Performance Modeling and Analysis of Disk Arrays

by

Edward Kihyen Lee

B.S. (University of California at Los Angeles) 1987
M.S. (University of California at Berkeley) 1990

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY of CALIFORNIA at BERKELEY

Committee in Charge

Professor Randy H. Katz, Chair
Professor David A. Patterson
Professor Ronald Wolff

1993

The dissertation of Edward Kihyen Lee is approved:

Chair	Date
	Date
	Date

University of California at Berkeley

1993

Performance Modeling and Analysis of Disk Arrays

by

Edward Kihyen Lee

Abstract

Performance Modeling and Analysis of Disk Arrays

by

Edward Kihyen Lee

Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Randy H. Katz, Chair

As disk arrays become widely used, tools for modeling and analyzing the performance of disk arrays become increasingly important. In particular, accurate performance models and systematic analysis techniques, combined with a thorough understanding of the expected workload, are invaluable in both configuring and designing disk arrays. Unfortunately, disk arrays, like many parallel systems, are difficult to model and analyze because of *queueing* and *fork-join synchronization*. In this dissertation, we present an analytic performance model for non-redundant disk arrays and a new technique based on *utilization profiles* for analyzing the performance of redundant disk arrays. In both cases, we provide applications of our work. We use the analytic model to derive an equation for the optimal size of data striping in disk arrays, and we apply utilization profiles to analyze the

performance of RAID-II, our second disk array prototype. The results of the analysis are used to answer several performance related questions about RAID-II and to compare the performance of RAID-II to RAID-I, our first disk array prototype.

Professor Randy H. Katz, Chair   Date

To my parents.

Contents

List of Figures

List of Tables

Acknowledgements

My experiences as a graduate student at UC Berkeley have been the most rewarding of my educational and personal life. Never before have I worked with so many bright, motivated and supportive individuals. The experience has been both exhilarating and humbling, and I feel extremely fortunate for having worked with them.

First and foremost, I would like to thank my advisor, Prof. Randy Katz, for supporting my work and for his technical and professional guidance. He has allowed me to pursue my own interests while also giving me the opportunity to contribute to group projects. He has contributed immeasurably to both my professional and personal development.

I would like to thank Prof. David Patterson for his support and especially for his philosophical guidance. I will never forget his timeless comments and advice. I have frequently thought of him as my second advisor and guardian angel.

I would like to thank Prof. John Ousterhout for his invaluable advice in the organization and presentation of talks and his technical guidance in the development of software for RAID.

I would like to thank Prof. Ronald Wolff for his courses on queueing theory, the conversations I have had with him regarding analytic models for disk arrays, and his participation on my dissertation committee.

I would like to thank the other members of the RAID Group, Peter Chen, Ann Drapeau, Garth Gibson, Ken Lutz, Bob Miller, Ethan Miller, Srini Seshan, and Theresa Lessard-Smith. I especially owe Peter Chen, Ann Drapeau and Garth Gibson my gratitude for the many interesting discussions that have influenced my thinking, their collaboration on the RAID project, and their friendship. Ken Lutz, with his attention to details and valuable experience, kept the hardware side of the project running smoothly. Without the dedication and efforts of Bob Miller and Theresa Lessard-Smith, the orderly organization of the project would be overcome by chaos.

I would like to thank members of the Sprite operating system group and the Postgress database group with whom I have had the pleasure to work. In particular I would like to thank Mary Baker, Fred Douglis, John Hartman, Wei Hong, Mendel Rosenblum, Margo Seltzer, Ken Shirriff, Mark Sullivan and Brent Welch. I would especially like to thank Mary Baker, John Hartman, Mendel Rosenblum and Ken Shirriff for explaining the operation of the Sprite operating system, and their support of the software components of the RAID project.

I would like to thank our government and industrial affiliates, Array Technologies, DARPA/NASA (NAG2-591), DEC, Hewlett-Packard, IBM, Intel Scientific Computers, California MICRO, NSF (MIP 8715235), Seagate, Storage Tek, Sun Microsystems and Thinking Machines Corporation for their support.

Last but not least, I would like to thank my family for their encouragements, understanding and love.

Chapter 1

Introduction

1.1 Motivation

In recent years, interest in RAID, Redundant Arrays of Inexpensive Disks, as high-performance, reliable secondary storage systems has grown explosively. The driving force behind this phenomenon is the sustained exponential improvements in the performance and density of semiconductors. Improvements in semiconductor technology makes possible faster microprocessors and larger primary memory systems, which in turn require larger, higher-performance secondary storage systems. More specifically, the consequences of improvements in microprocessors on secondary storage systems can be broken into quantitative and qualitative components.

On the quantitative side, Amdahl's Law predicts that, in general, large improvements in microprocessors will result in only marginal improvements in overall system performance unless accompanied by corresponding improvements in secondary storage systems. Unfortunately, while RISC microprocessor performance has been improving 50% per year [33], disk access times, which depend on improvements of mechanical systems, have been improving less than 10 % per year. Over the same time interval, disk transfer rates, which track improvements in both mechanical systems and magnetic media densities, have improved at a significantly faster rate of 20 % per year. Assuming semiconductor and disk technologies continue their current trends, we must conclude that the performance gap between microprocessors and magnetic disks will continue to widen. Thus, as long as secondary storage systems are based on magnetic disks, technologies such as disk arrays will become increasingly important.

In addition to the quantitative effect, a second, perhaps more important, qualitative effect is driving the need for higher-performance secondary storage systems. As microprocessors become faster, they make possible new applications and greatly expand the scope of existing applications. In particular, applications such as video, hypertext and multi-media are becoming common. Even in existing application areas such as CAD and scientific computing, faster microprocessors make it possible to process and generate increasingly larger datasets. This shift in applications combined with a trend toward large, shared, high-performance, network-based storage systems is causing us to reevaluate the way we design and use secondary storage systems.

Disk arrays, which organize many independent disks into a large, high-performance logical disk, are a natural solution to the problem. They offer several advantages over independent disks. First, disk arrays stripe data across multiple disks and access them in parallel to achieve both higher data transfer rates on large data accesses and higher I/O rates on small data accesses. Second, data striping results in uniform load balancing across all the disks, eliminating hot spots that saturate a small number of disks while the majority of disks sit idle. Third, by adding redundancy, disk arrays can protect against disk failures and continue to operate while the contents of the failed disk is reconstructed onto a spare disk. Fourth, a few large logical disks are easier to administer than hundreds of small independent disks and facilitate the implementation of extremely large, shared file systems composed of very large objects.

Given the important role disk arrays will play in the future, tools for modeling and analyzing the performance of disk arrays become increasingly important. In particular, accurate performance models and systematic analysis techniques, combined with a thorough understanding of an installation's workload, are invaluable in both configuring and designing disk arrays. Unfortunately, disk arrays, like many parallel systems, are difficult to model and analyze because of *queueing* and *fork-join synchronization*; a logical request to a disk array is split-up into multiple disk requests which are queued and serviced independently; however, the logical request is not completed until all the corresponding disk requests are completed. Either queueing or fork-join synchronization by itself is tractable but the combination of the two is intractable. This is a classic problem in the modeling of queueing systems that currently has no general solution [2, 10, 11, 39]. The modeling and analysis of such systems is highly dependent on the characteristics of the particular system and requires the judicious use of approximations and simplifying assumptions.

In this dissertation, we develop an analytic performance model for non-redundant disk arrays by approximating the queueing and fork-join overheads. We also develop a new empirical technique based on *utilization profiles* for analyzing the performance of redundant disk arrays. The technique is not specific to disk arrays and can be used to analyze a wide variety of systems. In both cases, we provide applications of our work. We use the analytic model to derive an equation for the optimal size of data striping in disk arrays and utilization profiles are used to analyze the performance of RAID-II, our second hardware disk-array prototype. The results of the analysis are used to answer several performance questions about RAID-II and to compare the performance of RAID-II to RAID-I, our first disk array prototype.

1.2 Contributions

The primary contributions of this work are the development of an analytic performance model for non-redundant disk arrays, the application of the analytic performance model to determine a mathematical equation for the optimal size of data striping, the development of a general performance analysis technique based on *utilization profiles*, and the application of the technique to analyze the performance of RAID-II, our second disk array prototype.

In this dissertation, we develop, validate and apply an analytic performance model for disk arrays by modeling them as closed queueing systems and approximating the queueing and fork-join overheads. Our model is different from previous analytic models of disk arrays [4, 8, 19, 20, 26, 32, 36] for the following reasons. First, we use a closed queueing model with a fixed number of processes. Previous analytic models of disk arrays used open queueing models with Poisson arrivals [8, 19, 26]. A closed model more accurately models the synchronous I/O behavior of scientific, time-sharing and distributed systems. In such systems, processes tends to wait for previous I/O requests to complete before issuing new I/O requests, whereas in transaction oriented systems, I/O requests are issued randomly in time regardless of whether the previous I/O requests have completed.

Second, to the best of our knowledge, we present the first analytic model for disk arrays that have both queueing at individual disks and the fork-join synchronization introduced by data striping, over a continuum of request sizes. Previous analytic models for disk arrays with both queueing and fork-join synchronization model only small or large requests [8, 26].

We have found the analytic model especially useful for comparative studies that focus on the relative performance of different disk array configurations, such as determining the optimal size of data striping. The model can also be used in calculating quantitative price/performance metrics for disk arrays under a range of system and workload parameters, in identifying important factors in the performance of disk arrays, and has provided us with useful insights in characterizing the performance of disk arrays.

The model, of course, is not without limitations. Although our analytic model handles both reads and writes to non-redundant disk arrays and reads to a subclass of redundant disk arrays, it does not handle writes to redundant disk arrays. Also, the workload model is not expressive enough to accurately model real workloads. Thus, although the model can be used to accurately predict the performance of real disk arrays under the synthetic workloads used in this paper, its prediction of performance under real workloads is highly approximate. A final limitation is that we only model the disk components of a disk array. Real disk arrays have many other components such as strings, I/O controllers, I/O busses and CPU's that affect the performance of the system.

Due to the above limitations, which are characteristic of analytic models in general, it is desirable to have more general tools for analyzing the performance of disk arrays. In particular, we will present an analysis technique that combines empirical performance models with statistical techniques to summarize the performance of disk arrays over a wide range of system and workload parameters. The analysis technique is generalizable to a wide variety of systems, requires minimal information about the system to be analyzed, does not require measurements of internal system resources, and produces results that are compact, intuitive, and easily compared with those of similar systems.

Finally, we will apply the analysis technique to analyze the performance of RAID-II, our second disk array prototype. We will show how it can be used to determine the limits of system performance, to determine the bottleneck resource for a given system configuration and workload, to eliminate resource bottlenecks, to prevent excessive queueing delays at specified resources, and to compare similar systems.

1.3 Organization

This document is concerned with the modeling and analysis of disk arrays. Figure 1.1 schematically illustrates the two processes. Modeling consists of formulating the system of interest as a formal specification, deriving a solution to the specification, and validating the solution. Analysis consists of designing experiments to study the system of interest, performing the experiments either via measurement or simulation, and interpreting the resulting data. In both processes, successive steps will uncover new information that

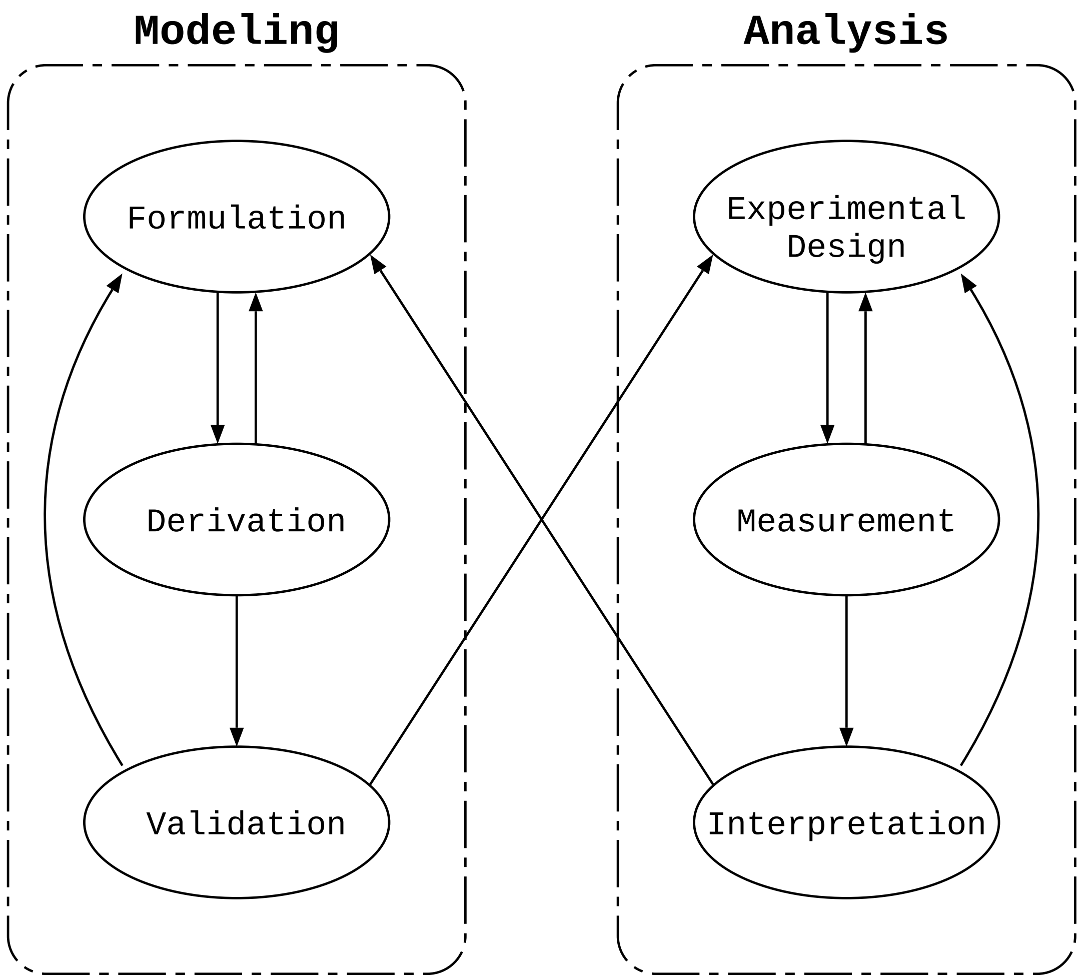


Figure 1.1: The Modeling and Analysis Processes.

may require back-tracking and repeating previous steps.

As illustrated in Figure 1.1, modeling and analysis are mutually dependent processes; the validation of a model requires analysis and the interpretation of data requires a model to guide the interpretation. Without analysis, we can never know the accuracy of our model and without modeling, we cannot draw general conclusions from our analysis.

Although modeling and analysis are interdependent, most studies emphasize one over the other. Studies interested in the general properties of a certain type of system will emphasize modeling while studies interested in the properties of a specific system will emphasize analysis. This document is no exception to this rule. Modeling is emphasized in Chapter 4, which develops an analytic performance model for disk arrays, validates the model via simulation, and applies the model to derive an equation for the optimal size of data striping in disk arrays. Analysis is emphasized in Chapter 5, which develops a new technique for analyzing systems based on *utilization profiles*, applies the technique to analyze the performance of RAID-II, our second disk array prototype, and uses the results of the analysis to answer several performance questions about RAID-II. These are the two core chapters of this dissertation.

The remaining chapters are organized as follows: Chapter 2 provides an overview of disk arrays and a summary of the related work in the field, Chapter 3 describes the empirical methods that will be used in both the modeling and analysis of disk arrays, and Chapter 6 summarizes the main results of this dissertation and concludes with our experiences in modeling and analyzing disk arrays.

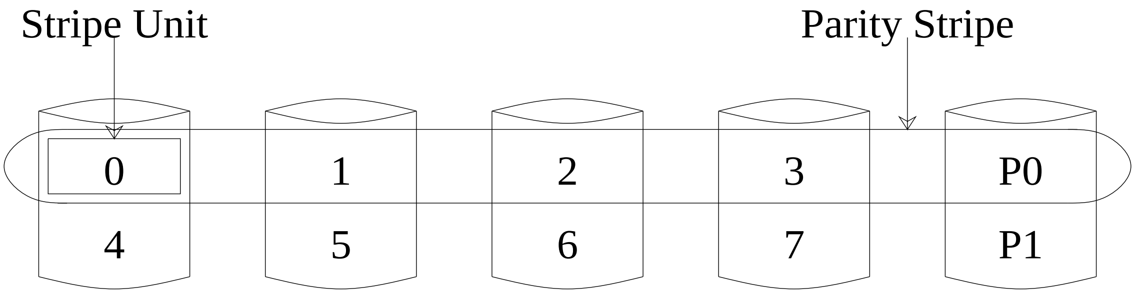
Chapter 2

Overview

Redundant Arrays of Inexpensive Disks (RAID) employ two orthogonal concepts, data striping and redundancy, for improved performance and reliability. Data striping provides high-performance by increasing parallelism and improving load-balance. Redundancy improves reliability by allowing RAID to operate in the face of disk failures without data loss. Although the basic concepts of data striping and redundancy are simple, selecting between the many possible data striping and redundancy schemes involves complex tradeoffs between reliability, performance and cost. This chapter defines basic terms for describing RAID, the commonly accepted RAID organizations, how a RAID services I/O requests, and compares their performance and cost.

2.1 Basic Definitions

Figure 2.1 illustrates and defines the terms *stripe unit* and *parity stripe* in the context of a simple redundant disk array consisting of five disks. In the disk array illustrated



Stripe unit is the unit of data interleaving, that is, the amount of data that is placed on a disk before data is placed on the next disk. Stripe units typically range from a sector to a track in size (512 bytes to 64 kilobytes).

Parity stripe is a collection of data stripe units along with its corresponding parity stripe unit.

Figure 2.1: RAID Definitions.

in Figure 2.1, the parity stripe units are simply grouped together on a single disk but many other RAID organizations distribute the parity in complex ways across the disks in the disk array.

2.2 The RAID Taxonomy

Most redundant disk array organizations can be distinguished based on the granularity of data interleaving and the method and pattern in which the redundant information is computed and distributed, respectively, across the disk array. Data interleaving can be characterized as either fine-grained or coarse-grained. Fine-grained disk arrays conceptually interleave data at a small unit such that all I/O requests, regardless or their size, access all the disks in the disk array. This results in high data transfer rates for all I/O requests but only one I/O request can be serviced at a time. Coarse-grained disk arrays interleave data at a large unit so that small I/O requests access a few disks but large requests access all the disks in the disk array. This allows many small requests to be serviced simultaneously but allows large requests to use all the disks.

The incorporation of redundancy in disk arrays brings up two somewhat orthogonal problems. The first problem is selecting the method for computing the redundant information. Most redundant disk arrays today use parity, but there are some that use Hamming or Reed-Solomon codes [12]. The second problem is selecting a method for distributing the redundant information across the disk array. Although there are an unlimited number of patterns in which redundant information can be distributed, we can roughly distinguish between two different distributions schemes: those that concentrate redundant information on a few disks, and those that distributed redundant information uniformly across all disks. Schemes that uniformly distribute redundant information are generally more desirable because they avoid contention for the redundant information, which must be updated on each write to the disk array.

2.2.1 Non-Redundant (RAID Level 0)

The simplest redundancy scheme is to have no redundancy; this is clearly the lowest cost redundancy scheme. This scheme has the best write performance, because it never needs to update redundant information. Surprisingly, it does not have the best read performance. Redundancy schemes such as mirroring, which duplicate data, can perform better on small reads by selectively scheduling requests on the disk with the shortest expected seek and rotational delays. Without redundancy, any single disk failure will result in data-loss, making this the least reliable redundancy scheme. Non-redundant disk arrays are widely used in supercomputing environments where performance rather than reliability

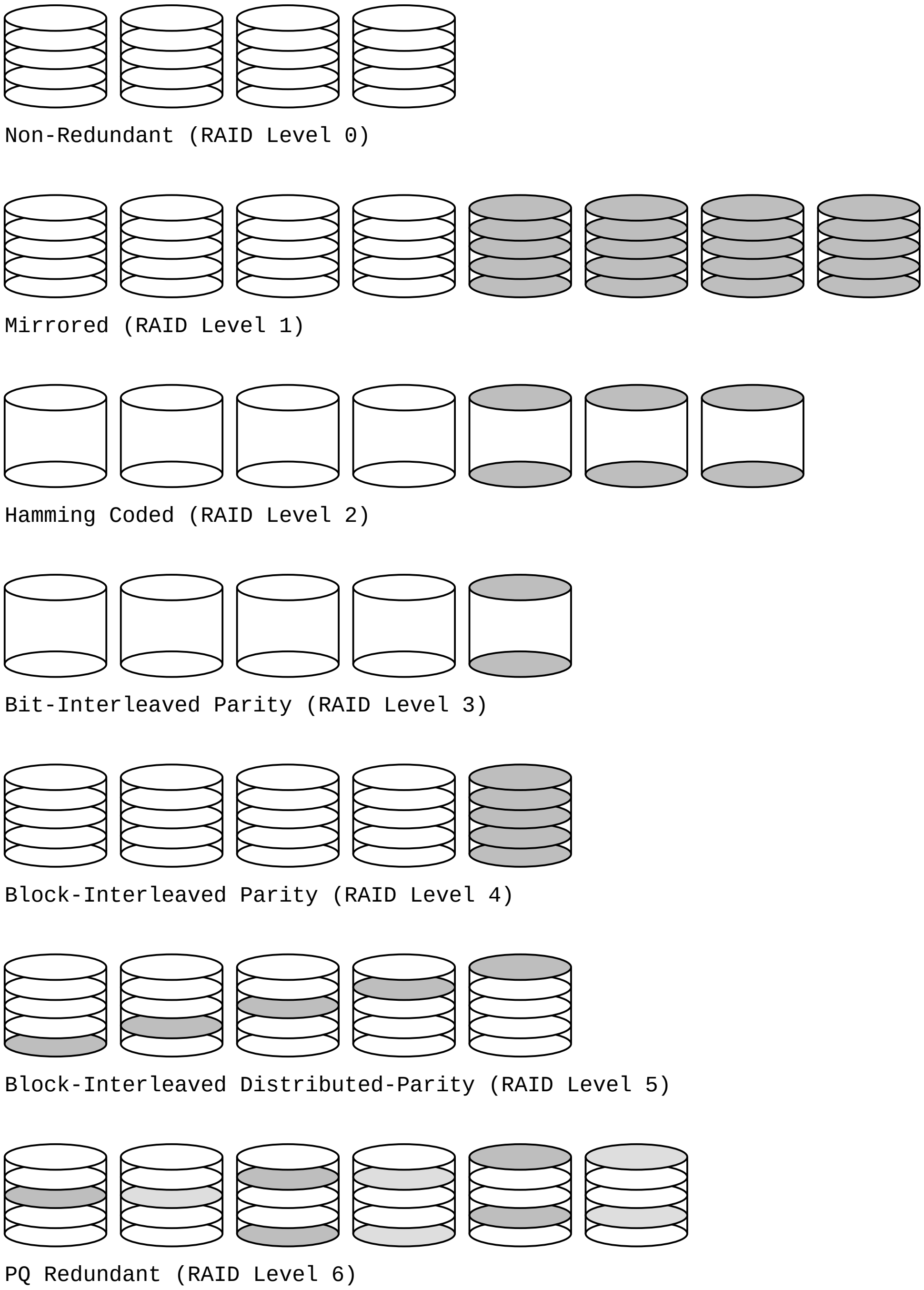


Figure 2.2: RAID Levels 0 Through 6. All RAID levels are illustrated at a user capacity of four disks. Disks with multiple platters indicate block-level striping while disks without platters indicate bit-level striping. The shaded platters represent redundant information.

is the primary concern.

