500	168	4	1	8	1290.6	1300.9
5	13	50	2.0	500	168	9	1	8	1331.9	1365.6
5	13	200	8.0	50	72	0	0	24	70.0	69.8
5	13	200	8.0	50	72	4	1	24	374.9	395.6
5	13	200	8.0	50	72	9	1	24	396.4	388.1

Table D.4 – Data for Section 5.7, Figures 5.34 and 5.36, part 4

		Disk		String		Order			MTTDL	
G	N+1	MTTF	MTTR	MTTF	MTTR	THRES	SIZE	TIME	EST	SIMUL
5	17	50	0.5	1000	24	0	0	72	26.2	24.0
5	17	50	0.5	1000	24	4	1	72	3165.1	3238.1
5	17	50	0.5	1000	24	9	1	72	3339.9	3317.8
5	17	50	8.0	100	24	0	0	72	14.7	15.6
5	17	50	8.0	100	24	4	1	72	106.9	107.6
5	17	50	8.0	100	24	9	1	72	109.8	109.6
5	17	100	0.5	50	24	0	0	24	53.1	55.5
5	17	100	0.5	50	24	4	1	24	1702.6	1693.0
5	17	100	0.5	50	24	9	1	24	2153.5	2188.8
5	17	200	24.0	100	168	0	0	72	37.9	39.0
5	17	200	24.0	100	168	4	1	72	169.4	173.2
5	17	200	24.0	100	168	9	1	72	182.7	184.4
5	21	50	2.0	200	24	0	0	24	31.7	32.4
5	21	50	2.0	200	24	4	1	24	379.8	385.3
5	21	50	2.0	200	24	9	1	24	390.3	397.5
5	21	200	8.0	500	8	0	0	72	165.2	168.8
5	21	200	8.0	500	8	4	1	72	1271.8	1276.7
5	21	200	8.0	500	8	9	1	72	1274.8	1252.8
7	4	20	2.0	1000	24	0	0	72	67.1	64.5
7	4	20	2.0	1000	24	6	1	72	2281.5	2212.3
7	4	20	2.0	1000	24	13	1	72	2290.7	2274.9
7	5	20	0.5	200	2	0	0	8	300.5	304.8
7	5	20	0.5	200	2	6	1	8	4744.3	4631.8
7	5	20	0.5	200	2	13	1	8	4748.6	4661.6
7	5	50	2.0	200	8	0	0	72	194.7	191.6
7	5	50	2.0	200	8	6	1	72	5822.9	5962.0
7	5	50	2.0	200	8	13	1	72	5865.2	5985.1
7	9	100	8.0	1000	24	0	0	8	956.5	985.2
7	9	100	8.0	1000	24	6	1	8	2063.5	2066.1
7	9	100	8.0	1000	24	13	1	8	2064.0	2065.4
7	9	400	8.0	200	72	0	0	24	1040.3	1033.8
7	9	400	8.0	200	72	6	1	24	6062.5	6129.0
7	9	400	8.0	200	72	13	1	24	6133.5	6241.8
7	17	20	1.0	200	8	0	0	24	8.2	8.0
7	17	20	1.0	200	8	6	1	24	172.4	174.5
7	17	20	1.0	200	8	13	1	24	174.9	181.9
7	17	100	1.0	100	2	0	0	72	37.7	37.7
7	17	100	1.0	100	2	6	1	72	1537.1	1562.0
7	17	100	1.0	100	2	13	1	72	1560.0	1543.7
7	17	100	24.0	200	72	0	0	72	28.8	29.7
7	17	100	24.0	200	72	6	1	72	104.8	104.9
7	17	100	24.0	200	72	13	1	72	106.4	104.4