2.2.2 Mirrored (RAID Level 1)

Mirroring is the simplest redundancy scheme. All data are duplicated so that a complete backup copy is available when a disk fails. The duplication of data results in poor storage utilization, but provides high reliability and allows small read requests to be serviced by one of two disks. This usually results in more efficient disk scheduling. Unfortunately, each write results in two disk accesses. This is a disadvantage compared to other RAID schemes which have lower write overheads for large write requests. Mirroring is frequently used in database systems where availability and transaction rate are more important than write bandwidth or storage efficiency.

2.2.3 Hamming Coded (RAID Level 2)

Hamming-coded disk arrays use a redundancy scheme similar to that used in main memory systems [12]. A collection of paity disks computes parity over different subsets of the data and parity disks. By noting which parity disks are in error, one can determine both the disk that has failed and the data that was stored on the failed disk. The storage efficiency for Hamming-coded disk arrays can be much better than for mirrored disk arrays. In practice, Hamming-coded disk arrays are not common because redundancy schemes such as bit-interleaved parity provide similar reliability at better performance and cost.

2.2.4 Bit-Interleaved Parity (RAID Level 3)

In a bit-interleaved parity disk array, data is interleaved bit-wise over the data disks and a single parity disk is added to tolerate any single disk failure. Each read request accesses all the data disks and each write request accesses all the data disks and parity disk. Thus, only one read or write request can be serviced at a time. Because the parity disk contains only parity and no data, the parity disk cannot participate in reads, resulting in slightly lower read performance than for redundancy schemes that distribute the parity and data over all disks. Bit-interleaved parity disk arrays are frequently used in high-bandwidth applications that do not require many small I/O's per second.

2.2.5 Block-Interleaved Parity (RAID Level 4)

The block-interleaved parity disk array is similar to the bit-interleaved parity disk array except that data is interleaved across disks in blocks rather than bits. Small read requests access only a single data disk and small write requests access a single data disk and the parity disk. Each small write request results in a read-modify-write operation. The old data and parity is read from the corresponding data and parity disks and xored—checksum modulo two—with the new data to compute the new parity. The new data and parity are then written out to the corresponding disks. Thus, each small write request results in four disk accesses: two reads and two writes. Because a block-interleaved parity disk array has only one parity disk, which must be updated on all write operations, the parity disk can easily become a bottleneck. Because of this limitation, the distributed-parity disk array is universally preferred over the non-distributed parity disk array.

2.2.6 Block-Interleaved Distributed-Parity (RAID Level 5)

The block-interleaved distributed-parity disk array eliminates the parity disk bottleneck in RAID level 4 disk arrays by distributing the parity uniformly over all the disks. A frequently overlooked advantage to distributing the parity is that it also distributes the data over all the disks. This allows all disks to participate in servicing read operations, in contrast to redundancy schemes with dedicated parity disks that cannot participate in servicing read requests. Block-interleaved distributed-parity disk arrays have one of the best small read, large read and large write performance of any redundant disk array. Small write requests are somewhat inefficient compared with redundancy schemes such as mirroring, however, because of the read-modify-write operations.

The exact method used to distribute parity in block-interleaved distributed-parity disk arrays can make a significant impact on performance. Figure 2.3 illustrates four simple parity distribution schemes. The symmetric parity distributions are obtained by numbering each striping unit starting from the parity striping unit; the asymmetric parity distributions are obtained by numbering each striping unit starting from the left-most striping unit. In earlier work, we shows that, of the four parity distributions illustrated in Figure 2.3, the left-symmetric parity distribution is the most desirable for several reasons [22]. The most important reason is that a sequential traversal of the striping units in the left-symmetric parity placement will access each disk once before accessing any disk twice. This reduces disk conflicts when servicing large requests.

2.2.7 PQ Redundant (RAID Level 6)

Asymmetric — Right

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4

Symmetric — Right

P0	0	1	2	3
7	P1	4	5	6
10	11	P2	8	9
13	14	15	P3	12
16	17	18	19	P4

Asymmetric — Left

0	1	2	3	P0
4	5	6	P1	7
8	9	P2	10	11
12	P3	13	14	15
P4	16	17	18	19

Symmetric — Left

0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

Figure 2.3: RAID Level 5 Parity Placements. Each square corresponds to a stripe unit. Each column of squares corresponds to a disk. Only the minimum repeating pattern is shown.

0	1	2	3	P0	Q0
6	7	P1	Q1	4	5
P2	Q2	8	9	10	11
12	13	14	15	P3	Q3
18	19	P4	Q4	16	17

Figure 2.4: RAID Level 6 Left Symmetric Parity Placement. Each square corresponds to a stripe unit. Each column of squares corresponds to a disk.

The PQ redundant disk array is identical to the RAID Level 5 disk array except that it uses two disks per redundancy group instead of one. The P disk stores parity while the Q disk stores a Reed-Solomon code [12]. The Reed-Solomon code in combination with the parity allows the recovery of data on any two disks. Figure 2.4 illustrates a variation of the left-symmetric parity placement when extended to PQ redundant disk arrays.

2.3 RAID Access Modes

The access modes for bit-interleaved disk arrays are simple; all the data and parity in a parity stripe are read and written together. The access modes for block-interleaved disk arrays are more complex since only a part of a parity stripe may be read or written. During normal operation, a block-interleaved RAID uses three different access modes to service I/O requests. Figure 2.5 illustrates the read, read-modify-write and reconstruct-write access modes.

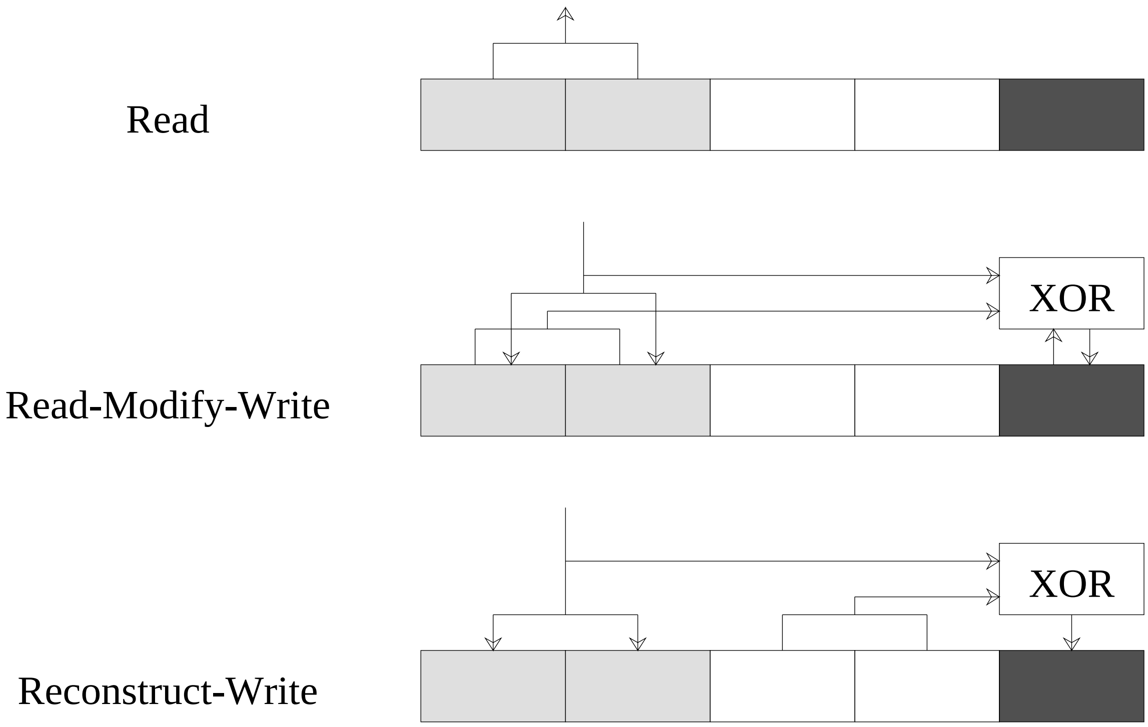


Figure 2.5: Block-Interleaved RAID Access Modes. The lightly shaded regions correspond to data that is to be read or written. The dark regions correspond to parity.

2.3.1 Read

Read is the simplest access mode. It simply reads the requested data from the appropriate disks.

2.3.2 Read-Modify-Write

The read-modify-write access mode is used if a small portion of the stripe is written. In such a case, fewer disk accesses are required if the parity is updated incrementally by xoring the new data, old data and old parity than by xoring the new data with the rest of the parity stripe. The read-modify-write method is executed in three steps:

1. Read old data and old parity.

2. Compute new parity.

3. Write new data and new parity.

2.3.3 Reconstruct-Write

The reconstruct-write access mode is used if a large portion of the stripe is written. Usually, the reconstruct-write method is preferred to the read-modify-write method if half or more of the parity stripe is being written. The reconstruct-write method is executed in three distinct steps.

1. Read data from rest of the parity stripe.

2. Compute parity.

3. Write new data and new parity.

If an exact parity stripe is being written, step one is trivial and does not require disk accesses.

2.4 Comparison of RAID levels

There are many ways to compare RAID organizations. To a large extent, the metrics used depend on the intended use of the systems. In time-sharing applications, the primary metric may be user capacity—disk capacity excluding the redundant information—per dollar; in transaction processing applications the primary metric may be I/O rate per dollar; and in scientific applications, the primary metric may be bandwidth per dollar. In

	Small Read	Small Write	Large Read	Large Write	Storage Efficiency
RAID 0	1	1	1	1	1
RAID 1	1	1/2	1	1/2	1/2
RAID 3	1/G	1/G	(G-1)/G	(G-1)/G	(G-1)/G
RAID 5	1	max(1/G,1/4)	1	(G-1)/G	(G-1)/G
RAID 6	1	max(1/G,1/6)	1	(G-2)/G	(G-2)/G

Table 2.1: Throughput per dollar relative to RAID level 0. G is the parity group size in disks. For small requests, throughput is measured in I/O's per second while for large requests, throughput is measured in bytes per second. Storage efficiency is the fraction of total storage capacity the is available to the user.

certain heterogeneous systems, both I/O rate and bandwidth may be important. In this section, we will use throughput, either I/O rate or bandwidth, per dollar as our primary performance metric in comparing RAID organizations.

Even after the appropriate metrics have been chosen, comparisons between RAID levels one through six can be confusing. Frequently, a RAID level specifies not a specific implementation of a system but its configuration and use. For example, a RAID level 5 disk array with a parity group size of two is equivalent to RAID level 1 with the exception that in a mirrored disk array, certain disk scheduling optimizations on reads are performed that generally are not implemented for RAID level 5 disk arrays. Analogously, a RAID level 5 disk array can be configured to operate equivalently to a RAID level 3 disk array by choosing a striping unit such that the smallest I/O request always accesses a full parity stripe of data. RAID level 1 and RAID level 3 disk arrays can be viewed as a subclass of RAID level 5 disk arrays. Since RAID level 2 and RAID level 4 disk arrays are, practically speaking, in all ways inferior to RAID level 5 disk arrays, we will exclude them from our comparisons to accentuate the differences between RAID levels 0, 1, 3, and 5.

Table 2.1 tabulates the maximum throughput per dollar relative to RAID level 0. The cost of each system is assumed to be proportional to the total number of disks in the disk array; this is somewhat inaccurate since more complex RAID organizations may require more expensive hardware and software. The table illustrates, for example, that given equivalent cost RAID level 0 and RAID level 1 systems, the RAID level 1 system can sustain half the number of small writes per second that a RAID level 0 system can sustain. Equivalently, we can say that the cost of small writes is twice as expensive in a RAID level 1 system as in a RAID level 0 system. In addition to performance, the table shows the storage efficiency of each disk array organization. The storage efficiency is inverse the cost of each unit of user capacity relative to a RAID level 0 system. Note that the storage efficiency is the same as the performance/cost for large writes.

As expected, Table 2.1 shows that performance/cost of RAID level 1 is equivalent to the performance/cost of RAID level 5 when the parity group size is equal to two. The performance/cost of RAID level 3 is always less than or equal to the performance/cost of RAID level 5. This is expected given that RAID level 3 is a subclass of RAID level 5 derived by restricting the striping unit size such that requests access exactly a parity stripe. Since the configuration of RAID level 5 is not subject to such a restriction, the performance/cost of RAID level 5 can never be less than that of an equivalent RAID level 3. In the rest of this dissertation, we will deal exclusively with RAID level 0 and RAID level 5 disk arrays, the two most "efficient" disk array organizations.

2.5 Related Work

This section describes related work in the modeling and analysis of disk arrays. The related work most relevant to this dissertation concerns the analytic modeling of disk arrays. The analysis work presented in this dissertation is difficult to compare directly with other related work because we develop a new analysis technique and use it to analyze a unique, real disk array prototype. Still, there are a few measurement studies that are relevant to our work. For the benefit of our readers, we will also describe disk array research that is not directly relevant to this dissertation.

2.5.1 Related Modeling Work

Analytic queueing models of disk arrays are difficult to formulate because of queuing and fork-join synchronization, that is, a request to a disk array is broken up into independent disk requests that must all complete. Related work in analytic models for disk arrays fall into one of three categories: models that ignore queueing, models that ignore fork-join synchronization, and models that handle both queueing and fork-join synchronization using approximate techniques.

Analytic models that ignore queueing are frequently used to calculate expected minimum response times and maximum throughputs of systems. Such metrics are useful for estimating the limits of system performance but are applicable to only extreme workloads when the load in the system is either very light or very heavy. Salem and Garcia-Molina [36] derive expected minimum response time equations to study the benefits of data striping in synchronized (disk arm and rotational position) non-redundant disk arrays. They also show the effects of several low-level disk optimizations on the response time for individual disks. Bitton and Gray [4] derive expected disk seek times for unsynchronized mirrored disk arrays. Kim and Tantawi [20] derive service time distributions for unsynchronized, bit-interleaved, non-redundant disk arrays. Patterson, Gibson and Katz [32] derive maximum throughput estimates for RAID levels 0 through 5.

Analytic models that handle queueing but ignore fork-join synchronization tend to be more sophisticated than models that ignore queueing. Such models are frequently used to model synchronized, bit-interleaved disk arrays, which can be accurately modeled without fork-join synchronization. Kim [19] models synchronized bit-interleaved disk arrays as an $M/G/1$ queueing system and shows that bit-interleaved disk arrays provide lower service times and better load balancing but decrease the number of requests that can be serviced concurrently. Chen and Towsley [8] analytically model RAID level 1 and RAID level 5 disk arrays. Bounds are used to model the queueing and fork-join synchronization in RAID level 1 disk arrays. The fork-join synchronization overhead is ignored for small write requests, resulting in an optimistic model. Large requests are modeled by using a single queue for all the disks in the disk array. The main drawback with the model is that it only models small and large requests. Moderate sized requests are ignored.

Systems that exhibit both queueing and fork-join synchronization are beyond the current capabilities of queueing theory. Few analytic models for disk arrays attempt to model both elements. Menon and Mattson [26] formulate an approximate model for RAID level 5 systems under transaction processing workloads. Their model is based on a mathematical relationship for two server $M/M/1$ systems presented by Nelson and Tantawi [31].

The work presented by Menon and Mattson is flawed, however, since they directly apply the results of Nelson and Tantawi to systems with more than two servers without justifying the generalization. In earlier work, we derive an analytic performance model for unsynchronized, block-interleaved, non-redundant disk arrays by approximating the queueing and fork-join overheads [23]. This work is the first to model such disk arrays as closed queueing systems with a continuum of request sizes from small to large requests. The work reported in this dissertation is a superset of the work reported in that paper.

To summarize, a definitive analytic model for block-interleaved redundant disk arrays does not exist. Basic advances in the analysis of fork-join queueing systems are needed before such a model can be derived. Such advances are not likely in the near future. Until they occur, we will have to live with many specialized approximate models for disk arrays. Aside from the work mentioned above, readers interested in modeling disk arrays may be interested in several queueing theoretic papers that deal specifically with fork-join queueing models [1, 2, 10, 11, 14, 15, 18].

2.5.2 Related Analysis Work

Work on the analysis of disk arrays typically focus on measurement or simulation. Measurement studies usually attempt to characterize the overall performance of a system or to answer specific performance question about the system. Simulation studies usually examine the effect of specific design or configuration tradeoffs in canonical disk arrays. Simulation studies usually address one of three main topics: data striping, redundancy schemes, or operation during the reconstruction of failed disks.

Since a large part of this dissertation is concerned with the analysis of RAID-II, our second disk array prototype, other measurement-based studies are the most relevant for the purposes of comparison with our work. Chen *et al.* [5] compares the throughput of RAID level 1 disk arrays versus RAID level 5 disk arrays using Amdahl mainframes. The primary purpose of the study is to validate the maximum throughput comparisons presented in a paper by Patterson, Gibson and Katz [32]. Drapeau [9], also know as Chervenak, presents low-level performance measurements of RAID-I, a disk array prototype constructed using off-the-shelf components. The paper is primarily concerned with the efficiency of the SCSI busses used in the system. Chen, Lee, et al. [6] describe the design, implementation and performance of RAID-II. The paper presents system-level and component-level performance measurements and provides justifications for major design decisions. Our work differs from the above related work in that we use a systematic framework, utilization profiles, to both characterize overall system performance and to answer specific performance questions about RAID-II, instead of ad-hoc techniques and measurements.

Data striping is the most studied topic in disk arrays. Livny [24] and Gray, Horst and Walker [13] study the benefits of striping versus not striping in non-redundant and redundant disk arrays, respectively. Livny shows that striping is generally preferable except at high loads. Gray, Horst and Walker show that it is sometimes desirable to distributed the redundant information across many disks even when the data is not striped. Reddy and Banerjee [34] investigate the performance of bit-level versus block-level data striping in non-redundant disk arrays. Chen and Patterson [7] and Lee and Katz [21] extend the work by Reddy and Banerjee by studying performance over a continuum of data striping sizes. Chen and Patterson present empirical rules of thumb for picking the data striping size. Lee and Katz derive analytic equations for the best data striping size.

Redundancy schemes concern the pattern in which redundant information is distributed across the disks in a disk array, the method in which the redundant information is computed, and the technique used to update the redundant information when data is written to the disks. This is currently a very active area of disk array research. Lee and Katz [22] investigate the performance consequences of different methods of placing parity in RAID level 5 disk arrays and proposes several properties that are generally desirable of parity placements. Bhide and Dias [3] and Stodolsky and Gibson [37] propose using logs to transform random parity updates into sequential disk accesses, significantly reducing the parity update overhead for small writes in RAID level 5 disk arrays. Menon, Roche and Kasson [25] propose an alternative scheme, called *floating parity*, for reducing small write overheads. Floating parity significantly reduces the rotational latency for parity updates by opportunistically writing the new parity to the rotationally nearest sectors.

Because most disk arrays will operate with a failed disk for only brief periods of time, performance in the face of disk failures has not been given serious attention until recently. The commercialization of disk arrays for continuous, real-time applications such as video service is changing the trend. In many applications, it is simply not acceptable for I/O performance to degrade sharply without warning. Menon and Mattson [27, 28] and Reddy and Banerjee [35] collectively propose three different ways, of "sparing" redundant disk arrays: *dedicated sparing*, *parity sparing*, and *distributed sparing*. In redundant disk arrays, it is desirable to have a spare disk that can be used immediately to replace a failed disk. More complex sparing schemes, such as parity sparing and distributed sparing, allow the spare disk to be used during the normal operation of the disk array by distributing the spare disk storage across all disks in the array. In addition to improved performance during the normal operation of the system, the distribution of the spare storage results in more uniform load distribution when a disk fails. Muntz and Lui [30] propose a method of organizing parity similar to RAID level 5 disk arrays that uniformly distributes the load during reconstruction. In contrast, a disk failure in a standard RAID level 5 disk array dramatically increases the load for the small group of disks in the same parity group as the failed disk due to the additional I/O needed to reconstruct the failed disk and to service I/O requests issued to the failed disk. Hou, Menon and Patt [16] investigate the effects on the response time of user requests when using different sized units of data recovery during the rebuilding of a failed disk.

2.5.3 Summary

Analytic queueing models of disk arrays are difficult to analyze because of queuing and fork-join synchronization. Basic advances in the analysis of fork-join queueing systems are needed before general analytic models of disk arrays are feasible. Until they occur, we will have to live with many specialized approximate models for disk arrays. In this dissertation, we present a specialized analytic model for disk arrays. It differs from other models in that it models the disk array as a closed rather than open queueing system, and handles a continuum of request sizes from small to large requests.

The analysis of disk arrays has so far focussed mainly on simulation studies of canonical disk arrays rather than the measurement or real disk arrays. In either case, the analysis studies rely on ad-hoc techniques. The lack of systematic analysis procedures is characteristic of computer science in general and especially noticeable in the analysis of disk arrays. In this dissertation, we present a new analysis technique based on *utilization profiles* and apply the technique to systematically analyze RAID-II, our second disk array prototype.

Chapter 3

Experimental Methodology

The previous chapter has provided an overview of RAID and a survey of the related work on RAID. This chapter introduces the experimental tools and techniques that were used in conducting the work presented in this dissertation. In particular, we describe the RAID simulator used to validate our analytic model; the RAID performance measurement tool used to measure the performance of RAID-II, our second disk array prototype; RAID-II itself; and the performance evaluation process.

3.1 The RAID Simulator

The RAID Simulator, *raidSim*, is an event-driven simulator developed by us at Berkeley for modeling both non-redundant and redundant disk arrays. The simulator models only disks; in particular, it does not model the host CPU, host disk controllers, or I/O busses. This allows us to study idealized disk arrays without being concerned with implementation details. *RaidSim* is used in Chapter 4 to validate our analytic model of non-redundant disk arrays.

Two questions basic to any event-driven simulation are:(1) when to start collecting simulation statistics, and (2) when to stop. When to start statistics collection is the simpler question to answer. In *regenerative simulation*, the process that is being modeled has well defined beginning and end points referred to as *regeneration points*. Thus, if you are interested in the behavior of the entire process from beginning to end, then statistics collection should begin immediately from the initial regeneration point.

Usually however, the processes we are interested in, such as the servicing of I/O requests by a disk array, are non-regenerative. It is a continuous process without well defined beginning or end points, and we are primarily interested in the long-term steady-state behavior of the system. In such systems, the initial behavior of the system is unrepresentative of the steady state behavior of the system because the process begins from an initial state that may never or rarely occur in the steady state. Thus, it is desirable to delay statistics collection until the system has "warmed up".

In our simulations, we use two different techniques to avoid the initial transient behavior. First, instead of issuing all the requests at the same time at the beginning of the simulation, we stagger their start. This starts the simulation in an initial state that is closer to the steady state. Second, we wait for all the initially issued requests to complete before collecting statistics. Other requests are issued as the initial requests complete; however, we do not start collecting statistics until the *initially* issued requests have been flushed from the system.

The question of when to end statistics collection is more difficult. With regenerative simulation, statistics collection should clearly end at a regeneration point. But even so, how many regeneration points should be simulated? The longer we simulate, the more accurate the results, but the more expensive the simulation.

Several simple stopping conditions are as follows: (1) simulate a constant number of regeneration points or requests, (2) simulate for a constant period of real time, or (3) simulate for a constant period of simulation time. Alas, all these alternatives are highly wasteful of resources. Because the variance in the collected statistics varies greatly as a function of the simulation parameters, low variance parameters are over-simulated to get accurate results for the few high variance parameters.

A more sophisticated stopping condition is to run a "presimulation" with a few regeneration points or requests before running the main simulation. The pre-simulation provides an estimate of the variance of individual regeneration points or requests and the estimated variance can be used to calculate the number of regeneration points or requests necessary to achieve the desired variance in the final average statistics. The main simulation stops when a constant number of regeneration points or requests, as determined by the pre-simulation, has been processed. This alternative, however, has a couple of unaesthetic qualities. First, one needs a mechanism to determine when the estimated variance is sufficiently accurate. Second, one cannot reuse the work performed by the pre-simulation during the actual simulation.

A general problem with any stopping condition dependent on variance estimation is correlation. If correlation is ignored, the estimated variance of average statistics can be much higher or lower than the actual variance of the average statistics. Regenerative simulation, by definition, does not suffer from correlation. In non-regenerative simulation, however, statistics taken close together in simulation time will tend to be positively correlated and hence result in estimated variances for averages that are much smaller than the actual variances. That is, the average statistics will vary much more than expected. When statistics are correlated in time, the correlation is referred to as *autocorrelation*.

In most non-regenerative simulation, the autocorrelation is highest for measurements taken close together in time and will rapidly decrease as they are taken farther apart. To make the problem more concrete, consider the following two sequences binary integers: $(0, 1, 1, 0, 1, 0, 0, 1)$ and $(0, 0, 1, 1, 1, 1, 0, 0)$. The first is generated by a set of uncorrelated binary random variables and the second is generated by a set of highly correlated binary random variables. Both sequences contain the same number of zeros and ones and, thus, the individual variances and the means are the same. In the second sequence, however, the zeros and ones tend to occur in groups of two. The second sequence might just as well have been represented by the following third sequence: $(0, 2, 2, 0)$. Thus, although the individual variances and the means of the two sequences are the same, the variance of the mean for the second sequence is significantly higher simply because the second sequence was determined with much less "freedom" than the first sequence.

In our simulations, we employ a modified version of the stopping condition based on variance estimation previously described. First, we batch together statistics from many requests when estimating the variance in order to reduce autocorrelation. Since most of the correlation occurs close together in time, the correlated statistics will be grouped together in batches and the batches themselves will be only slightly correlated. We examine autocorrelation coefficients for the batches to ensure that the batches are large enough to eliminate most of the correlation. Second, rather than use a pre-simulation to estimate variance, we iteratively generate variance estimates as the simulation proceeds, stopping when the desired variance for the average statistics is reached. To prevent premature termination of a simulation, a minimum of thirty requests must complete before checking the variance of the average statistics. Unlike the original stopping condition based on variance estimation, this stopping condition is dependent on the estimated variance. Thus, it will tend to produce variance estimates that are slightly smaller than the actual variance.

3.2 The RAID Performance Measurement Tool

The RAID Performance Measurement Tool, *raidPerf*, developed by us at Berkeley, is a generic tool for measuring the performance of concurrent I/O systems. Unlike most I/O devices today, a concurrent I/O system can service multiple requests simultaneously. Although the internal structures of *raidPerf* and *raidSim* are quite different, both use similar application-level interfaces. This makes it easy to run the same workloads on both the simulator and the real system. *RaidPerf* is used in Chapter 5 to gather performance measurements of RAID-II, our second disk array prototype. The measurements are subsequently used to analyze RAID-II using *utilization profiles*, a new empirical analysis technique presented in Chapter 5.

RaidPerf is primarily a workload generation tool with built-in statistic collection. As in *raidSim*, *raidPerf* models a closed system with a fixed number of independent processes continuously issuing I/O requests. Each process is created via the UNIX `fork()` system call.

The processes communicate with a central process that generates workloads and collects statistics via UNIX pipes. Using a single process that uses asynchronous I/O interfaces would be much more efficient than using multiple processes and pipes, but asynchronous I/O interfaces are not generally available on UNIX.

We have found that using pipes to communicate between multiple processes is highly CPU intensive. In the worst case, each communication operation between processes requires a context switch that takes on the order of a few milliseconds. To reduce the overhead, the central control process pipelines one request ahead, effectively creating an "on-deck" request for each process issuing I/O requests. This buffering of requests, however, creates greater measurement uncertainties for the central command process since more requests are outstanding at a time. The tradeoff is thus CPU overhead versus variations in the measured statistics.

Statistics collection during measurement also brings up the question of when to start and stop. Although many of the issues are the same as with simulation, the solutions are different due to the practical constraints of measurement. For example, in a simulation, getting the current simulation time is inexpensive since it is usually kept in a global variable. In a measurement run, however, getting the current real time usually requires an expensive system call. Furthermore, there is a wide disparity in the precision of the time available on different machines, with some machines keeping track of time in increments as large as 20 milliseconds!

Thus, stopping conditions based on the measurement of individual events is undesirable in measurements. A throughput based criterion is generally more desirable. In *raidPerf*, measurement stops when the measured throughput as calculated from the start of measurement to the current point in time varies by less than one percent for two consecutive *measurement intervals*. Each successive measurement interval contains twenty percent more requests than the previous interval.

3.3 The RAID-II Disk Array Prototype

This section describes RAID-II, our second disk array prototype, which will be analyzed in depth in Chapter 5. In the simplest view, RAID-II is a network disk controller that interfaces a 100 MB/s HIPPI (High-Performance Peripheral Interconnect) interconnect to a large number of 3.5" SCSI disks. In its overall operation, it is somewhat more complicated. RAID-II is a complete network file server that supports a custom network file system that efficiently support both large and small file requests from a wide variety of clients from supercomputers to desktop workstations. As of this writing, RAID-II is fully operational as a network disk controller and in experimental use as a network file server. It is RAID-II's former capabilities that interest us.

Figure 3.1 illustrates RAID-II's current network environment. RAID-II is connected via HIPPI to a network of HIPPI switches and an UltraNet. Only one of the two connections can be in operation at any given time. A HIPPI fiber extender connects a HIPPI switch ports to Lawrence Berkeley Labs where, in the near future, we will have access to both high-performance clients and peripherals that can effectively use RAID-II. The UltraNet currently spans two buildings at UC Berkeley and interconnects three Sun 4 workstations via 250 Mb/s Ultra links. In addition to the two high-performance networks,

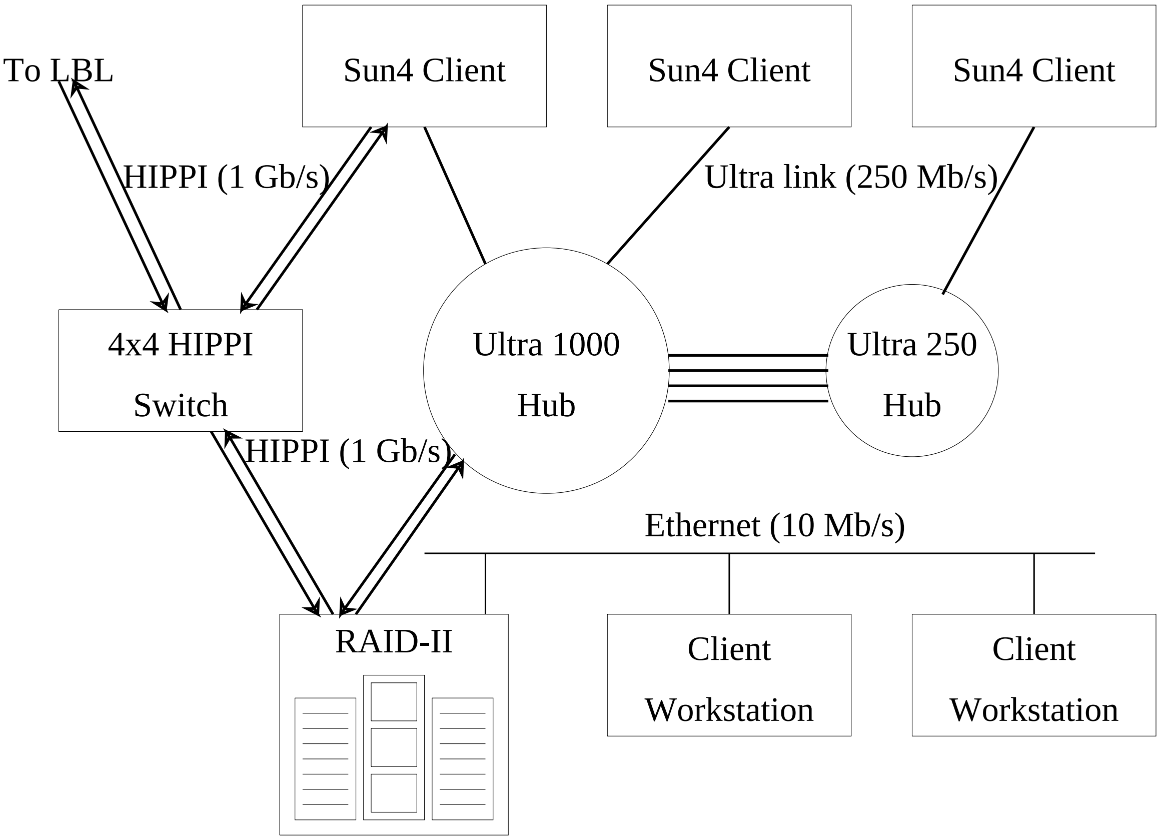


Figure 3.1: RAID-II Network Environment.

RAID-II is connected to an Ethernet to support client workstations without access to a high-performance network.

Figure 3.2 and Figure 3.3 illustrate RAID-II, which consists of three racks: two side racks each containing 72 SCSI disks and a center rack containing most of the electronic components of the system. The center rack contains most of the electronic components of the system in three chassis. The top chassis contains up to seven independent VME backplanes of which only four are currently in use. Each VME backplane can support up to two off-the-shelf VME disk controllers. The middle chassis contains the HIPPI interfaces

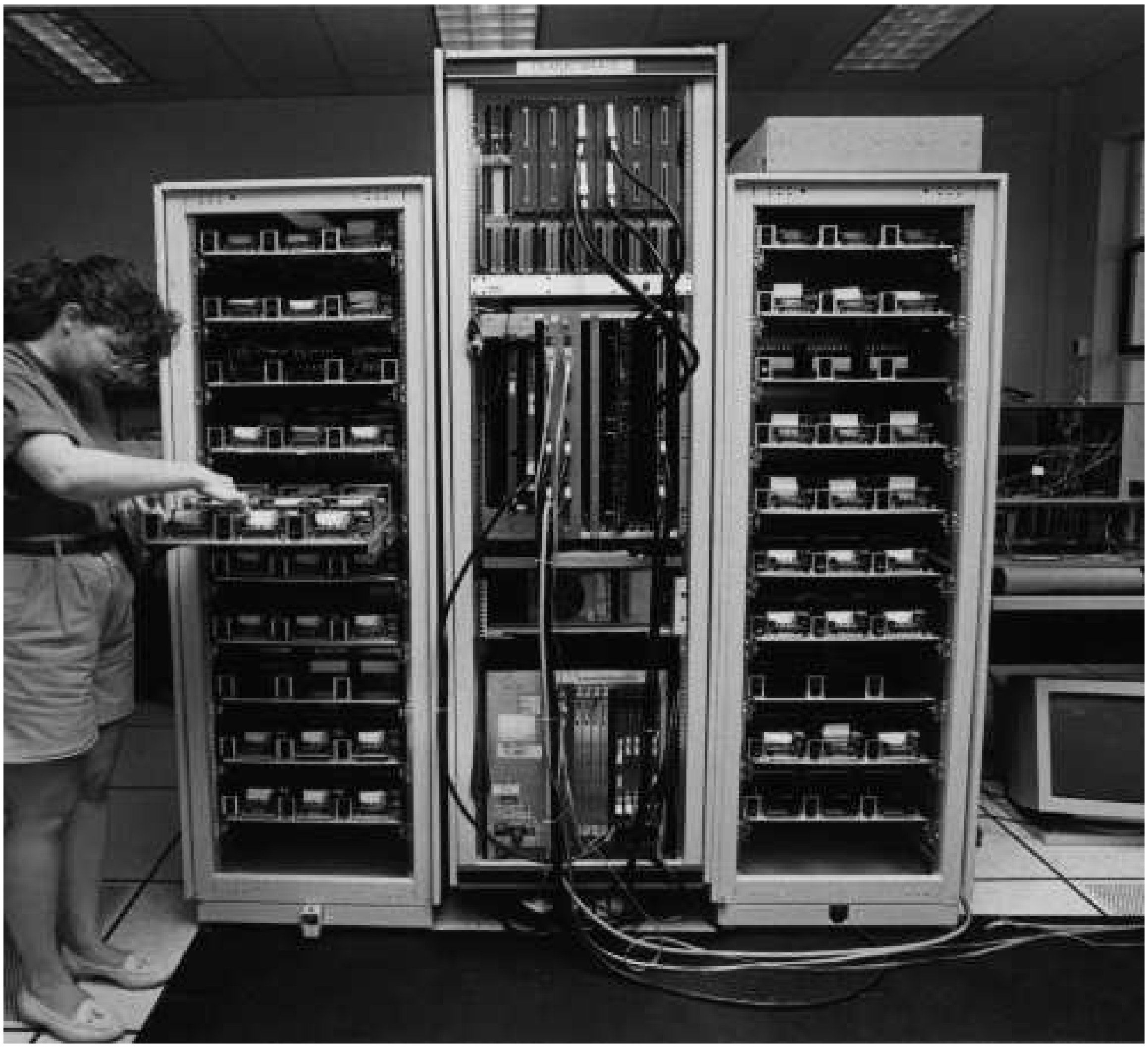


Figure 3.2: Photograph of the RAID-II Storage Server.

and a custom designed board called the *X-Bus board* which acts as the central memory and data interconnect for the system. The bottom chassis contains a Sun 4 workstation that controls all the previously mentioned components of the system via off-the-shelf VME-to-VME links. Each of the VME backplanes in the top shelf are connected via independent VME links to dedicated ports on the Xbus board. This allows data to be transferred directly

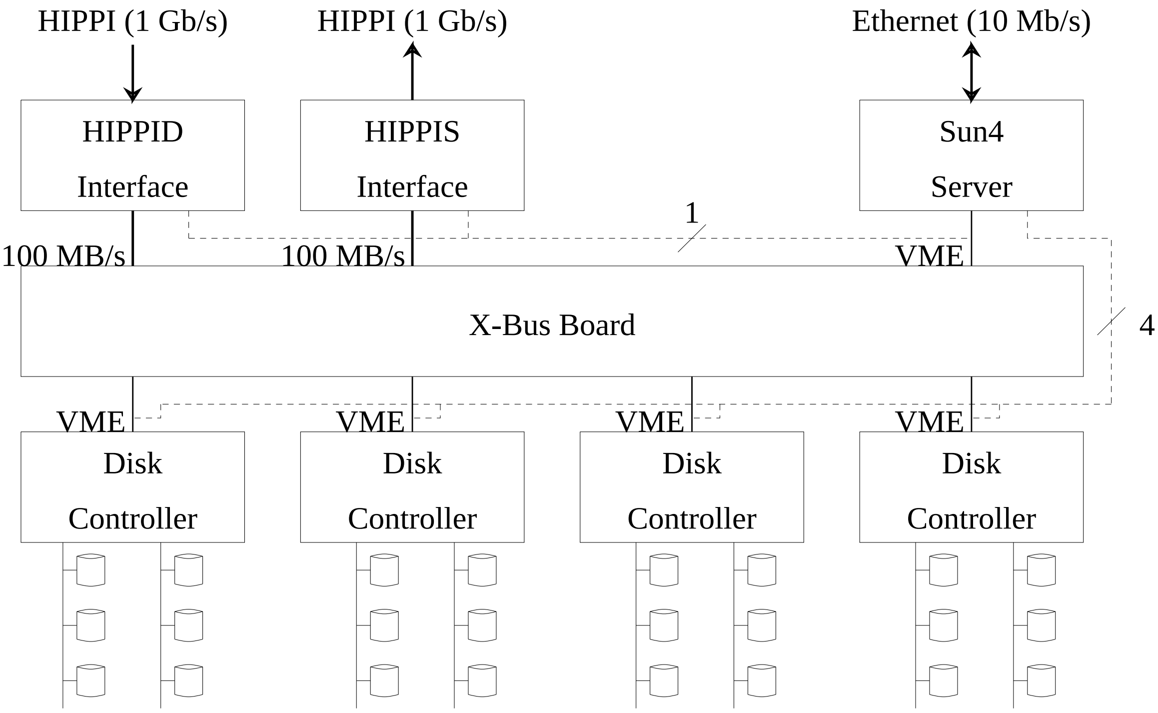


Figure 3.3: Schematic View of the RAID-II Storage Server. Solid lines indicate data paths and dotted lines indicate VME control paths.

between the disks and the network without having to go through the low-bandwidth memory system of the Sun 4 workstation. The X-Bus board serves as the primary data interconnect in the system. Each of the two unidirectional HIPPI source and destination interfaces are connected via 100 MB/s data busses to the X-Bus board. The rest of the five visible X-Bus ports consist of VME ports, four of which are used to connect to the VME disk controllers and one of which is used by the Sun 4 workstation to access network headers and file system metadata.

As illustrated by Figure 3.4, the memory system of the X-Bus board consists of 32 MB of DRAM organized into four independent memory banks with a sixteen-word interleave. By using the page-mode accesses supported by the DRAM, each memory bank

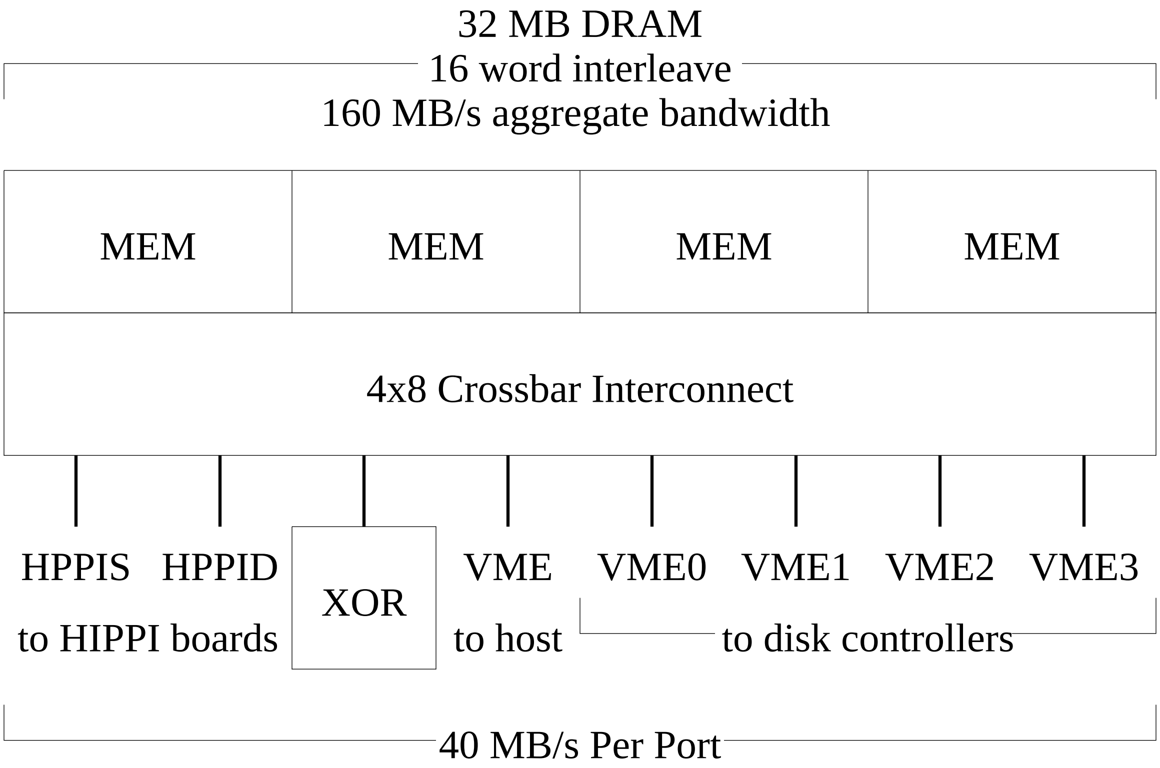


Figure 3.4: X-Bus Board.

can supply a 32-bit word every 80 ns, resulting in a peak data transfer rate of 50 MB/s per memory bank. Due to arbitration, addressing and memory latency overheads, each memory bank can sustain up to 40 MB/s on reads and 44 MB/s on writes.

The memory banks are connected via a four-by-eight crossbar interconnect to eight independent X-bus ports. Each X-Bus port can sustain up to 40 MB/s on reads and 44 MB/s on writes up to the aggregate bandwidth of the four memory banks. The X-Bus interconnects the X-Bus ports only to the memory banks, not to other X-Bus ports. All data transfers must go through the memory banks. Of the eight X-Bus ports, we have already described all of them except for the XOR/DMA port. The XOR/DMA port is the only port that is completely internal to the X-Bus board. It is used by the XOR/DMA engine to generate parity in support of RAID storage systems and also to perform block data transfers within the X-Bus board.