Table D.4 – Data for Section 5.7, Figures 5.34 and 5.36, part 5

G	N+1	Disk		String		Order			MTTDL	
		MTTF	MTTR	MTTF	MTTR	THRES	SIZE	TIME	EST	SIMUL
7	17	200	24.0	500	24	0	0	24	241.4	236.6
7	17	200	24.0	500	24	6	1	24	474.2	460.2
7	17	200	24.0	500	24	13	1	24	474.5	470.2
10	2	400	72.0	50	168	0	0	24	1555.1	1581.7
10	2	400	72.0	50	168	9	1	24	3124.0	3058.2
10	2	400	72.0	50	168	19	1	24	3143.6	3071.0
10	5	400	24.0	200	72	0	0	72	1503.8	1510.1
10	5	400	24.0	200	72	9	1	72	5393.2	5341.7
10	5	400	24.0	200	72	19	1	72	5419.6	5388.0
10	9	20	2.0	1000	24	0	0	72	8.7	8.6
10	9	20	2.0	1000	24	9	1	72	265.6	250.2
10	9	20	2.0	1000	24	19	1	72	267.6	271.2
10	13	50	0.5	100	24	0	0	24	33.9	32.8
10	13	50	0.5	100	24	9	1	24	1365.6	1341.8
10	13	50	0.5	100	24	19	1	24	1546.6	1529.7
10	13	200	2.0	500	24	0	0	8	1039.8	1065.8
10	13	200	2.0	500	24	9	1	8	6925.2	6769.2
10	13	200	2.0	500	24	19	1	8	6945.8	6523.9
10	17	50	0.5	50	24	0	0	8	22.2	21.8
10	17	50	0.5	50	24	9	1	8	486.8	501.1
10	17	50	0.5	50	24	19	1	8	558.6	548.7
10	17	200	2.0	1000	72	0	0	8	561.3	569.5
10	17	200	2.0	1000	72	9	1	8	5171.8	5015.8
10	17	200	2.0	1000	72	19	1	8	5211.5	5173.1
10	21	200	8.0	100	8	0	0	8	104.3	107.3
10	21	200	8.0	100	8	9	1	8	194.7	195.2
10	21	200	8.0	100	8	19	1	8	194.9	188.3
12	4	50	8.0	500	168	0	0	24	343.4	350.7
12	4	50	8.0	500	168	11	1	24	1796.6	1798.6
12	4	50	8.0	500	168	23	1	24	1807.4	1772.6
12	13	50	1.0	50	2	0	0	8	63.3	62.1
12	13	50	1.0	50	2	11	1	8	409.5	404.3
12	13	50	1.0	50	2	23	1	8	410.7	400.6
12	13	50	1.0	100	168	0	0	8	18.8	19.0
12	13	50	1.0	100	168	11	1	8	428.3	437.4
12	13	50	1.0	100	168	23	1	8	644.6	665.2
12	13	100	0.5	1000	168	0	0	72	60.7	60.1
12	13	100	0.5	1000	168	11	1	72	6743.3	6871.0
12	13	100	0.5	1000	168	23	1	72	8882.3	8533.6
12	13	400	2.0	200	168	0	0	24	191.8	188.7
12	13	400	2.0	200	168	11	1	24	5654.7	5673.0
12	13	400	2.0	200	168	23	1	24	7079.4	7295.2

Table D.4 – Data for Section 5.7, Figures 5.34 and 5.36, part 6										
		Disk		String		Order			MTTDL	
G	N+1	MTTF	MTTR	MTTF	MTTR	THRES	SIZE	TIME	EST	SIMUL
12	13	400	24.0	50	168	0	0	8	31.1	31.2
12	13	400	24.0	50	168	11	1	8	103.1	102.9
12	13	400	24.0	50	168	23	1	8	109.8	106.6
12	13	800	72.0	100	168	0	0	24	75.5	76.5
12	13	800	72.0	100	168	11	1	24	147.0	151.2
12	13	800	72.0	100	168	23	1	24	149.0	142.5
12	17	50	2.0	50	72	0	0	8	9.5	9.7
12	17	50	2.0	50	72	11	1	8	99.3	99.8
12	17	50	2.0	50	72	23	1	8	118.0	119.2
12	21	20	0.5	100	8	0	0	8	7.1	6.9
12	21	20	0.5	100	8	11	1	8	110.1	108.0
12	21	20	0.5	100	8	23	1	8	112.8	112.3
12	21	50	2.0	100	72	0	0	72	4.2	4.8
12	21	50	2.0	100	72	11	1	72	87.8	87.6
12	21	50	2.0	100	72	23	1	72	120.4	118.0
12	21	400	8.0	1000	168	0	0	24	287.9	292.5
12	21	400	8.0	1000	168	11	1	24	2124.4	2075.7
12	21	400	8.0	1000	168	23	1	24	2159.6	2227.4
15	4	20	8.0	500	168	0	0	8	103.1	104.2
15	4	20	8.0	500	168	14	1	8	257.4	251.9
15	4	20	8.0	500	168	29	1	8	258.0	248.1
15	5	50	1.0	500	72	0	0	8	502.3	519.1
15	5	50	1.0	500	72	14	1	8	6872.6	6724.2
15	5	50	1.0	500	72	29	1	8	6933.6	6965.8
15	9	100	2.0	500	8	0	0	24	283.8	277.6
15	9	100	2.0	500	8	14	1	24	3282.9	3343.9
15	9	100	2.0	500	8	29	1	24	3288.4	3302.8
15	13	200	2.0	200	2	0	0	24	306.5	309.7
15	13	200	2.0	200	2	14	1	24	2653.7	2640.0
15	13	200	2.0	200	2	29	1	24	2656.6	2651.0
15	21	50	0.5	1000	24	0	0	24	16.1	15.8
15	21	50	0.5	1000	24	14	1	24	705.4	700.5
15	21	50	0.5	1000	24	29	1	24	721.5	714.2
20	2	50	72.0	500	72	0	0	8	615.9	622.0
20	2	50	72.0	500	72	19	1	8	727.1	744.3
20	2	50	72.0	500	72	39	1	8	727.1	735.0
20	9	50	0.5	100	72	0	0	24	26.8	28.2
20	9	50	0.5	100	72	19	1	24	1255.8	1266.0
20	9	50	0.5	100	72	39	1	24	1696.9	1682.2
20	9	100	0.5	200	8	0	0	24	173.0	180.2
20	9	100	0.5	200	8	19	1	24	6693.6	6475.0
20	9	100	0.5	200	8	39	1	24	6793.9	6800.6