On a typical read operation, data is transferred from the disks via the VME disk controllers to the X-Bus memory and out over the HIPPI source interface. On a write operation, data is transferred from the network via the HIPPI destination interface to the X-Bus memory, the XOR/DMA computes the parity, and the parity and data are written to the disks via the VME disk controllers. To properly orchestrate the data transfers, network headers and other control information is transferred back and forth between the X-bus memory and the Sun 4 server. The Sun 4 server also controls the HIPPI interfaces and VME disk controllers by directly reading and writing control words over the VME links.

Figure 3.5 summarizes the system-level performance of the RAID-II storage server when accessed as a raw disk device over the HIPPI interconnect. Up to approximately 20 MB/s can be sustained for both large reads and large writes. The performance for writes is somewhat less than the performance for reads because parity information must be updated during writes and the X-Bus board's VME interfaces are slightly more efficient for disk reads (writes to memory) than disk writes (reads from memory). The bottlenecks in both cases turn out to be the Interphase Cougar disk controllers and our implementation of the X-Bus board's VME interfaces. Since we can easily add four more disk controllers, the ultimate bottleneck is our implementation of the VME interfaces. These can support only 7 MB/s each because we implemented them as less efficient synchronous rather than asynchronous interfaces. Fortunately, they are implemented as daughtercards on the X-Bus board and can easily be replaced. With an asynchronous VME interface, each VME port

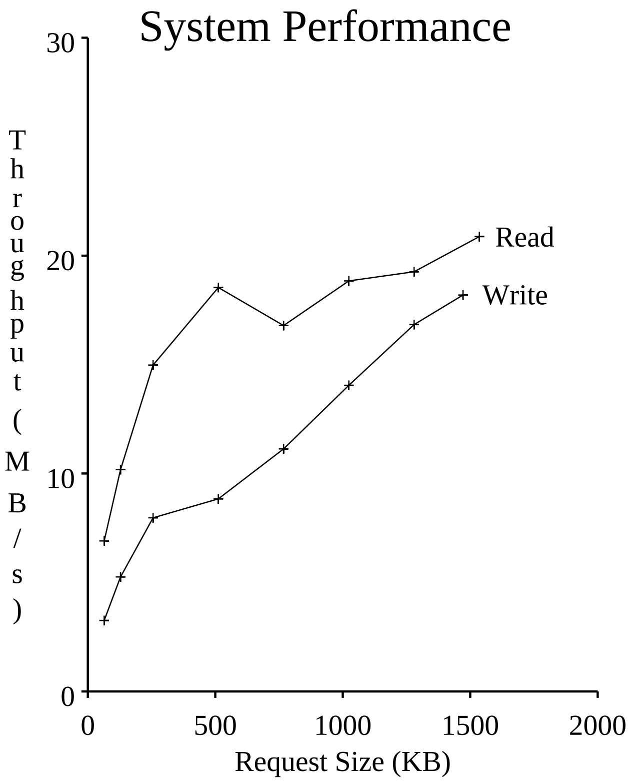


Figure 3.5: System Level Performance. Performance over HIPPI to/from disks. RAID-II is configured with 4 Interphase Cougar disk controllers, 8 SCSI strings and 24 SCSI disks in a RAID level 5 configuration. Due to the current lack of a client machine that can sustain the bandwidth provided by RAID-II, all HIPPI transfers are performed using the loopback mode of the HIPPI interfaces. Thus, data sent out from the X-Bus board over the HIPPI source interface is always looped back into X-Bus board via the HIPPI destination interface. This stresses RAID-II more than it would if an actual client were sourcing and sinking the data.

should be able to support up to 20 MB/s, and RAID-II should be able to sustain close to 40 MB/s in both reads and writes.

3.4 The Performance Evaluation Process

The previous sections have described the tools at our disposal for performing analytic, simulation and measurement studies. This section describes the performance evaluation *process* in which the tools are used. Much of the process described here is used both in Chapter 4 to validate our analytic model and in Chapter 5 to analyze the performance of RAID-II.

3.4.1 Introduction

We identify four main steps in the performance evaluation process:

1. Definition of goals.
2. Experimental design.
3. Analysis of *data*.
4. Presentation of *results*.

In most studies, backtracking will frequently occur as new results suggest different things to try. Because each step is dependent on the previous step, it is particularly important to avoid mistakes in the earlier steps.

Experimental design refers to the process of identifying the important system and workload parameters, picking the most significant values for each parameter, and the combination of parameter values to study. Frequently, investigators select a too narrow range of parameters or design studies in which only one parameter is varied at a time. Such studies lead to misleading results that not generalizable. On the other hand, selecting a too large range of parameters lead to results that are "generally" applicable but does not apply well to the most interesting parameter ranges. One can think of experimental design as analogous to the selection of questions an investigator may ask to solve a mystery. Just as the answer to each question reveals an aspect of the mystery, the performance of the system at each parameter combination reveals an aspect of the system under study. Clearly, without the right questions, you cannot get the right answers.

Once the experimental design is complete and the required data has been collected, we need to analyze the data and present the results. Unfortunately, investigators frequently do not distinguish between the analysis of data, and the presentation of results. The consequence are papers that are full of graphs and tables illustrating simulation or measurement output but lacking in significant conclusions. The analysis of data involves the formulation of analytic models, and the application of statistical techniques to summarize measured data. The presentation of results, on the other hand, communicates the conclusions derived from the analysis in a form that is both convincing and easily understood by the intended readers. In particular, the presentation of results should leave the reader with the following three things:

- An understanding of the results.
- The conditions under which the results apply.
- An understanding of the strength of the results.

We do not consider describing the process used to derive the results as absolutely necessary if the above three goals can be achieved without it. Usually, however, it will be impossible to give the reader an understanding of the strength of the results without describing the process that was used.

3.4.2 Experimental Design

Central to any experimental study, is the determination of which variables are important, what values of the variables are significant and what combination of variable values should be used. This process is referred to as *experimental design*, and is equally applicable to both simulation and measurement studies. In both cases, we are interested in studying the effect of certain parameter values on the performance of the system. We use experimental design extensively in both Chapter 4 and Chapter 5 to determine the parameters for our simulation and measurement experiments, respectively.

We will find the following definitions useful in describing the experimental design process:

Experiment	A single measurement or simulation.
Response	The result of an experiment.
Factor	An input variable that can influence the response.
Factor level	A possible value for a factor. For example, the factor *request size* may have a factor level of 4 KB, 8 KB, or 16 KB.
Replication	Repetition of an experiment at the same factor levels.
Design	An experimental design is specified by the list of factors, factor levels, combination of factor levels used for each experiment, and the number of replications.
Design point	The factor levels for a single experiment. For example, (request type = read, request size = 4 KB).
Design set	The set of design points used in an experimental design.

Replication of experiments allows the quantification of experimental, or "random," errors. In simulations, replication requires repeating the simulation at the same factor levels with a different random seed. Replication is frequently ignored in experimental studies in computer science, but some way to gauge experimental errors is always necessary. Without such a gauge, it is impossible to tell if the data is strong enough to support the results of the study.

Given the factors and factor levels for an experimental study, there are several common ways to design experiments. A *full factorial design* uses a complete cross product of all possible factor levels. This has the benefit that the design set is *balanced*, that is, it is easier to study the effect of a factor independently of the other factors. It also has the advantage that all interactions between any combination of factors can be studied. The only drawback is the large number of experiments required.

Other techniques are sometimes preferred over full factorial designs because they require many fewer experiments [17]. Such designs assume that the effect from higher order interactions between two or three factors is usually much smaller than first or second order effects. They construct design sets in which higher order effects cannot be distinguished from lower order effects.

In this dissertation, we always use full factorial designs because the number of factors and factor levels we are interested in is small enough and we have sufficient resources to perform the necessary experiments. The alternative design techniques are more frequently used in fields such as biology or chemistry where large amounts of time and resources are often required for each experiment.

3.4.3 Linear Regression

In Chapter 5, we will frequently fit equations to experimental data to formulate models for the measured system, study the effects of parameters, and to summarize the experimental data. Given enough CPU power, it is feasible to fit arbitrary equations using arbitrary criteria; however, it is far more efficient to fit linear equations while minimizing the sum of squared errors. This process is called *linear regression*, and is the main form of curve fitting used in this dissertation.

Linear regression takes as input the data to be fitted and a list of *basis functions.* It outputs the *best-fit coefficients*—the coefficients for the basis functions that minimizes the sum of squared errors. For example, given the basis functions $(1, x_1, x_2, x_1x_2)$, linear regression will calculate coefficients β_0 through β_3 such that the equation $y(x_1, x_2) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2$ minimizes the sum of squared errors. Here, β_0 is the *base response* of the system—the system response when x_1 and x_2 are both zero. β_1 is the *first order effect* of x_1; when x_2 is equal to zero, a unit change in x_1 will change the response on average by $\beta 1$. Likewise, β_2 is the first order effect of x_2. β_3 is the *interaction effect* of x_1 and x_2; a non-zero β_3 indicates that a change in the response due to a change in x_1 or x_2 depends on the current value of x_2 or x_1, respectively.

Aside from the best-fit coefficients, most linear regression packages also output statistics describing the significance of each coefficient, the correlation between each pair of coefficients, and the function's goodness-of-fit. This allows us to eliminate insignificant or redundant factors and to gauge the importance of each factor to the accuracy of the linear regression. As the preceding sentence illustrates, it is important to distinguish between *statistical significance* and *practical significance.* A factor is statistically significance if it has a distinctly measurable effect; it is practically significant only if the effect is sufficiently large. For example, using suntan lotion before jumping into a volcano may have a statistically significant effect, but for all practical purposes, the effect is insignificant.

The value of each coefficient divided by its standard deviation is referred to as a *t-statistic* and provides a measure the coefficient's statistical significance. A small t-statistic results when the standard deviation is large relative to the estimated value of the coefficient, and indicates that the coefficient is statistically insignificant. Practically speaking, you may want to eliminate the corresponding basis function and retry the linear regression. If the errors in the regression are independently distributed as a normal variable with a common mean, the t-statistics can be used to formulate confidence intervals for the best-fit coefficients. Most errors in modeling and analyzing computer systems, however, do not meet these requirements.

The *correlation matrix* tabulates the correlation coefficient between each pair of best-fit coefficients. High correlation between a pair of coefficients indicates that it may be possible to remove a correlated factor without significantly affecting the accuracy of the linear regression. Although the t-statistic and the correlation matrix for best-fit coefficients can be used to iteratively eliminate unimportant or redundant factors, the only sure way to get the best regression with the fewest factors is to try all subsets of factors as a basis.

So far, we have discussed how to tell if the best-fit coefficients are statistically significant and some techniques for eliminating unimportant or redundant basis functions. We have yet to discuss, however, a method for characterizing the errors in a regression to determine if the regression is sufficiently accurate. Although a statistically insignificant result is useless, just because a result is statistically significant does not necessarily mean that it is practically significant or useful. Methods and statistics for characterizing such errors will be discussed in the next section.

3.4.4 Error Metrics

Performance evaluation often requires comparing two or more models and characterizing the differences between them. Usually, one model is considered less accurate than the other, such as when an analytic model is compared against a simulation model, or when a simulation model is compared to measurements of the modeled system. In such cases, the differences can be thought of as *errors* in the less accurate model. Because such comparisons frequently involve many data points, aggregate statistics and graphs must be used to summarize the errors in a form that is more easily interpreted. The following defines statistical terms and metrics that will be used in Chapter 4 and Chapter 5 to characterize errors in modeling and analyzing disk arrays, respectively:

y_i	The response for the ith design point.
$\hat{y}_i$	The predicted response for the ith design point.
e_i	$y_i - \hat{y}_i$ (Error for the ith design point.)
$\overline{y}$	Arithmetic mean of y_i's.
SSE	$\sum e_i^2$ (sum of squared errors)
SST	$\sum (y_i - \overline{y})^2$ (sum of squares total)
R^2	$(SST - SSE)/SST$ (coefficient of determination)
max error	$\max e_i$ (maximum error)
90 % error	The 90th percentile of e_i.

The coefficient of determination, R^2, is a measure of how closely the predicted response matches the actual response. A baseline prediction predicting all responses equal to the mean, $\overline{y}$, would have $R^2 = 0$, while a perfect predictor would have $R^2 = 1$. R^2 can be interpreted as the fraction of "total variation" that is accurately predicted. Although R^2 is a good measure of overall accuracy, it may be highly dependent on the choice of the design set and may vary significantly for certain subsets of the design set. That is, a predictor that is wildly inaccurate for a significant collection of design points can still have $R^2 \simeq 1$ if the predictor is accurate for the other design points. Because R^2 can be insensitive to large errors in the predictor, the *max error* metric is useful as a measure of the worst-case prediction. The *90% error* metric is a compromise between R^2 and *max error* which compensates for the extreme sensitivity of *max error* to outliers. We will frequently use the coefficient of determination, *max error*, and *90% error* in characterizing the error of analytic models and linear regressions.

In addition to the above error metrics, we will frequently use graphs to illustrate errors. During analysis, it is useful to graph each predicted response along with its actual response; however, this usually results in either a large number of graphs of a very cluttered graph. Thus, we will frequently graph the actual response in aggregate form as *percentile intervals* containing the indicated percentage of the response. We specifically use percentile intervals rather than confidence intervals because standard methods for calculating confidence intervals require that errors be independently distributed as a normal variable with a common standard deviation. If the errors are independent, certain transformations can be made on responses to satisfy the requirements but these transformations are often artificial. Furthermore, errors resulting from modeling errors are almost never independent. Percentile intervals, on the other hand, do not require any assumptions in the distribution of errors or artificial transformations of the response. They do, however, require a larger number of data points.

3.4.5 Summary

Simulation and measurement are widely used in computer science but few researchers use systematic performance evaluation procedures. Experimental design, linear regression, and statistical error metrics are widely used in other experimental sciences, but have yet to gain acceptance in computer science. This makes it difficult to compare or duplicate the work of other researchers and sometimes makes it impossible to assess the validity of advanced research. In this section, we have described several basic tools and techniques that are used throughout this dissertation to guarantee the accuracy and validity of our results. In particular, the RAID simulator is used in Chapter 4 to validate our analytic model and the results derived from the model. The RAID performance measurement tool is used in Chapter 5 to analyze the performance of RAID-II. The performance evaluation process is used throughout the dissertation in designing experiments and interpreting the results of both simulation and measurement.

Chapter 4

The Analytic Model

In this chapter, we derive and validate via simulation an analytic performance model for block-interleaved, non-redundant disk arrays. Our approach is to derive the expected utilization of a given disk in the disk array. Because we are modeling a closed system where each disk plays a symmetric role, knowing the expected utilization of a given disk will allow us to compute the entire system's throughput and response time. We examine via simulation the accuracy of the analytic model and the error introduced by each approximation in the derivation of the analytic model. We then extend the basic analytic model to handle variable sized requests and validate the result. Finally, we apply the model to derive an equation for the optimal size of data striping in disk arrays.

For the sake of convenience and clarity, we present the complete derivation of the analytic model before presenting its empirical validation; the actual development of the model followed an iterative process, alternating between analytical derivations and empirical validations.

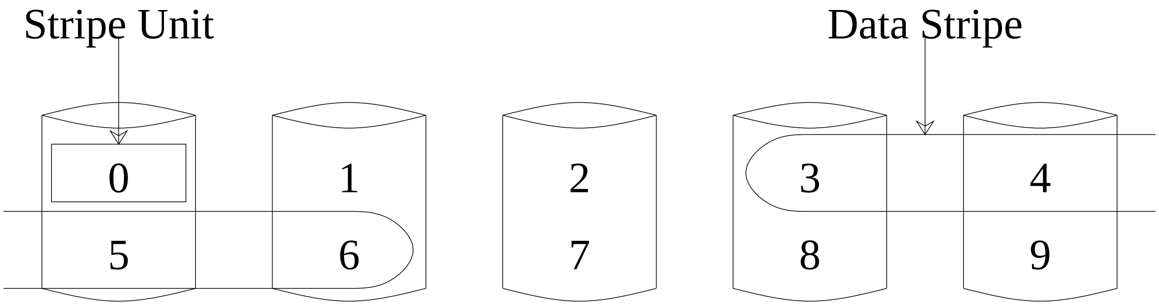


Figure 4.1: Data Striping in Disk Arrays.

Stripe unit is the unit of data interleaving, that is, the amount of data that is placed on a disk before data is placed on the next disk. Stripe units typically range from a sector to a track in size (512 bytes to 64 kilobytes). The figure illustrates a disk array with five disks with the first ten stripe units labeled.

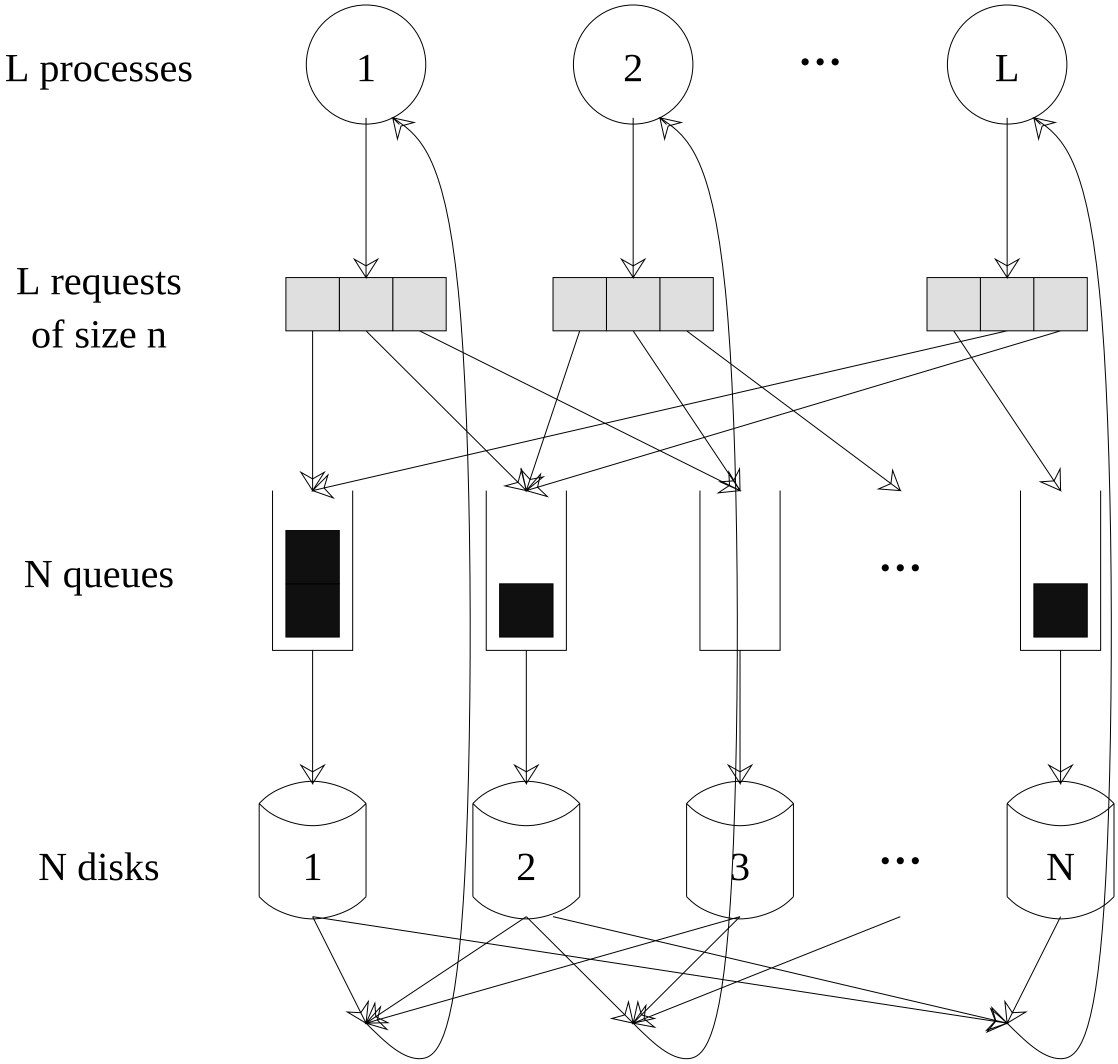
Data stripe is a sequence of logically consecutive stripe units. A logical I/O request to a disk array corresponds to a data stripe. The figure illustrates a data stripe consisting of four stripe units spanning stripe units three through six.

4.1 The Modeled System

Our primary focus is on modeling non-redundant asynchronous disk arrays. Figure 4.1 illustrates the basic disk array of interest and the terms *stripe unit* and *data stripe*. For readers who are familiar with the RAID taxonomy, we mention that the analytic model we develop can also be used to model reads for RAID level 5 disk arrays using the left-symmetric parity placement [22]; the left-symmetric parity placement does not disturb the *data* mapping illustrated in Figure 4.1.

4.2 The Model System

Consider the *closed* queueing system illustrated in Figure 4.2. The system consists of L processes, each of which issues, one at a time, an *array request* of size n stripe units.



$L \equiv$ Number of processes issuing requests.
$N \equiv$ Number of disks.
$n \equiv$ Request size in disks.
$n \leq N$.
$S \equiv$ Service time of a given disk request.

Figure 4.2: Closed Queuing Model for Disk Arrays.

Each array request is broken up into n *disk requests* and the disk requests are queued round-robin starting from a randomly chosen disk. Each disk services a single disk request at a time in a FIFO manner. When all the disk requests corresponding to an array request are serviced, the issuing process generates another array request, repeating the cycle. Two or more array requests may partially overlap on some of the disks, resulting in complex interactions. We sometimes refer to array requests simply as *requests.* In the derivation of the analytic model, we will assume that L and n are fixed. We will also assume that the processes do nothing but issue I/O requests. Later, we will extend the model to allow variable sized requests.

4.3 The Expected Utilization

In deriving the expected utilization of the model system, we will find the following definitions useful:

U	$\equiv$	Expected utilization of a given disk.
R	$\equiv$	Response time of a given array request.
W	$\equiv$	Disk idle (wait) time between disk request servicings.
Q	$\equiv$	Queue length at a given disk when a request arrives.
p_0	$\equiv$	Probability that the queue at a given disk is empty when the disk finishes servicing a disk request.
p	$\equiv$	n/N, the probability that a request will access a given disk;

If we visualize the activity at a given disk as an alternating sequence of busy periods of length S and idle periods of length W, the expected utilization of a given disk is,

$$U = \frac{E(S)}{E(S) + E(W)}. \tag{4.1}$$

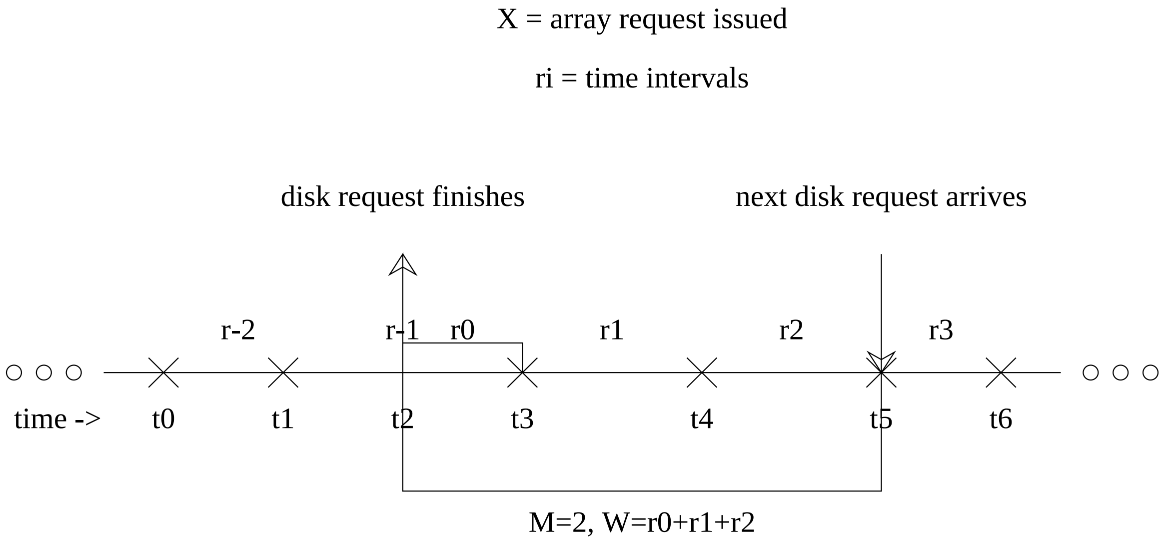


Figure 4.3: Time-line of Events at a Given Disk. After the disk request finishes service at time t_2, $M = 2$ array requests that do not access the given disk are issued at times t_3 and t_4 before an array request that accesses the given disk is issued at time t_5. The disk remains idle for a time period of $W = r_0 + r_1 + r_2$.

Idle periods of length zero can occur and imply that another disk request is already waiting for service, $Q > 0$, when the current disk request finishes service.

Let r_0 denote the time between the end of service of a given disk request and the issuing of a new array request into the system. Let $r_i, i \in \{1, 2, \ldots\}$, denote the time intervals between successive issues of array requests numbered relative to r_0. Let M denote the number of array requests issued after a given disk finishes a disk request until, but excluding, the array request that accesses the given disk. Since each array request has probability p of accessing a given disk, M is a modified geometric random variable with $E(M) = 1/p - 1$. Figure 4.3 illustrates the above terms.

By conditioning on the queue length at the time a disk request finishes service, we have

$$\begin{aligned} E(W) &= P(Q>0)E(W|Q>0) + P(Q=0)E(W|Q=0), \\ E(W) &= (1-p_0)0 + p_0 E(\textstyle\sum_{i=0}^{M} r_i), \\ E(W) &= p_0(E(r_0) + E(\textstyle\sum_{i=1}^{M} r_i)). \end{aligned}$$

Substituting into Equation 4.1 we have

$$U = \frac{E(S)}{E(S) + p_0(E(r_0) + E(\sum_{i=1}^{M} r_i))} \tag{4.2}$$

Equation 4.2 is an exact, though not directly useful, equation for the expected utilization of the model system.

4.4 Approximating the Expected Utilization

In the previous section, we formulated Equation 4.2, an exact equation for the expected utilization of the model system. Unfortunately, the exact equation consists of terms that are difficult, if not impossible, to compute. In this section, we approximate the components of Equation 4.2 to make it analytically tractable.

To simplify Equation 4.2, the first approximation we make is

$$E(\sum_{i=1}^{M} r_i) \simeq E(M)E(r_i) = (1/p - 1)E(R)/L.$$

From Little's Law, we know that the average time between successive issues of array requests is $E(R)/L$. Note also that M is a *stopping time*, that is, the event that the mth request misses the given disk, $M > m$, is independent of the random variables $r_{m+1}, r_{m+2}, \ldots$.

Thus, from Wald's Equation, the above approximation would be exact if $r_i, i \in \{1, 2, \ldots\}$ were independently distributed with a common mean of $E(R)/L$. For the moment, we will take the above approximation as given, but later show via simulation that the above is an extremely good approximation.

The second approximation we make is to assume that $E(r_0) \simeq 0$. As motivation, we present the following observations about $E(r_0)$:

- $E(r_0) = 0$ implies that disk requests associated with the same array request finish at the same time and thus an array request is issued immediately whenever *any* disk request finishes.
- $E(r_0) = 0$ when $n = 1$, that is, when each array request consists of a single disk request, the completion of each disk request corresponds to the completion of the corresponding array request and, thus, the process that issued the disk request will immediately issue another array request.
- $E(r_0) \simeq 0$ when $n = N$, that is, when an array request always uses all the disks, disk requests associated with the same array request will tend to finish at close to the same time because all the disks will be in similar states and operate in a lock step fashion since disk service times are deterministic and disk requests across disks will be almost identical.

The third and final approximation is $p_0 \simeq E(S)/E(R)$. This equation is true for $M/M/1$ systems and approximately holds at low to moderate loads for $M/G/1$ systems but the primary motivation for the approximation comes from empirical observations.

Incorporating the three approximations, we can rewrite Equation 4.2 as,

$$U \simeq \frac{1}{1 + \frac{1}{L}(1/p - 1)}. \tag{4.3}$$

Under the approximations we have made, *the expected utilization is insensitive to the disk service time distribution, S.*

Since this is a closed system, the expected response time can be directly calculated from the expected utilization:

$$E(R) = \frac{E(S)Ln}{UN}. \tag{4.4}$$

The expected throughput can be written as,

$$T = \frac{UNB}{E(S)}, \tag{4.5}$$

where B is the size of the stripe unit. Future references to a specific analytic model will refer to the above equations and to Equation 4.3 in particular.

4.5 Experimental Design

We are interested in the following factors to validate our analytic model.

diskModel	The type of disk.
N	Number of disks in the disk array.
L	The number of processes issuing I/O requests.
B	Size of the stripe unit (block size).
p	Request size as a fraction of the disk array.

The design set consists of the complete cross product of the following factor levels.

Factor	Factor Levels
diskModel	Lightning, Fujitsu, FutureDisk
N	2, 3, 4, 8, 16
L	1, 2, 4, 8, 16, 32
B	(1, 4, 16, 64) KB
p	$(1, 2, \ldots, N)/N$

The parameters for the three disk models listed above are describe in Table 4.1. The number of factor levels for p depend on N and are expressed as multiples of $1/N$. This means, for example, that there are twice as many design points with $N = 4$ as $N = 2$. Thus, although we do not intrinsically value disk arrays with $N = 4$ more than disk arrays with $N = 2$, the design set implicitly assigns the former twice the importance of the latter. To compensate for this effect, we weigh all design points by the factor $1/N$ when calculating statistics.

We simulate each design point in the design set twice with two different random seeds to quantify the experimental error, which intrinsically cannot be explained by any model. Thus, the total number of simulation runs is $2(3\times4\times6\times(2+3+4+8+16)) = 4,752$. In each simulation run, the disks are rotationally synchronized, requests to the disk array are aligned on stripe unit boundaries, and L and p are held constant.

4.6 Validation of the Analytic Model

In this section, we examine via simulation the accuracy of the analytic model and the error introduced by each of the three approximations in the model of Section 4.4. Recall that the three approximations are as follows:

1. $E(\sum_{i=1}^{M} r_i) \simeq (1/p - 1)E(R)/L$,

2. $E(r_0) \simeq 0$,

	Lightning	Fujitsu	FutureDisk
bytes per sector	512	512	512
sectors per track	48	88	132
tracks per cylinder	14	20	20
cylinders per disk	949	1944	2500
revolution time	13.9 ms	11.1 ms	9.1 ms
single cylinder seek time	2.0 ms	2.0 ms	1.8 ms
average seek time	12.6 ms	11.0 ms	10.0 ms
max stroke seek time	25.0 ms	22.0 ms	20.0 ms
sustained transfer rate	1.8 MB/s	4.1 MB/s	7.4 MB/s

Table 4.1: Disk Model Parameters. Average-seek-time is the average time needed to seek between two equally randomly selected cylinders. Sustained-transfer-rate is a function of bytes-per-sector, sectors-per-track and revolution-time. *Lightning* is the IBM 0661 3.5" 320 MB SCSI disk drive, *Fujitsu* is the Fujitsu M2652H/S 5.25" 1.8 GB SCSI disk drive and *FutureDisk* is a hypothetical disk of the future created by projecting the parameters of the Fujitsu disk three years into the future based on current trends in disk technology. The most dramatic improvements are in the bit and track density of the disks rather than in mechanical positioning times. Thus, disks in the future will have much higher sustained transfer rates but only marginally better positioning times. The seek profile for each disk is computed using the following formula:

$$seekTime(x) = \begin{cases} 0 & \text{if } x = 0 \\ a\sqrt{x-1} + b(x-1) + c & \text{if } x > 0 \end{cases}$$

where x is the seek distance in cylinders and a, b and c are chosen to satisfy the single-cylinder-seek-time, average-seek-time and max-stroke-seek-time constraints. The square root term in the above formula models the constant acceleration/deceleration period of the disk head and the linear term models the period after maximum disk head velocity is reached. If cylinders-per-disk is greater than approximately 200, a, b and c can be approximated using the following formulas:

$$\begin{aligned} a &= (-10\, minSeek + 15\, avgSeek - 5\, maxSeek)/(3\sqrt{numCyl}) \\ b &= (7\, minSeek - 15\, avgSeek + 8\, maxSeek)/(3\, numCyl) \\ c &= minSeek \end{aligned}$$

where *minSeek, avgSeek, maxSeek* and *numCyl* correspond to the disk parameters single-cylinder-seek-time, average-seek-time, max-stroke-seek-time and cylinders-per-disk, respectively. We have compared the model to the seek profile of the Amdahl 6380A published by Thisquen [38] and have found the model to closely approximate the seek profile of the actual disk. In practice, we have found the model to be well behaved, although care must be taken to check that a and b evaluate to positive numbers.

3. $p_0 \simeq E(S)/E(R)$.

Consider the following definitions:

$$\begin{aligned}
\hat{U} &\equiv \frac{1}{1+\frac{1}{L}(1/p-1)} \\
\hat{U}_1 &\equiv \frac{E(S)}{E(S)+p_0(E(r_0)+(1/p-1)E(R)/L)} \\
\hat{U}_2 &\equiv \frac{E(S)}{E(S)+p_0(\sum_{i=1}^{M} r_i)} \\
\hat{U}_3 &\equiv \frac{1}{1+\frac{1}{E(R)}(E(r_0)+\sum_{i=1}^{M} r_i)}
\end{aligned}$$

The variable $\hat{U}$ represents the analytic model derived in Section 4.4 by applying approximations 1, 2 and 3. The variables $\hat{U}_1$, $\hat{U}_2$ and $\hat{U}_3$ are submodels derived by applying only one of approximations 1, 2 or 3, respectively. By examining the errors in $\hat{U}_1$, $\hat{U}_2$ and $\hat{U}_3$, we can study the errors introduced by each of the corresponding approximations.

The table below tabulates the error metrics discussed in Section 3.4.4 for the *best* empirical model, $\hat{U}$, $\hat{U}_1$, $\hat{U}_2$ and $\hat{U}_3$. The best empirical model is the model that minimizes the sum of squared errors or, equivalently, maximizes R^2 and is computed by averaging the responses of data points with the same design point. The error metrics for the best empirical model are useful for comparison purposes and give a feel for the magnitude of experimental errors that cannot be explained by any model. The metric $1-R^2$ is included for easier comparisons of models that have values of R^2 close to one. Because we are more interested in relative rather than absolute errors, the error metrics are calculated for the logarithm of utilization rather than utilization directly. This means that for small errors—less than approximately 20%—the *max error* and *90 % error* metrics can be interpreted as percentage deviations. For example, a *max error* of 0.0430 represents a 4.30 % deviation from the *predicted* utilization.

Model	R^2	$1 - R^2$	*max error*	*90 % error*
BEST	0.9995	0.0005	0.0430	0.0133
$\hat{U}$	0.9814	0.0186	0.1863	0.0987
$\hat{U}_1$	0.9990	0.0010	0.0834	0.0192
$\hat{U}_2$	0.9888	0.0112	0.1676	0.0742
$\hat{U}_3$	0.9808	0.0192	0.1807	0.0951

The above table shows that 0.05 % of the variation in response is caused by experimental errors that cannot be explained by any model. The maximum experimental error is 4.30 %, and 90 % of all experimental errors are less than 1.33 %. The maximum error for the analytic model, $\hat{U}$, derived in Section 4.4 is 18.63 % with 90 % of errors less than 9.87 % of the predicted utilization. Finally, approximations 2 and 3 introduce most of the experimental errors, while approximation 1, as expected, introduces very few errors.

Figure 4.4 plots the response predicted by the analytic model $\hat{U}$ together with the 90 percentile intervals as a function of L and p, the only two factors considered important by the analytic model. In the figure, each 90 percentile interval encloses 90 % of the weighted simulated responses for the given values of L and p. We have elected to give percentile intervals rather than confidence intervals because standard methods for calculating confidence intervals require that errors be normally distributed with a constant standard deviation. Certain transformations can be made on responses to satisfy the requirement but these transformations are often artificial. Percentile intervals, on the other hand, do not require any assumptions in the distribution of errors but do require more data points to calculate. Figure 4.4 shows good correspondence between the analytically predicted utilization and the 90 percentile intervals determined by simulation. As we will see later, the

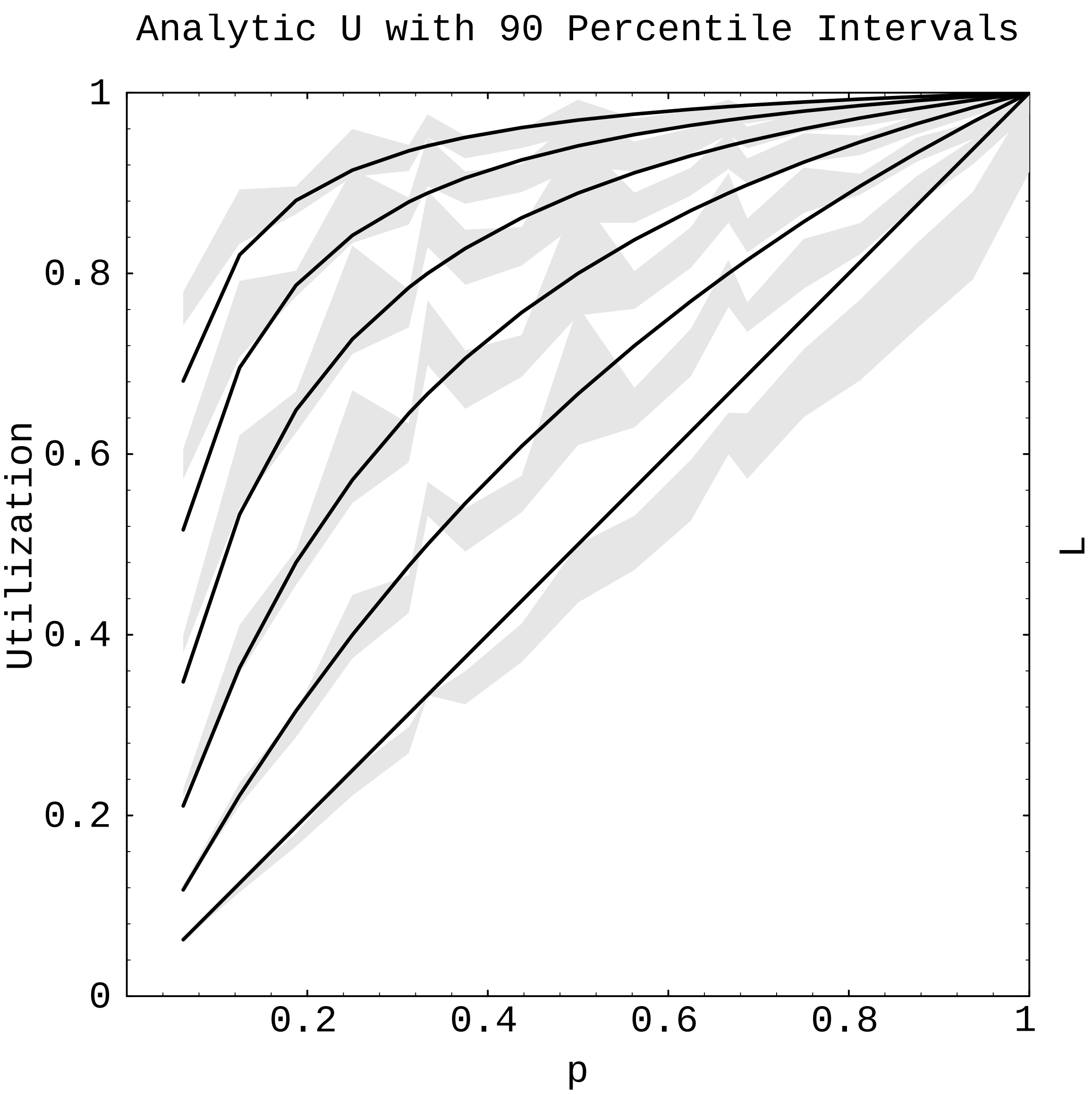


Figure 4.4: Analytic U with 90 Percentile Intervals as a Function of L and p. Each line and shaded region represents the analytically predicted response and empirically determined 90 percentile intervals, respectively, for a different value of L. From bottom to top the values for L are 1, 2, 4, 8, 16 and 32, respectively.

percentile intervals are jagged because the factor N, which is ignored by the analytic model, is significant in explaining variations in the response.

4.7 Validation of Model Properties

The previous section examined the errors in the analytic model, but regardless of the accuracy of the model, certain properties of the model may be valid where the model itself is not. This section will investigate the model's prediction that the utilization is independent of the disk service time distribution, S, or more specifically is dependent only on the factors L and p. We will also examine whether, as implied by the model, the utilization can be accurately modeled by an equation of the form $U = 1/(1 + \frac{1}{L}f(p, N))$ where $f(p, N)$ represents an arbitrary function of p and N. Recall that $f(p, N)$ is equal to $1/p - 1$ in the analytic model.

4.7.1 Significance of Factors

The following table tabulates the error metrics for the best empirical models—models that maximize R^2—when certain factors are excluded. The first entry labeled *NONE* corresponds to the best empirical model that can be constructed when no factors are excluded and is identical to the model identified as *BEST* in the previous section. If the exclusion of a factor results in error metrics that are only slightly different from the error metrics of the *NONE* entry, this provides strong evidence that the factor can be safely ignored by the analytic model, at least over the range of factor levels investigated. The last two entries in the table illustrate the effects of excluding more than one factor at a time. As before, the metrics are calculated for the logarithm of utilization rather than for utilization directly.

Deleted Factors	R^2	$1 - R^2$	*max error*	*90 % error*
NONE	0.9995	0.0005	0.0430	0.0133
diskModel	0.9990	0.0010	0.0570	0.0205
N	0.9936	0.0064	0.1653	0.0525
L	0.4238	0.5762	1.4311	0.4835
B	0.9986	0.0014	0.0789	0.0252
p	0.4285	0.5715	1.8733	0.5018
diskModel B	0.9983	0.0017	0.0843	0.0269
diskModel B N	0.9926	0.0074	0.1678	0.0577

As predicted by the analytic model, the above table illustrates that utilization is insensitive to the disk service time distribution, S, or more specifically to the two principle factors, *diskModel* and B, that determine S. Excluding both the factors *diskModel* and B result in error metrics *max error* = 8.43 % and *90 % error* = 2.69 % which compare favorably with the error metrics of the best case when no factors are excluded of *max error* = 4.30 % and *90 % error* = 1.33 %. Thus, we conclude that utilization is insensitive to the disk service time distribution, S.

The insensitivity of utilization to the disk service time distribution is hardly surprising given that we have a closed queueing system where processes do nothing but I/O. Consider the following thought experiment. If we replaced all the disks with devices twice as fast but the same in other respects, we would expect throughput to exactly double but utilization to remain the same. Real systems are more complicated because changing disks or stripe unit sizes not only changes the mean of the service time distribution but also its shape. However, this simple thought experiment shows why utilization is insensitive to the disk service time distribution. It also provides evidence that this property will hold under factor levels not investigated in this paper.

In contrast to excluding factors *diskModel* and B, excluding factor N introduces much larger errors. The errors, however, are not large enough to say that it is never acceptable to ignore N. Whether N can be ignored depends on the actual use of the model. If one is primarily interested in the effects of varying N, it may not be acceptable, but otherwise, it is probably acceptable. The table shows that the remaining factors, L and p, are clearly important and cannot be ignored.

4.7.2 Relationships Between Factors

The previous section identified the factors N, L and p as significant in formulating a model for the utilization of the modeled system. In this section, we empirically investigate the relationships between these factors. In particular, we examine whether, as implied by the model, the utilization can be accurately modeled by an equation of the form $U = 1/(1 + \frac{1}{L} f(p, N))$ where $f(p, N)$ represents an arbitrary function of p and N. We then search for values of $f(p, N)$ that result in accurate analytic models.

Figure 4.5 plots $\log(1/\overline{U} - 1)$ versus $\log_2 L$ for the different factor levels of N and p where $\overline{U}$ is the geometric mean of the utilization over all design points with the same values of N, L and p. We use $\overline{U}$ rather than arbitrarily selecting design points with a specific value for *diskModel* and B to reduce experimental error. If $U \simeq 1/(1 + \frac{1}{L} f(p, N))$, a plot of $\log(1/\overline{U} - 1)$ versus $\log_2 L$ should result in straight lines with a slope of -1. As is evident from the figure, this is approximately the case.