Table D.4 – Data for Section 5.7, Figures 5.34 and 5.36, part 7

		Disk		String		Order			MTTDL	
G	N+1	MTTF	MTTR	MTTF	MTTR	THRES	SIZE	TIME	EST	SIMUL
20	13	800	8.0	100	72	0	0	72	125.9	121.0
20	13	800	8.0	100	72	19	1	72	864.8	878.1
20	13	800	8.0	100	72	39	1	72	905.3	868.6
20	17	20	0.5	1000	8	0	0	72	1.5	1.5
20	17	20	0.5	1000	8	19	1	72	139.6	136.8
20	17	20	0.5	1000	8	39	1	72	141.5	147.2
20	21	50	2.0	500	24	0	0	24	10.6	10.9
20	21	50	2.0	500	24	19	1	24	122.9	118.2
20	21	50	2.0	500	24	39	1	24	124.2	122.7
50	2	20	8.0	500	24	0	0	8	226.7	221.3
50	2	20	8.0	500	24	49	1	8	463.7	455.2
50	2	20	8.0	500	24	99	1	8	463.8	478.9
50	2	20	8.0	1000	8	0	0	72	55.9	53.6
50	2	20	8.0	1000	8	49	1	72	481.4	472.0
50	2	20	8.0	1000	8	99	1	72	481.5	476.5
50	4	50	1.0	100	72	0	0	72	32.9	36.8
50	4	50	1.0	100	72	49	1	72	1715.9	1680.5
50	4	50	1.0	100	72	99	1	72	2061.3	2032.3
50	9	50	0.5	200	8	0	0	8	56.6	58.2
50	9	50	0.5	200	8	49	1	8	918.0	890.0
50	9	50	0.5	200	8	99	1	8	923.3	932.8
50	9	200	8.0	50	8	0	0	72	25.0	27.1
50	9	200	8.0	50	8	49	1	72	134.2	132.9
50	9	200	8.0	50	8	99	1	72	135.1	128.0
50	13	200	1.0	500	72	0	0	24	88.7	90.0
50	13	200	1.0	500	72	49	1	24	2694.7	2701.3
50	13	200	1.0	500	72	99	1	24	2828.7	2853.5
50	13	200	2.0	50	24	0	0	8	33.1	34.4
50	13	200	2.0	50	24	49	1	8	240.1	235.9
50	13	200	2.0	50	24	99	1	8	247.0	242.2
50	13	200	8.0	1000	24	0	0	24	118.8	118.6
50	13	200	8.0	1000	24	49	1	24	458.3	465.8
50	13	200	8.0	1000	24	99	1	24	458.7	459.8
50	17	50	0.5	1000	24	0	0	24	8.1	8.3
50	17	50	0.5	1000	24	49	1	24	328.8	315.4
50	17	50	0.5	1000	24	99	1	24	334.4	340.4
50	17	100	2.0	500	2	0	0	24	25.1	26.0
50	17	100	2.0	500	2	49	1	24	262.5	257.3
50	17	100	2.0	500	2	99	1	24	262.7	261.8
50	17	200	1.0	100	8	0	0	24	32.9	34.1
50	17	200	1.0	100	8	49	1	24	539.1	534.2
50	17	200	1.0	100	8	99	1	24	550.1	572.5

Table D.4 – Data for Section 5.7, Figures 5.34 and 5.36, part 8

G	N+1	Disk		String		Order			MTTDL	
		MTTF	MTTR	MTTF	MTTR	THRES	SIZE	TIME	EST	SIMUL
50	17	400	1.0	500	168	0	0	8	99.6	101.8
50	17	400	1.0	500	168	49	1	8	3810.0	3937.9
50	17	400	1.0	500	168	99	1	8	4428.7	4305.3
50	21	200	8.0	500	72	0	0	24	28.8	30.2
50	21	200	8.0	500	72	49	1	24	131.4	134.1
50	21	200	8.0	500	72	99	1	24	132.7	136.4