Now that we have verified that $U \simeq 1/(1 + \frac{1}{L} f(p, N))$, it remains to determine a good approximation for $f(p, N)$. Since f is a function only of p and N and not L, we can theoretically determine f by fixing L to any particular value. In particular, if $L = 1$, the

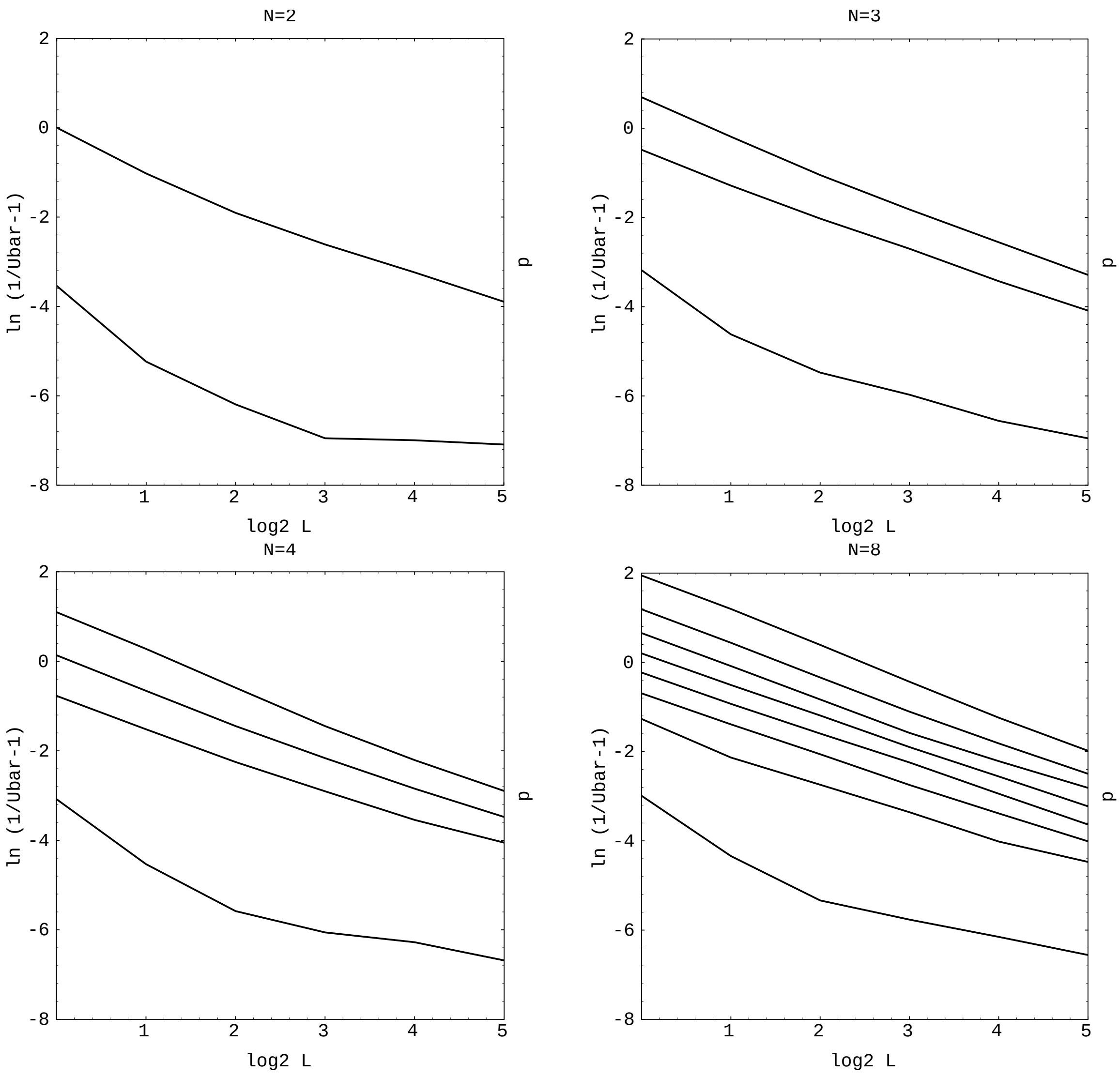


Figure 4.5: Plots of $\log(1/\overline{U}-1)$ **vs.** $\log_2 L$**.** Each plot corresponds to a different value of N. Within each plot, each line represents a plot of $\log(1/\overline{U}-1)$ versus $\log_2 L$ for a different value of p. From top to bottom the values for p are $1/N, 2/N, \ldots$, and N/N, respectively.

resulting system has no queueing and $U \simeq p$. Substituting, we have $p \simeq 1/(1 + \frac{1}{L}f(p, N))$ and solving for $f(p, N)$ we have $f(p, N) = 1/p - 1$. This results in the same analytic model, $U \simeq 1/(1 + \frac{1}{L}(1/p - 1))$ derived in Section 4.4. The reader can look upon the above result as an alternative derivation of the analytic model based on empirical techniques.

Having rederived the analytical model above, two questions immediately arise:

1. Can we get a more accurate model by solving for utilization with $L = 2$ rather than $L = 1$? Theoretically, a solution to such a model would take into account a greater amount of the interaction between processes and should result in a more accurate model.

2. How good is the best model of the form $U \simeq 1/(1 + \frac{1}{L}f(p, N))$?

The second question is easily answered empirically; we simply calculate the values of $f(p, N)$ which maximizes R^2 and examine the errors of the resulting model. The first question is more difficult to answer. Even for $L = 2$, we must take into account both queueing and fork-join synchronization. We model the system with $L = 2$ as a discrete-time discrete-state Markov chain. If we assume that disk service times are constant, the number of states required to model the system is equal to the number of disks in the system. That is, the queue length at each disk is either zero or one, disks with a queue length of one are always consecutively located in the disk array, and the state where every disk has a queue can be merged with the state where no disk has a queue. We have formulated and solved such a model for arbitrary N. The solution is complex enough and would require sufficient explanation that it is not presented here.

Let $\hat{U}_L$ represent the best empirical model of the form $U \simeq 1/(1 + \frac{1}{L}f(p, N))$, let

$\hat{U}_{L1} \equiv 1/(1 + \frac{1}{L}(1/p - 1))$, and let $\hat{U}_{L2}$ represent the approximate solution for utilization derived by assuming $L = 2$. The table below tabulates the error metrics for $\hat{U}_L$, $\hat{U}_{L1}$ and $\hat{U}_{L2}$.

Model	R^2	$1 - R^2$	*max error*	*90 % error*
$\hat{U}_L$	0.9929	0.0071	0.1299	0.0642
$\hat{U}_{L1}$	0.9814	0.0186	0.1863	0.0987
$\hat{U}_{L2}$	0.9814	0.0186	0.2176	0.0920

The error metrics of $\hat{U}_{L1}$, the analytic model derived in Section 4.4, compares favorably with the error metrics of $\hat{U}_L$, the best empirical model of the form $U \simeq 1/(1 + \frac{1}{L}f(p, N))$. Somewhat surprisingly, The *max error* for $\hat{U}_{L2}$ is larger than that for $\hat{U}_{L1}$ although the *90 % error* is smaller. Although not visible from the table due to roundoff, R^2 for $\hat{U}_{L2}$ is slightly larger than that for $\hat{U}_{L1}$. We conclude that the additional complexities of $\hat{U}_{L2}$ does not, in general, merit its use over $\hat{U}_{L1}$. To more accurately gauge the differences between $\hat{U}_L$ and $\hat{U}_{L1}$, Figure 4.6 plots $\hat{U}_L$, $\hat{U}_{L1}$ and the 90 percentile intervals as a function of N, L and p.

4.8 Variable Request Sizes

In this section, we extend our model to handle variable sized workloads. Although we derived Equation 4.3 assuming a constant request size, it can be easily extended by noting that the parameter p is just the probability that a given request will access a given disk. Thus, given a workload that is f_1 fraction requests of size p_1 and $f_2 = 1 - f_1$ fraction requests of size p_2, $p = f_1p_1 + f_2p_2$. In general, if x is the size of requests as a fraction of

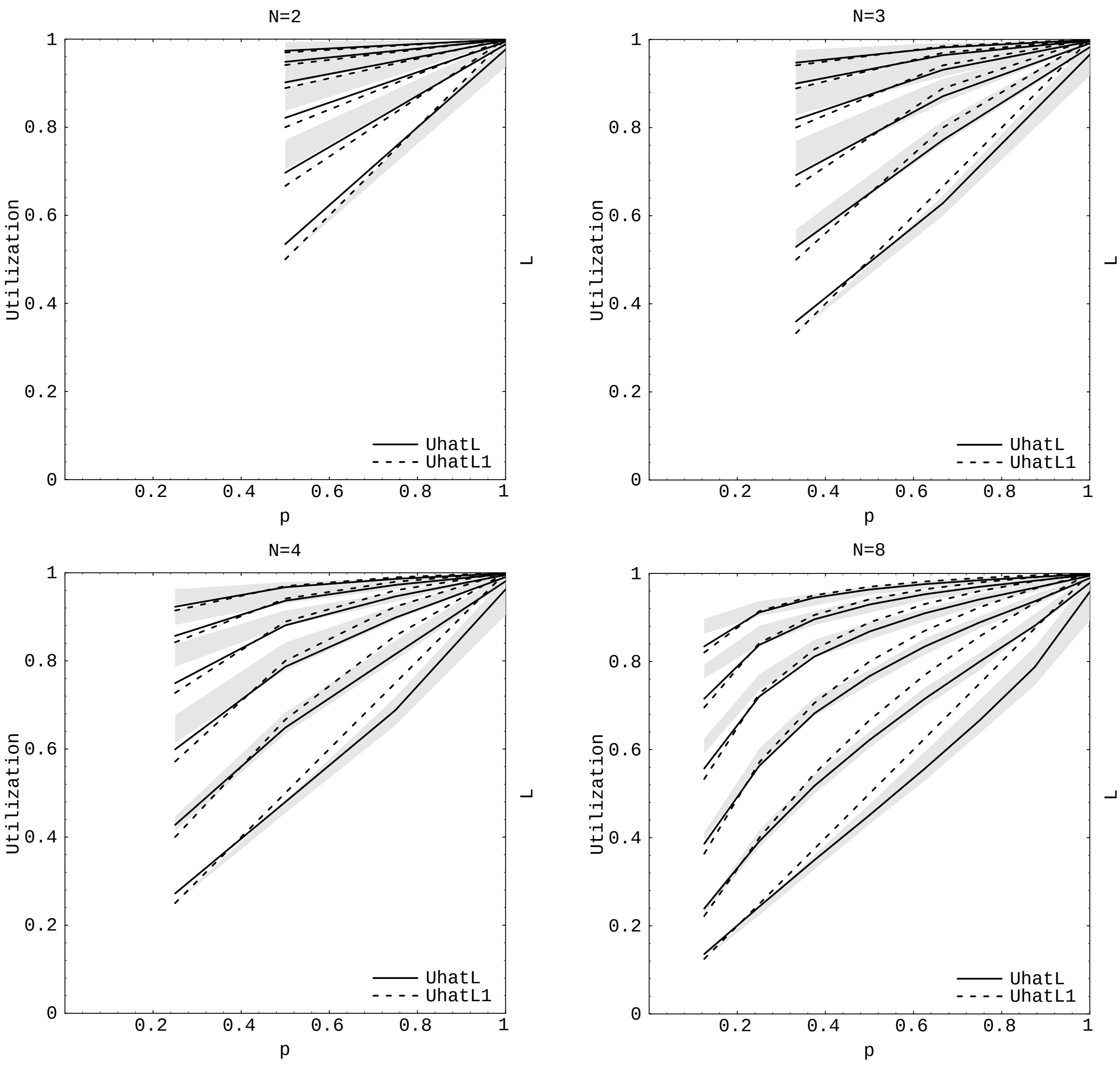


Figure 4.6: $\hat{U}_L$ vs. $\hat{U}_{L1}$. Each plot corresponds to a different value of N. Within each plot, each set of solid line, dotted line and shaded region represents $\hat{U}_L$, $\hat{U}_{L1}$ and the 90 percentile intervals, respectively, for a different value of L. From bottom to top the values for L are 1,2,4,8,16 and 32, respectively.

the disk array and $F(x)$ its corresponding cumulative distribution function. Then,

$$p = \int_0^1 x \; dF(x) = \overline{p} \tag{4.6}$$

where $\overline{p}$ is the average request size as a fraction of the disk array.

To validate the above result, consider the design set consisting of the following parameter and factor levels:

Parameter	Parameter Value
diskModel	Fujitsu
N	8
B	32 KB

Factor	Factor Levels
L	1, 2, 4, 8, 16, 32
p_1	$(2, \ldots, N)/N$
p_2	$(1, \ldots, p_1 N - 1)/N$
f_1	0.20, 0.40, 0.60, 0.80

Since we have shown in Section 4.7.1 that utilization is insensitive to *diskModel,* N and B, we hold them constant. In each simulation run, L processes randomly issue requests of size p_1, f_1 fraction of the time, and requests of size p_2, $1 - f_1$ fraction of the time. Each design point in the design set is simulated twice with two different random seeds to quantify the experimental error. Thus, the total number of simulation runs is $2(6 \times 28 \times 4)) = 1344$. In each simulation run, the disks are rotationally synchronized, requests to the disk array are aligned on stripe unit boundaries and L is held constant.

The following are the error metrics from the experiment where $\hat{U} \equiv 1/(1+\frac{1}{L}(1/\overline{p}-1))$.

Model	R^2	$1 - R^2$	*max error*	*90 % error*
BEST	0.9986	0.0014	0.0801	0.0192
$\hat{U}$	0.9969	0.0031	0.0934	0.0300

The table shows that the analytic model that only uses the factors L and $\overline{p}$ compares favorably with the best empirical model that uses all the factors L, p_1, p_2 and f_1.

Figure 4.7 plots the response predicted by the analytic model $\hat{U}$ together with the maximum error intervals and individual data points as a function of L and $\overline{p}$. The analytically predicted response approximately passes through the simulated data points.

4.9 The Optimal Stripe Unit Size

In this chapter, we will illustrate the utility of the analytic model derived in the previous chapter by using it to derive an equation for the *optimal stripe unit size*, the stripe unit size that maximizes throughput in bytes per second. The equation for the optimal stripe unit size is useful as a rule of thumb in configuring disk arrays and also provides valuable insights into the factors that influence the optimal stripe unit size. Given today's disk technology and disk arrays of approximately eight disks in size, the optimal stripe unit equation is most useful for servicing I/O requests that are a couple of hundred or more kilobytes in size. Miller [29] has shown that such workloads are typical of scientific applications. For such workloads, simulation shows that there is typically a 20% degradation in performance when the stripe unit is a factor of two smaller or larger than the optimal size.

In addition to deriving the equation for the optimal stripe unit size, we will show that the stripe unit size that maximizes throughput also minimizes response time. Note,

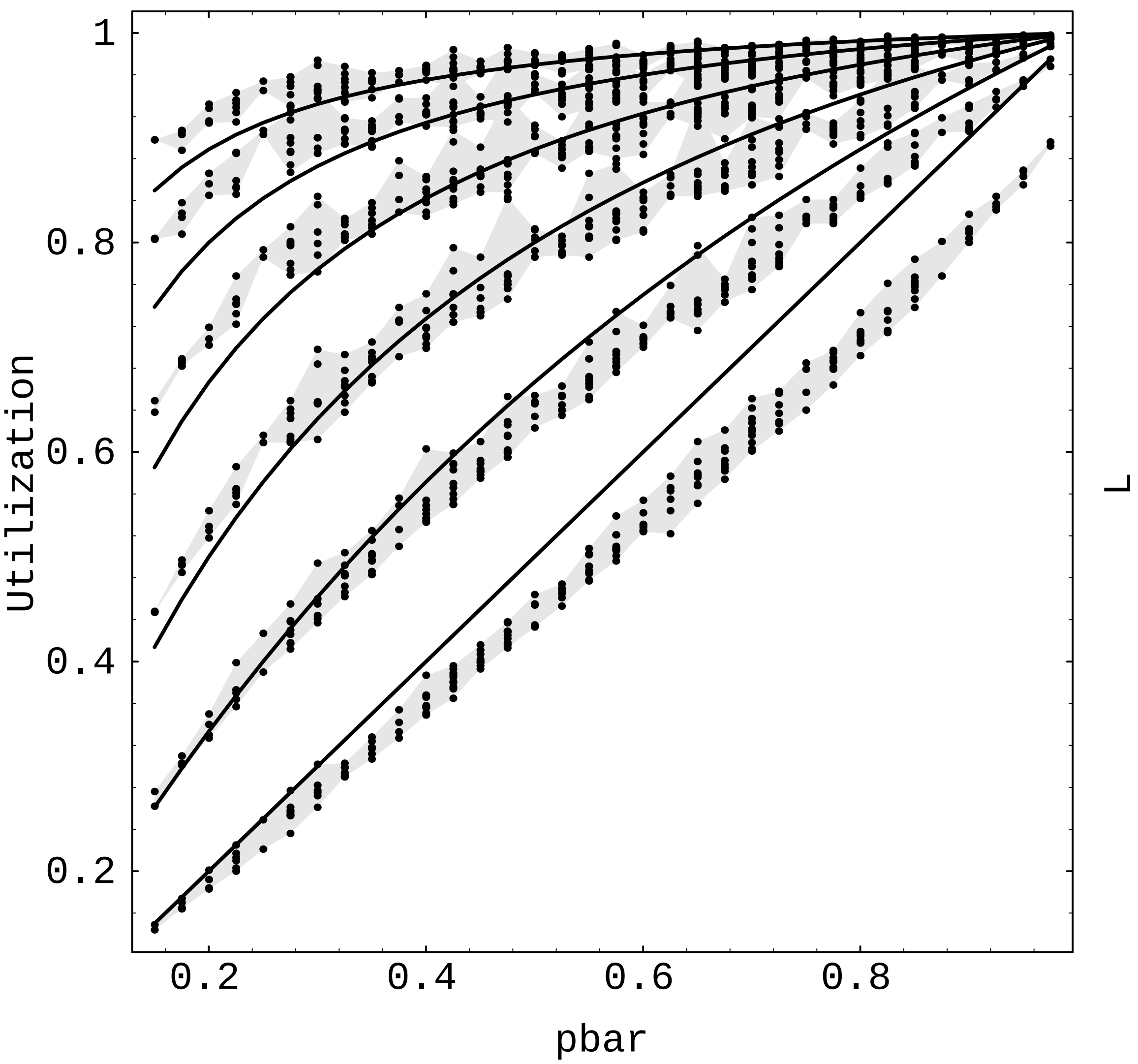


Figure 4.7: Analytic U with Maximum Error Intervals and Individual Data Points. In the above plot, *diskModel* = Fujitsu, $N = 8$ and $B = 32\,\mathrm{KB}$. Each line, shaded region and set of datapoints corresponds to a different value of L. From bottom to top the values for L are 1,2,4,8,16 and 32, respectively.

however, that maximizing throughput is *not* the same as maximizing utilization; just because a disk is busy does not mean that it is doing useful work. The fundamental tradeoff in selecting a stripe unit size is one of parallelism versus concurrency. Small stripe unit sizes increase the parallelism for a single request by mapping the request over many disks, but reduce concurrency because each request uses more disks [7].

4.9.1 Derivation

We will derive the equation for the optimal stripe unit size from Equation 4.5. But first, because the disk service time, S, is dependent on the stripe unit size, B, we must formulate a simple model that makes this dependency explicit. Consider the following definitions:

T	Throughput in bytes per second.
$E(S)$	Expected disk service time.
P	Average disk positioning time (seek + rotational latency).
X	Sustained disk transfer rate.
N	Number of disks in the disk array.
L	The number of processes issuing I/O requests.
B	Size of the stripe unit (block size).
Z	Request size in bytes.

Then,

$$E(S) = P + B/X. \tag{4.7}$$

Note that n, the number of stripe units per request can be calculated as follows:

$$n = Z/B. \tag{4.8}$$

Substituting equations 4.3, 4.7 and 4.8 into Equation 4.5 and simplifying we can write the throughput in bytes per second as,

$$T = \frac{LNXBZ}{(PX + B)(NB + Z(L-1))}. \tag{4.9}$$

Solving for the local maxima in the above equation as a function of B we get the following equation for the optimal stripe unit size:

$$B_{opt} = \sqrt{\frac{PX(L-1)Z}{N}}. \tag{4.10}$$

Repeating the above procedure to minimize response time starting from Equation 4.4 results in the same equation for the optimal stripe size; thus, *the stripe unit size that maximizes throughput also minimizes response time* and is given by Equation 4.10.

The following remarks can be made about Equation 4.10:

- Changes to the system that increase the effective load, that is, an increase in L, an increase in Z, or a decrease in N, favor larger optimal stripe units. The opposite is true for changes that decrease the effective load.

- In our model system, the optimal stripe unit size is dependent only on the product PX, the relative rate at which a disk can position and transfer data, and not on P or X independently. If you replace the disks with those that position and transfer data twice as quickly, the optimal stripe unit size remains unchanged [7]. In this respect, the selection of an optimal stripe unit size is a trade-off between the disk positioning time and the data transfer time.

4.9.2 Validation

As a further validation of the analytic model and of Equation 4.10 in particular, we compare the analytic values for the optimal stripe unit size with empirically determined values. Figure 4.8 plots the analytically determined optimal stripe unit sizes versus the empirically determined optimal stripe unit sizes on a log-log scale. The shaded regions on the figure represent optimal stripe unit sizes that can be ruled out for the following reasons. First, throughput when $B < Z/N$ for fixed Z is less than or equal to the throughput when $B = Z/N$. At this stripe unit size, requests are being distributed uniformly across all disks and it is not possible to increase parallelism or concurrency by reducing the stripe unit size. Second, throughput when $B > Z$ for fixed Z is identical to when $B = Z$. In this case, the request already fits completely within a single disk and there is no advantage or disadvantage to increasing the stripe unit size. We have empirically verified the above two facts. Thus, $Z/N \leq B \leq Z$.

For comparison purposes, Figure 4.9 adds the optimal stripe unit sizes predicted by Chen [7]. Chen's model assumes that the optimal stripe unit size is independent of the request size. As can be seen, our optimal stripe unit sizes correspond well with Chen's optimal stripe unit sizes and both correspond closely with the measured optimal stripe unit sizes.

To get a feel for the sensitivity of performance to the choice of stripe unit size, Figure 4.10 individually plots each group of lines from Figure 4.9 with vertical bars to indicate the range of stripe unit sizes providing 95% of the throughput of the optimal stripe unit size.