Table D.4: Raw Data for Figures 5.34 and 5.36 in Section 5.7. *These 300 data sets each represent a comparison between the estimate of mean time to data loss (MTTDL) derived in Section 5.6 and the corresponding estimate generated by simulation. In this model, there are both dependent disk failures and on-line spare disks. These parameter sets correspond to 100 collections of three sets. In each group of three, one parameter set has no on-line spare disks, one has one string of on-line spare disks, and one set has two strings of on-line spare disks. The values for the remaining parameters in each group of three sets were selected by first generating the cross-product of sets of conceivable values for each parameter, then selecting approximately 100 sets at random. The units for "Disk MTTR," "String MTTR," and "Order Time" is hours, and for "Disk MTTF," "String MTTF," and both "MTTDL" columns, the units is 1,000 hours. Expressions for "MTTDL Estimated" are found in Equations 5.16, 5.19, 5.30, 5.31, 5.32, 5.33, 5.34, and 5.35. Simulated MTTDL has a 95% confidence interval that extends ±5% of its value.*

Table D.5 – Data for Section 5.7, Figure 5.40, part 1

		Disk		String		Order			Simul.	
G	N+1	MTTF	MTTR	MTTF	MTTR	THRES	SIZE	TIME	MTTDL	REL10
7	11	150	1.0	150	72	0	0	72	130.9	0.499
7	11	150	1.0	150	72	0	1	72	389.9	0.805
7	11	150	1.0	150	72	0	2	72	429.1	0.812
7	11	150	1.0	150	72	1	1	72	464.5	0.825
7	11	150	1.0	150	72	0	3	72	478.3	0.846
7	11	150	1.0	150	72	1	2	72	518.9	0.843
7	11	150	1.0	150	72	2	1	72	546.6	0.852
7	11	150	1.0	150	72	0	4	72	556.3	0.867
7	11	150	1.0	150	72	2	2	72	626.2	0.869
7	11	150	1.0	150	72	3	1	72	710.1	0.885
7	11	150	1.0	150	72	0	5	72	635.9	0.878
7	11	150	1.0	150	72	2	3	72	734.1	0.887
7	11	150	1.0	150	72	4	1	72	860.7	0.886
7	11	150	1.0	150	72	0	6	72	801.6	0.897
7	11	150	1.0	150	72	3	3	72	983.0	0.912
7	11	150	1.0	150	72	5	1	72	1215.1	0.925
7	11	150	1.0	150	72	0	7	72	1146.6	0.929
7	11	150	1.0	150	72	3	4	72	1790.5	0.955
7	11	150	1.0	150	72	6	1	72	6825.9	0.981
7	11	150	1.0	150	72	0	8	72	1394.7	0.945
7	11	150	1.0	150	72	4	4	72	2841.4	0.961
7	11	150	1.0	150	72	7	1	72	8117.9	0.992
7	11	150	1.0	150	72	0	10	72	2019.5	0.963
7	11	150	1.0	150	72	5	5	72	5105.7	0.983
7	11	150	1.0	150	72	9	1	72	8200.2	0.989
7	11	150	1.0	150	72	0	12	72	2454.4	0.967
7	11	150	1.0	150	72	6	6	72	8462.7	0.992
7	11	150	1.0	150	72	11	1	72	8628.7	0.991
7	11	150	1.0	150	72	0	14	72	3165.7	0.974
7	11	150	1.0	150	72	7	7	72	8867.2	0.992
7	11	150	1.0	150	72	13	1	72	8730.8	0.990

Table D.5: Raw Data for Figure 5.40 in Section 5.7. *These 31 data sets describe estimates of mean time to data loss (MTTDL) generated by simulation and presented in Figure 5.40 of Section 5.7. This data represents the reliability of various configurations of the strawman disk array first introduced in Table 5.1. The units for "Disk MTTR," "String MTTR," and "Order Time" is hours, and for "Disk MTTF," "String MTTF," and "Simul MTTDL" columns, the units is 1,000 hours. Using Equations 5.16, 5.19, 5.30, 5.31, 5.32, 5.33, 5.34, and 5.35, estimates for MTTDL are 133,000 hours with no spares, 6,595,000 hours with one string of spare disks (7) and a threshold of 6 spare disks, and 8,666,000 hours with two strings of spare disks (14) and a threshold of 13 spare disks. Simulated MTTDL has a 95% confidence interval that extends ±5% of its value.*