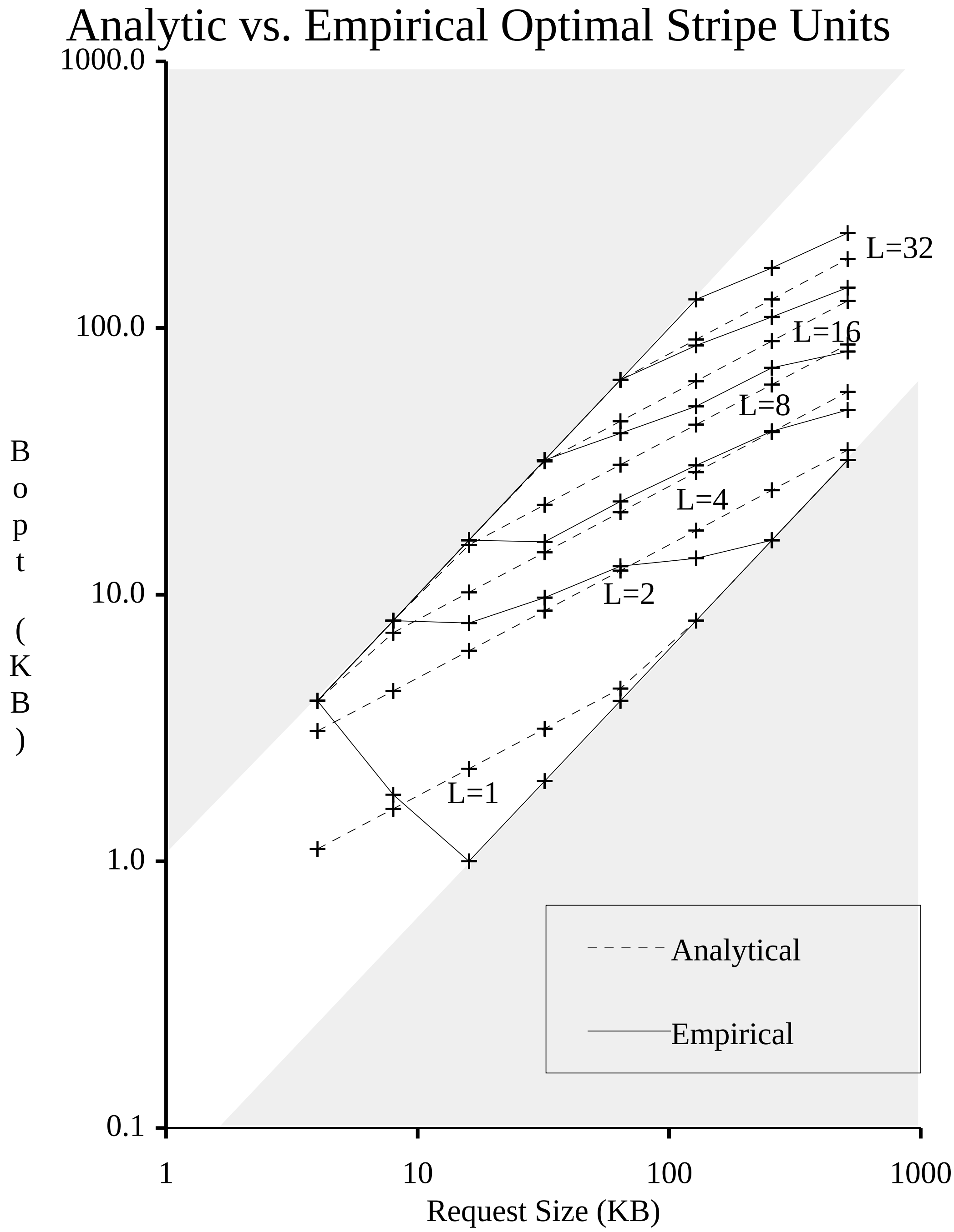


Figure 4.8: Analytic vs. Empirical Optimal Stripe Unit Sizes. $N = 16$; $B = 32KB$; $L \in \{1, 2, 4, 8, 16, 32\}$; Each set of three lines is labeled with its corresponding value of L. B_{opt} = Optimal stripe unit size. The graph may appear odd at first because the beginning and end of the individual lines overlap at the edges of the shaded regions.

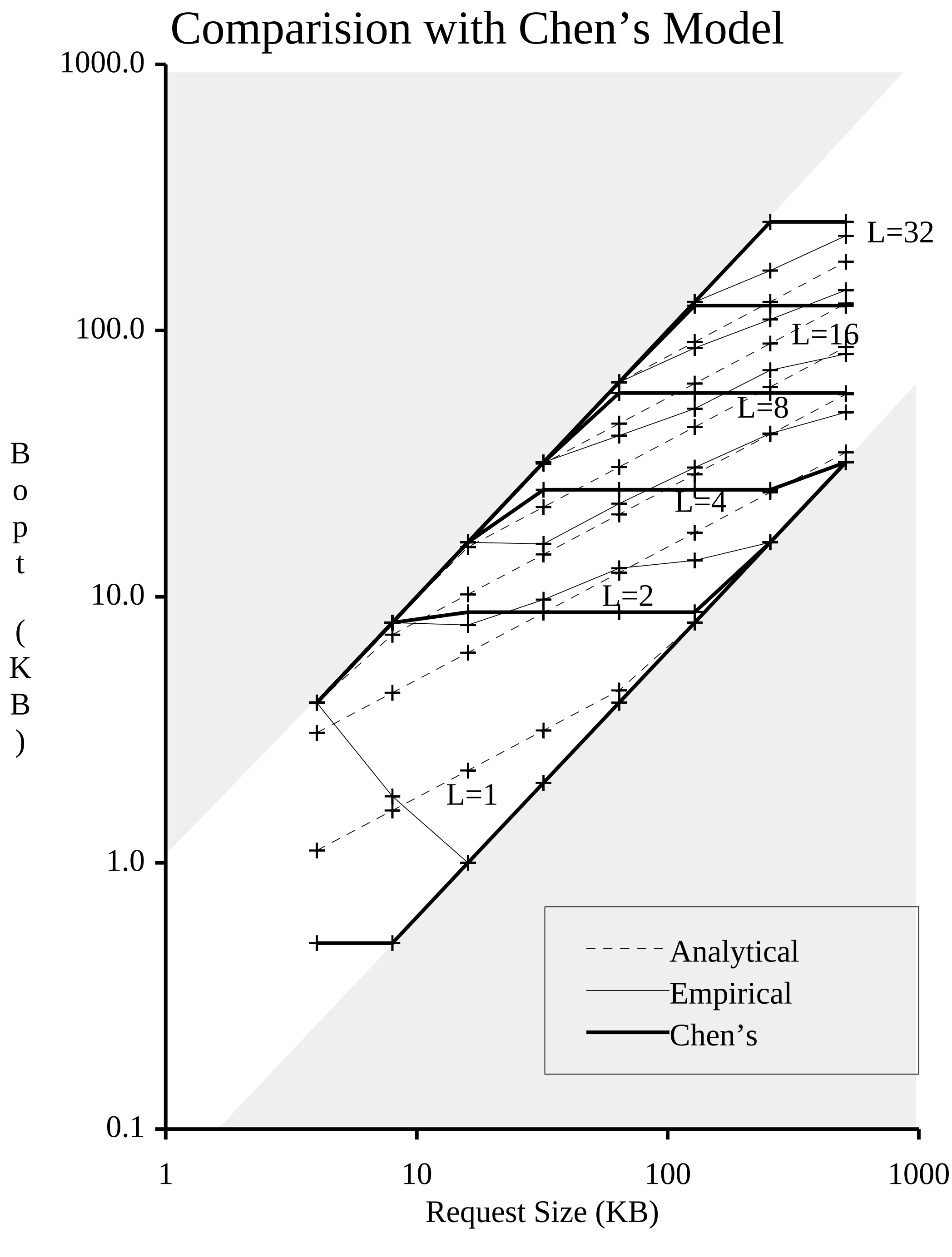


Figure 4.9: Comparison with Chen's Model. $N = 16$; $B = 32KB$; $L \in \{1, 2, 4, 8, 16, 32\}$; Each pair of lines is labeled with its corresponding value of L. B_{opt} = Optimal stripe unit size.

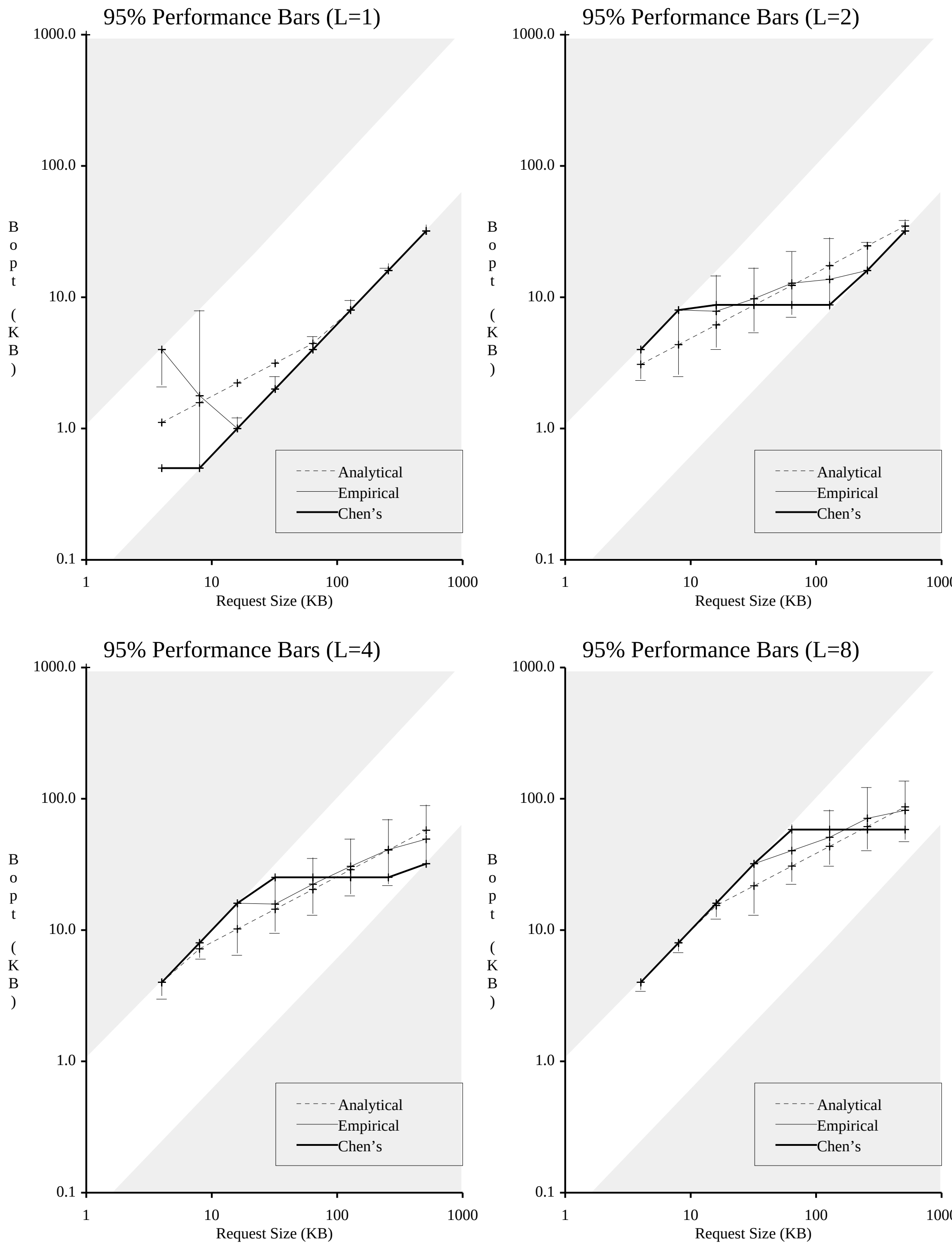


Figure 4.10: Optimal Stripe Unit Sizes with 95% Performance Intervals. $N = 16$; $B = 32KB$; B_{opt} = Optimal stripe unit size. Stripe unit sizes within the solid vertical bars provide 95% of the performance of the optimal stripe unit size indicated by the oblique thin solid line.

There is a good correspondence between the analytically and empirically determined values of the optimal stripe unit size. Except when $L = 1$ and request sizes are small, the optimal stripe unit sizes determined by both Chen's model and our model lie within the 95% performance intervals.

4.10 Summary

In this section, we derived and validated an analytic performance model for disk arrays. We initially modeled disk arrays as a closed queueing system consisting of a fixed number, L, of processes continuously issuing requests of a fixed size, p, to a disk array consisting of N disks. We then extended the model to handle variable sized requests. The resulting model predicts the expected utilization of the model system, U, as $\frac{1}{1+\frac{1}{L}(1/\overline{p}-1)}$ where $\overline{p}$ is the average size of requests as a fraction of the number of disks in the disk array. We then derived the expected response time and throughput as a function of utilization. We showed via simulation that the simulated utilization is generally within $\pm 10\%$ of the utilization predicted by the analytic model. We also examined the error introduced by each approximation made in the derivation of the analytic model to better understand the validity of the approximations. Finally, we applied the analytic model to show that the optimal size of data striping simultaneously maximizes throughput and minimizes response time and is equal to $\sqrt{\frac{PX(L-1)Z}{N}}$ where P is the average disk positioning time, X is the average disk transfer rate and Z is the request size.

To the best of our knowledge, we have presented the first analytic performance model that treats disk arrays as a closed queueing system and handles a continuum of request sizes from small requests to large requests. The primary weaknesses of the model are twofold. First, it assumes that processes do nothing but issue I/O requests. In particular, it does not model CPU-think-time. Second, although the model can be used to analyze reads in RAID level 5 disk arrays, it can not be used to analyze writes. Because of the above limitations, the next chapter will develop and apply an empirical technique based on *utilization profiles*, which can be used to analyze a much wider variety of systems under a wide range of workloads.

Chapter 5

Utilization Profiles

The performance analysis of real systems is currently an art with techniques ranging from analytic models to measurement. When feasible, analytic models are the most desirable because they can be quickly evaluated, are applicable under a wide range of system and workload parameters, and can be manipulated by a range of mathematical techniques. Unfortunately, analytic models are highly limited in the types of workloads and systems they can analyze.

Another approach to performance analysis is based on measuring the system of interest over a large range of system and workload parameters. Such measurements can be used to accurately interpolate the performance of the system over the measured range of parameters. Unfortunately, it is dangerous to directly extrapolate performance from such measurements and the large number of measurements is very cumbersome to manipulate.

A technique that falls between the purely analytic and measurement based approaches is to construct empirical performance models from the measurements using high-level information about the measured system. At the crudest level, this may consist of nothing more than curve-fitting the measured data with arbitrary functions. At its most refined, the measurements may be used to parameterize a mostly analytic model. In both cases, however, the basic techniques are similar; we seek to *summarize* the vast quantities of measured data in a mathematical form by using additional information specific to the measured system.

This section presents one method for summarizing the measured data using what we call *utilization profiles*. We will show that utilization profiles are useful for system level performance characterization and are especially useful in analyzing throughput oriented systems. The following are some of the questions that can be answered by utilization profiles:

- What is the bottleneck at a given workload?
- What will be the bottleneck if you improve the bottleneck resource?
- How much do you need to improve the bottleneck resource before it stops being the bottleneck?
- What is the utilization of a given resource at a given system throughput?

In the following sections, we will a present a simple way of viewing utilization profiles, an example application of utilization profiles in analyzing a single disk system, a generalization of the technique to analyze blackbox systems, and an alternative but equivalent way of viewing utilization profiles. We will see that the two different views of utilization profiles each stress a different aspect of the analysis technique and provide different insights into how utilization profiles can be used to model real systems.

5.1 First Look at Utilization Profiles

One view of utilization profiles is based on modeling the expected service time of requests at a given resource as a linear combination of arbitrary functions of system and workload parameters. Here it is important to distinguish between the terms *service time* and *response time*. Service time is the actual amount of time spent by a resource servicing a request whereas response time is the total elapsed time between the issuing of a request and its completion. Thus, in addition to service time, response time may include queueing and synchronization delays. By arbitrary functions, we mean that we do not impose any restriction on the mathematical form of the functions. Mathematically, we can summarize the above statements as follows.

$$S = k_1 f_1 + k_2 f_2 + \ldots + k_n f_n \tag{5.1}$$

$$S = \vec{k} \cdot \vec{f} \tag{5.2}$$

S Mean service time.
$\vec{k}$ *Utilization profile.*
A constant vector.
$\vec{f}$ *Throughput transform.*
A vector of arbitrary functions.

Note that S in this chapter refers to the *mean* service time whereas S in Chapter 4 refers to the service time. Henceforth, we will refer to the vector of constants, $\vec{k}$, as the *utilization profile* and the vector of functions, $\vec{f}$, as the *throughput transform*.

Strictly speaking, each resource in the system is characterized by its own utilization profile and throughput transform. We will frequently find, however, that all the resources in a system can be conveniently characterized by a single throughput transform. This means that many systems can be characterized by a single throughput transform and a utilization profile for each resource in the system. Further simplification results if the system contains multiple identical resources that are uniformly utilized, such as disks in a disk array. In such cases, a single utilization profile can be used to characterize all the disks.

For a given resource in the system, the throughput transform, $\vec{f}$ is not unique; there are usually many reasonable choices. The particular throughput transform that is used depends on the characteristics of the system that are of interest, the knowledge of the system that is available for modeling the system, and the desired accuracy of the model. In general, the throughput transform characterizes a "class" of systems and the utilization profile characterizes a specific implementation of a system. For example, once determined, the same throughput transforms can be used to analyze most RAID level 5 systems, where the systems will differ is in the values of their utilization profiles. This makes it convenient to compare similar systems by directly comparing their utilization profiles.

5.2 An Example: Utilization Profile for a Disk

In this section, we solidify the abstract concepts discussed in the previous section by calculating the utilization profile for a disk. We start by modeling the expected service time of the disk as a sum of the average data positioning time, k_1, and data transfer time, k_2Z, where k_2 is the time needed to transfer a single byte of data and Z is the request size in bytes.

$$S = k_1 + k_2 Z \tag{5.3}$$

$$\vec{k} = (k_1, k_2) \tag{5.4}$$

$$\vec{f} = (1, Z) \tag{5.5}$$

S	Expected service time.
Z	Request size in bytes.
k_1	Per request overhead (latency).
k_2	Per byte overhead (1/transfer-rate).
$\vec{k}$	Utilization profile.
$\vec{f}$	Throughput transform.

In this example, the utilization profile $\vec{k}$ consists of the constants k_1 and k_2 and the throughput transform $\vec{f}$ consists of the functions 1 and Z. In general, we will find that the utilization profile is a vector of overhead values and the throughput transform corresponds to the number of each type of overhead in each request. In this example, k_1 is the per request overhead and k_2 is the per byte overhead; correspondingly, there is one per request overhead per disk request and Z per byte overheads per disk request.

To calculate the values of the utilization profiles, we can measure the expected service times for a range of request sizes and use linear regression to solve for the value of $\vec{k}$ that minizes of sum of squared errors. Thus, given the following measurements,

Z	S
1 KB	21 ms
4 KB	24 ms
8 KB	27 ms
16 KB	37 ms

we get the following system of linear equations:

$$S = k_1 + k_2 \times Z \tag{5.6}$$

$$21\,\mathrm{ms} = k_1 + k_2 \times 1\,\mathrm{KB} \tag{5.7}$$

$$24\,\mathrm{ms} = k_1 + k_2 \times 4\,\mathrm{KB} \tag{5.8}$$

$$27\,\mathrm{ms} = k_1 + k_2 \times 8\,\mathrm{KB} \tag{5.9}$$

$$37\,\mathrm{ms} = k_1 + k_2 \times 16\,\mathrm{KB} \tag{5.10}$$

The above system is overconstrained and has no solution. If we use linear regression to minimize the sum of squared errors, we get the following value for $\vec{k}$:

$$\vec{k} = (20\,\mathrm{ms}, 500\,\mathrm{ns}) \tag{5.11}$$

That is, the average positioning time of the disk is 20 msand the sustained transfer rate of the disk is $1/500\,\mathrm{ns}=\ 2\,\mathrm{MB/s}$.

As this example illustrates, the throughput transform characterizes a class of systems whereas the utilization profile characterizes a specific implementation of the system. In the context of the current example, we see that the throughput transform $(1, Z)$ can be used to analyze many other disks, each of which will have a different utilization profile. Comparing two or more disks by directly comparing the utilization profiles can be simpler and more complete than comparing the performance of the disks over a range of workloads. This is because, given a reasonable model for the disk service time, the utilization profiles are very effective in characterizing the performance of disks. Similar analysis is possible for more complex systems consisting of many more resources. As we will see, the benefits from such analysis is even greater for the complex systems than for the simple disk system we analyzed here.

To summarize, in the above disk example, we used the following steps to determining the utilization profile.

1. Identify the relevant overheads. In the disk example, we assumed that the relevant overheads are the per request overhead and the per byte overhead.

2. Determine the corresponding throughput transform by noting the number of each type of overhead per request. In the disk example, we noted that there is one per request overhead per disk request and Z per byte overheads per disk request.

3. Measure the resource utilization and system throughput over a range of system and workload parameters.

4. Use linear regression to solve for the utilization profile.

5.3 Analysis of Blackbox Systems

The previous section provides a simple method for computing the utilization profiles for a system when the expected service time can be measure for each resource in the system. Frequently, however, it is not possible to measure the performance of each resource in the system. In many cases, the system must be treated as a blackbox. This section presents a method for computing the utilization profiles of resources in blackbox systems by expanding on the techniques previously presented.

Since most systems contain more than one resource, it will be useful at this point to rewrite Equation 5.2 to reflect this fact. For the purposes of analyzing blackbox systems, it will also be useful to express Equation 5.2 as a function of the resource's utilization and the system's overall throughput.

$$
\begin{aligned}
S_i &= \vec{k}_i' \cdot \vec{f}_i && (5.12)\\
U_i/T_i &= \vec{k}_i' \cdot \vec{f}_i && (5.13)\\
U_i/T &= \vec{k}_i' \cdot \vec{f}_i' && (5.14)
\end{aligned}
$$

S_i	Expected service time at resource i.
U_i	Utilization of resource i.
T_i	Throughput at resource i.
T	Overall system throughput.
$\vec{k}_i, \vec{k}_i'$	Utilization profile of resource i.
$\vec{f}_i, \vec{f}_i'$	Throughput transform of resource i.

Here, we have used two operational laws, the *utilization law*, which relates the expected service time at a resource to the utilization and throughput at the resource, and the *forced flow law*, which relates the throughput at a resource to the overall system throughput. The laws are very general and apply to both open and closed systems. The application of the forced flow law changes the values of the utilization profile and throughput transform but not the general form of the equation. The values of $\vec{k}_i$ and $\vec{f}_i$ can easily be computed from $\vec{k}_i'$ and $\vec{f}_i'$ and visa versa.

To apply Equation 5.14 in analyzing real systems, we need to measure the utilization of a given resource and the overall system throughput over a range of system and workload parameters. The system throughput is easily measured, but it is not always possible to measure the utilization of a given resource in the system. Fortunately, there are certain conditions under which the utilization of a resource can be determined without direct measurements. A trivial example is when the system is completely idle. In this case, the utilization of all resources in the system is 0%. A more useful case is when the system is operating at a bottleneck. In this case, the utilization of some resource in the system is close to 100 %. It is easy to identify the bottleneck states because in such a state, increasing the effective system workload does not result in an increase in system throughput. Thus, if we analyze the system only when it is bottlenecked, we can rewrite Equation 5.14 as follows:

$$1.0/T = \vec{k}_i \cdot \vec{f}_i \tag{5.15}$$

This eliminates utilization from the equation and allows us to calculate utilization profiles by measuring only the overall system throughput.

Until now, we have ignored an important practical consideration to blackbox analysis. Although it is easy to determine when a system is bottlenecked, it is not as easy to determine which resource is causing the bottleneck. Given two bottleneck points, a different resource may be responsible for each of the two bottlenecks. Although it is not essential to identify the exact resource that is responsible for each bottleneck, it is necessary to separate the bottleneck points belonging to different resources since the calculation of utilization profiles is performed on a per resource basis.

Because Equation 5.15 is linear, the process of separating the bottleneck points for each resource and computing the corresponding utilization profiles is analogous to using planar surfaces to fit the sides of a multidimensional polyhedron. The process is actually somewhat more complicated since a good value for the throughput transform may not be apparent. This means that the choice of the throughput transform and the calculation of the utilization profiles may have to proceed concurrently. In reference to the polyhedron example, changing the throughput transform is analogous to changing the shape of the polyhedron. In essence, we are trying to fit the surfaces of a polyhedron that simultaneously changes shape!

To summarize, the following steps should be followed in determining utilization profiles for a blackbox system:

1. Measure the system throughput over the range of system and workload parameters of interest.

2. Isolate the bottleneck throughputs.

3. Itemize the relevant overheads. A reasonable first guess can be made based on the system and workload parameters and a general knowledge of the system's logical function. In the worst case, when we have no intuition about the operation of the system, we can perform a "blind" statistical analysis by modeling the expected service time as a first order polynomial and gradually adding higher order terms as necessary.

4. Determine the corresponding throughput transform by noting the number of each type of overhead per request.

5. Use linear regression to fit the bottleneck throughputs and solve for the utilization profiles. The linear regression will also indicate if certain terms in the throughput transform are not useful and can be eliminated.

6. At this point, a poor fit may indicate that more than one resource is responsible for the bottleneck or that a new throughput transform should be tried. In the latter case, the graph of the fitted dataset can be used to guide in partitioning the dataset into two or more datasets such that the bottlenecks in each dataset are caused by a

single resource. If this does not result in a better fit, it is necessary to try a different throughput transform. Go back to Step 3 until a satisfactory fit is achieved.

5.4 Second Look at Utilization Profiles

Section 5.1 has defined utilization profiles from the view of modeling the expected service time of resources. In this section, we present an alternative throughput-oriented view based on a system with constant service time requests. This alternative view provides a better understanding of utilization profiles and insights that are useful in modeling real systems.

Consider a system consisting of a single resource that services two different types of requests: type-1 and type-2. Assume that each type-i request requires a constant amount of time, k_i, to service. If n_i type-i requests are serviced during a time interval t, then the total time spent service requests, Ut, where U is the utilization of the resource during time interval t, can be expressed as follows:

$$Ut = k_1 n_1 + k_2 n_2. \tag{5.16}$$

By dividing through by t, we have

$$U = k_1 T_1 + k_2 T_2. \tag{5.17}$$

Where $T_i = n_i/t$ is the throughput of type-i requests. By applying the forced flow law, we can express the above equation in terms of the system throughput, T, of both type-1 and type-2 requests as follows:

$$U = k_1 f_1 T + k_2 f_2 T. \tag{5.18}$$

Here, f_i is a function that maps the system throughput to the throughput of type-i requests. This is the reason we refer to the f_i, collectively, as a throughput transform. By rearranging terms we have

$$U/T = k_1 f_1 + k_2 f_2, \tag{5.19}$$

which is very similar to Equation 5.14. If we have N identical resources with uniformly distributed load, we can rewrite Equation 5.18 as

$$U = k_1 f_1 T/N + k_2 f_2 T/N. \tag{5.20}$$

That is, each resource gets only $1/N$ the requests it would have gotten if there were only one resource. Technically, the $1/N$ term could be incorporated into the f_i terms but we will find it useful to make f_i independent of the number of identical resources in the system. Rearranging, we finally have

$$UN/T = k_1 f_1 + k_2 f_2. \tag{5.21}$$

More generally we have

$$UN/T = k_1 f_1 + k_2 f_2 + \ldots + k_n f_n, \tag{5.22}$$

$$UN/T = \vec{k} \cdot \vec{f}. \tag{5.23}$$

which is a more general form of Equation 5.14. Specializing Equation 5.14 to bottleneck

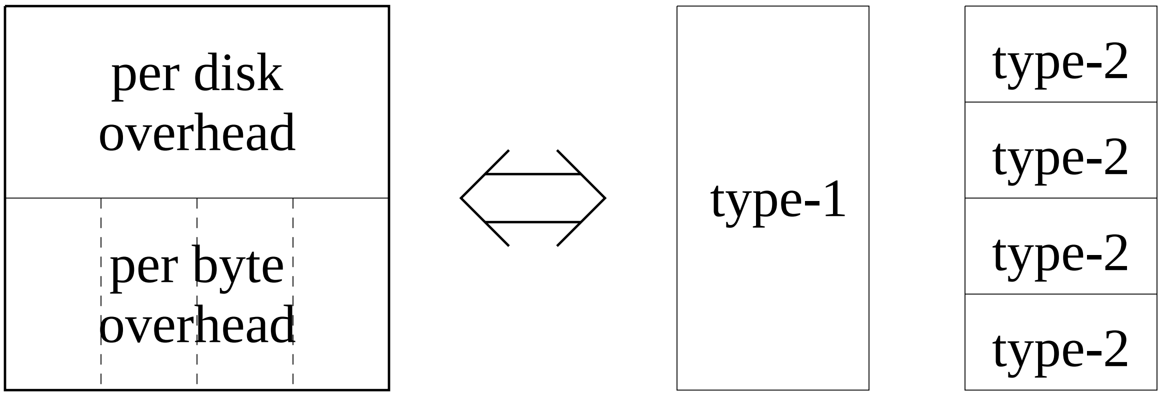


Figure 5.1: Compound Requests. Requests whose service times depend on system or workload parameters are modeled as a collection of simpler requests, each of which requires a constant amount of time to service.

throughputs, we have

$$N/T_{max} = \vec{k} \cdot \vec{f} \tag{5.24}$$

which is a generalization of Equation 5.15.

One problem with the derivation we have presented so far is that all type-i requests must require a constant amount of time, k_i, to service. In real systems, however, the service time of requests usually depend on the system and workload parameters. For example, it usually takes a disk more time to service an 8 KBrequest than it takes to service a 4 KBrequest. We get around this problem by modeling a real request as a compound request consisting of many simple requests, each of which takes a constant amount of time to service. For example, Figure 5.1 illustrates the modeling of a four byte disk request as a compound request consisting of a single type-1 request representing the per request overhead and four type-2 requests representing the per byte overheads. By generalizing this example, we can model arbitrarily complex requests by breaking them down into simpler components each of which takes a constant amount of time to service.

In the next section, we will apply the concepts discussed in this and the preceding sections to analyze the performance of RAID-II, our second disk array prototype. We will see that utilization profiles is a flexible tool that can systematically characterize the performance of complex systems. The analysis of RAID-II should also show that utilization profiles is a general tool that can be used to analyze a wide variety of systems in addition to disk arrays.

5.5 Analysis of RAID-II

In this section we analyze the performance of RAID-II's RAID level 5 disk subsystem as a blackbox. By blackbox, we do not mean that we have absolutely no information about the system but that we cannot directly measure the performance of individual components of the system. In particular, we examine in detail the performance of three access modes: read, read-modify-write, and reconstruct-write. These access modes, discussed in Section 2.3, are the only modes used during the normal operation of a RAID level 5 storage system.

The measurement environment consists of RAID-II configured with four Interphase Cougar disk controllers, eight SCSI strings and twenty-four SCSI disks in a RAID level 5 configuration. We use our measurement program, *raidPerf*, to fork a fixed number of processes that issue I/O requests to the disk array. In our measurements, data will be transferred between the X-Bus board's memory system and the disk array but not over the network. We are interested in analyzing the following factors for both reads and writes from/to the disk array.

N	Number of disks in disk array.
B	Block/Stripe-unit size.
n	Request size in stripe units.
L	Load (number of processes).

Note that nB equals the request size in bytes.

The design set consists of the complete crossproduct of the following factor levels.

Factor	Factor Levels
N	$2, 3, 4, 8, 12, 16, 20, 24$
B	$(1, 2, 4, 8, 16, 32, 64)$ KB
L	$1, 2, 4, 8, 16, 32$
n (reads)	$1, 2, \ldots, N$
n (writes)	$1, 2, \ldots, N-1$

For writes, the request size, n, ranges up to $N-1$ instead of N because the parity disk must also be updated. Each design point is measured twice to quantify the experimental errors. The total number of design points is $2(7 \times 6 \times (89 + 81)) = 14,280$.

5.5.1 Analysis of Reads

This section defines and calculates the utilization profiles for the read access mode on RAID-II. An additional purpose is to familiarize the reader with the analysis of real system using utilization profiles. Thus, this section will present a more detailed exposition of the process than later sections.

As described in Section 2.3, the read access mode is serviced by simply reading the requested data from the specified disks. We will assume that the relevant overheads are the per array request overhead, per disk overhead, and the per byte overhead. That is, since each array request is broken into independent disk requests, any resource that services an array request is likely to pay a fixed overhead for each array request, a fixed overhead for each disk accessed and a fixed overhead for each byte transferred. Thus, the throughput transform for the read access mode can be represented as,

$$\vec{f} = (1, n, nB). \tag{5.25}$$

That is, there is one array request overhead per read request, n disk overheads per array request, and nB byte overheads per array request.

At this point, we can simplify our analysis by identifying the resources that are likely to limit system performance and noting how they will be used by the system. In particular, the system resources are the CPU, VME data busses, Cougar disk controllers, SCSI strings, and disks. As previously described, a single Sun 4 CPU runs the code that implements the RAID level 5 storage system and controls the four Cougar disk controllers, each of which controls two SCSI strings with three disks each. Each disk controller also has access to its own VME data bus for transferring data to the memory system on the X-Bus board.

Although the general throughput transform for reads is given by Equation 5.25, individual resources can be characterized by simpler transforms. For example, in RAID-II, because the data by-passes the memory system of the Sun 4 CPU, the CPU never touches the data. Thus, for the purposes of analyzing the CPU, we can ignore the per byte overhead and use a simpler throughput transform that is a subset of the full transform:

$$\vec{f_1} = (1, n, -). \tag{5.26}$$

Also, except for the CPU, which breaks array requests into disk requests, none of the other resources see array requests. Thus, for the other resources, we can use a simpler transform that excludes array request overheads:

$$\vec{f_2} = (-, n, nB). \tag{5.27}$$

The above observations make it easier to analyze RAID-II, but it is not absolutely necessary to make them at this stage. As we will see, statistical analysis will lead us to the same conclusions as our intuition.

The next step is to select ranges of system and workload parameters for which a specific resource is likely to be a bottleneck and to analyze the resource within that range. To start off, we will select disks as our first bottleneck resource. Since disks are most likely to be a bottleneck when there are a few of them and the requests to the disks are large, we will analyze them as a bottleneck when the number of disks in the system, N, is equal to two, and the stripe-unit size, B, is greater than or equal to eight. According to Equation 5.24, the following relationship holds under the specified circumstances if the disks are the bottleneck:

$$2/T_{max} = \vec{k}_{disk} \cdot \vec{f_2}. \tag{5.28}$$

Solving for the disk utilization profile, $\vec{k}_{disk}$, using linear regression we have,

$$\vec{k}_{disk} = (0, 14.8\,\mathrm{ms}, 608\,\mathrm{ns}) \tag{5.29}$$

$$\vec{t}_{disk} = (-, 37.9, 58.6) \tag{5.30}$$

The above indicates that the average per disk overhead is 14.8 ms, and the average time to transfer a byte is 608 ns. That is, the disk has an average positioning time of 14.8 msand a sustained transfer rate of 1.64 MB/s. The entries for $\vec{t}_{disk}$, are the corresponding t-statistics discussed in Section 3.4.3. The t-statistics are obtained by dividing each element of $\vec{k}_{disk}$ by its corresponding standard deviation and is a measure of the statistical significance of each element of the utilization profile. The above indicates that both terms of the utilization profile are highly significant.

At this point, it is reasonable to ask what would have happened if we had not made our simplifying observations and had used $(1, n, nB)$ as our throughput transform instead of $(-, n, nB)$. The following illustrate the results.

$$\vec{k}_{disk} = (1.1\,\mathrm{ms}, 14.2\,\mathrm{ms}, 608\,\mathrm{ns}) \tag{5.31}$$

$$\vec{t}_{disk} = (1.0, 18.0, 58.6) \tag{5.32}$$

The t-statistic for the per request overhead is much smaller than the other t-statistics. Examination of the correlation matrix for the utilization profiles also indicates that there is a strong correlation between the per array overhead and the per disk overhead. Thus, it may be desirable to drop the per request overhead as our intuition indicated. This decision cannot be made, however, until the effect of dropping the per request overhead on the overall accuracy of the linear regression is known. To do this, we must examine the error metrics discussed in Section 3.4.4.

$\vec{f}$	R^2	$1-R^2$	*max err*	*90 % err*
BEST	0.9964	0.0036	0.093	0.075
$(1, n, nB)$	0.9948	0.0052	0.129	0.109
$(-, n, nB)$	0.9946	0.0054	0.152	0.116

The above table tabulates the errors in using the empirically best throughput transform, $(1, n, nB)$, and $(-, n, nB)$, respectively. The empirically best throughput transform is the throughput transform from the infinite set of all throughput transforms that maximizes the coefficient of determination, R^2. The best throughput transform takes into account all factors, n, B, and N, and all interactions among the factors. As can be seen, there is only a small difference in accuracy between $(1, n, nB)$ and $(-, n, nB)$, and both compare favorably with the best throughput transform. Thus, our earlier observation that disks do not see array requests is justified.

The reader should note that the results presented so far depend on the subset of the design set, $N = 2$ and $B >= 8$, we selected for analysis. Other subsets should yield equivalent results as long as the disks are the bottleneck but this is by no means guaranteed. In practice, our first choice of the subset is rarely the best choice and it is only by iterating through the analysis process that we gradually come to recognize the proper subsets for which each resource is the bottleneck. By applying the above analysis procedure to the other subsets, we can analyze each bottleneck resource in turn. When the response at all bottleneck design points are accurately predicted by the utilization profile for an analyzed resource, the analysis is complete.

Figure 5.2 summarizes the analysis for each bottleneck resource, including the disks, for the read access mode. In the case of the CPU, eliminating an element from the full throughput transform actually results in an improvement of the *90% err* metric from 0.062 to 0.059. This occasionally occurs when the eliminated element is statistically insignificant. Note, however, that using a simpler throughput transform always results in a worse coefficient of determination or R^2. This is because linear regression maximizes R^2 and thus having more elements can only improve the fit. In all cases, the use of a simpler throughput transform results in errors that are comparable to the empirically best throughput transforms.

Figure 5.3 summarizes the utilization profiles of the bottleneck resources, uses the profiles to create an upper bound, $\hat{T}_{max}$ for system throughput, and compares the bound with the measured bounds and the best empirical bound. As can be seen, the bound based on utilization profiles, $\hat{T}_{max}$, compares favorably with the best empirical bound with respect to the *max err* and *90 % err* metrics although the R^2 for $\hat{T}_{max}$ is significantly worse than for the best empirical bound. The graph in Figure 5.3 plots the predicted maximum throughput in MB/s and compares it to the range of measured maximum throughputs for $N = 24$. In the future, we will primarily rely on such figures and corresponding commentaries on the analysis to succinctly present our results.

5.5.2 Analysis of Read-Modify-Writes

This section defines and calculates the utilization profiles for the read-modify-write access mode on RAID-II. As described in Section 2.3, the read-modify-write access mode is serviced by reading the old data and old parity, xoring them with the new data to generate the new parity, and then writing the new data and new parity to disk.

For many of the resources in the system, we can assume that the relevant overheads are the per array request overhead, per read-modify-write disk overhead, and the per read-modify-write byte overhead. We specify the overheads in terms of read-modify-write accesses rather than simply disk or byte accesses because read-modify-write accesses can

Resource: CPU.
Design Subset: $N = 24$ and $B \leq 16\,\mathrm{KB}$.

$\vec{f}$	$\vec{k}$	$\vec{t}$ (t-stat for $\vec{k}$)
$(1, n, nB)$	(8.5 ms, 910 us, 1.3 ns)	(138, 177, 3.3)
$(1, n, -)$	(8.5 ms, 916 us, 0)	(137, 202, −)

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9970	0.0030	0.071	0.029
$(1, n, nB)$	0.9892	0.0108	0.161	0.062
$(1, n, -)$	0.9890	0.0110	0.158	0.059

Resource: Disks.
Design Subset: $N = 2$ and $B \geq 8\,\mathrm{KB}$.

$\vec{f}$	$\vec{k}$	$\vec{t}$ (t-stat for $\vec{k}$)
$(1, n, nB)$	(1.1 ms, 14.2 ms, 608 ns)	(1.0, 18.0, 58.6)
$(-, n, nB)$	(0, 14.8 ms, 608 ns)	(−, 37.9, 58.6)

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9964	0.0036	0.093	0.075
$(1, n, nB)$	0.9948	0.0052	0.129	0.109
$(-, n, nB)$	0.9946	0.0054	0.152	0.116

Resource: SCSI Strings.
Design Subset: $N = 24$, $B \geq 32\,\mathrm{KB}$ and $n \geq 12$.

$\vec{f}$	$\vec{k}$	$\vec{t}$ (t-stat for $\vec{k}$)
$(1, n, nB)$	(11.0 ms, 2.5 ms, 402 ns)	(2.1, 7.6, 114)
$(-, n, nB)$	(0, 3.1 ms, 402 ns)	(−, 16.7, 112)

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9958	0.0042	0.066	0.038
$(1, n, nB)$	0.9948	0.0052	0.070	0.042
$(-, n, nB)$	0.9945	0.0055	0.072	0.050

Figure 5.2: Analysis of Reads for each Resource.

$$\vec{f} = (1, n, nB)$$

Resource	$\vec{k}$
CPU	$(8.5\,\text{ms}, 916\,\text{us}, 0)$
Disk	$(0, 14.8\,\text{ms}, 608\,\text{ns})$
String	$(0, 3.1\,\text{ms}, 402\,\text{ns})$

$$1/\hat{T}_{max} = \max(\vec{k}_{cpu} \cdot \vec{f}, \vec{k}_{disk} \cdot \vec{f}/N, \vec{k}_{string} \cdot \vec{f}/8)$$

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9951	0.0049	0.243	0.079
$1/\hat{T}_{max}$	0.9654	0.0346	0.495	0.153

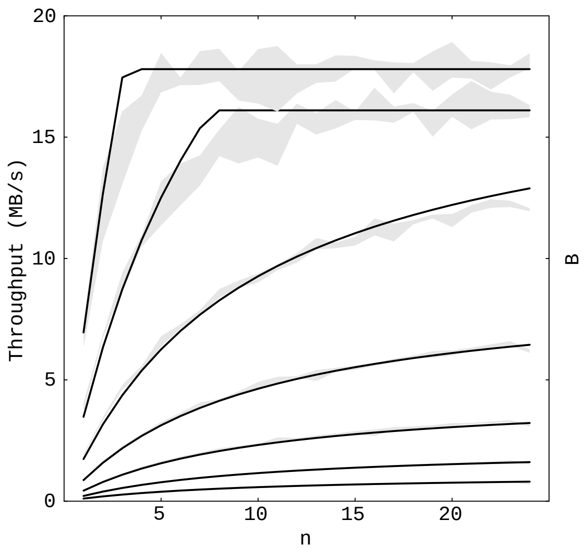


Figure 5.3: Final Analysis of Read. The graph plots the maximum throughput predicted by $\hat{T}_{max}$ in MB/s and compares it to the range of measured maximum throughputs for $N = 24$ and various values of B. From bottom to top, each line represents the predicted maximum throughput for $B = 1, 2, 4, 8, 16, 32, 64$, respectively. For the smaller values of B, the measured ranges are so small that they overlap with the prediction lines.

behave differently from independent disk or byte accesses. With disks, for example, a write access immediately following a read to the same sectors may have to wait a full rotational delay rather than the average of half a rotational delay. Additionally, writes on most disks are slightly slower than reads because the disks require that the disk head be more precisely positioned before writing data. The corresponding throughput transform is

$$\vec{f'} = (1, n+1, (n+1)B). \tag{5.33}$$

That is, for each read-modify-write array access, n data disks plus 1 parity disk are read and written, and correspondingly, $(n+1)B$ bytes are read and written.

Unfortunately, some resources are not accurately characterized by $\vec{f'}$. The xor engine, for example, which generates the new parity for each read-modify-write array request, is better characterized with the following overheads: per array request, per stripe-unit xored, per parity stripe-unit generated, per byte xored, and per parity byte generated. The corresponding throughput transform is

$$\vec{f''} = (1, 2n+1, 1, (2n+1)B, B). \tag{5.34}$$

Of course, we can use different throughput transforms to analyze each resource, but there is a more elegant solution. We can use a more general throughput transform of the form

$$\vec{f} = (1, n, B, nB) \tag{5.35}$$

by showing that using $\vec{f}$ is mathematically equivalent to using $\vec{f}'$ or $\vec{f}''$. That is,

$$\begin{aligned}
\vec{k} \cdot \vec{f} &= \vec{k}' \cdot \vec{f}' \\
k_1 + k_2 n + k_3 B + k_4 n B &= k_1' + k_2'(n+1) + k_3'(n+1)B \\
k_1 + k_2 n + k_3 B + k_4 n B &= (k_1' + k_2') + k_2' n + k_3' B + k_3' n B
\end{aligned}$$

Thus, $\vec{f}$ is at least as general as $\vec{f}'$ and we can easily convert from one set of utilization profiles to another by using the following equations:

$$\begin{aligned}
k_1 &= k_1' + k_2' \\
k_2 &= k_2' \\
k_3 &= k_3' \\
k_4 &= k_3'
\end{aligned}$$

Although the two sets of throughput transforms are mathematically equivalent, they are not statistically equivalent. Note that $\vec{f}$ has four elements while $\vec{f}'$ has only three. This additional degree of freedom is necessary in modeling resources such as the xor unit but is wasted on resources such as disks, which constrain some of the relationships. In fact, using the more general throughput transform for modeling disks has the detrimental effect that it produces larger variations in the computed utilization profiles than using the less general throughput transform. In cases where the design set is sufficiently large and accurate, the additional variation can be insignificant, but otherwise, the additional variation can be detrimental to the accuracy of the regression. In our analysis, the additional variation is sometimes unacceptable. In such cases, we will use the less general throughput transform in analyzing the resources, but convert them to the more general basis when summarizing the results. This technique should not come as a complete surprise to the reader. It has already been used trivially in the section analyzing the performance of the read access mode, where we excluded the per array request overhead in analyzing resources such as disks that we knew could not possibly see array requests.

Figure 5.4 summarizes the analysis for each bottleneck resource for the read-modify-write access mode. In the case of the CPU, it is clear from the t-statistics that only the terms 1 and n are significant. For the disks and SCSI strings, however, all the t-statistics corresponding to the throughput transform $(1, n, B, nB)$ are relatively small. This indicates that the data is not "strong" enough to accurately estimate the utilization profiles for the given throughput transform. To get more accurate estimates, we would have to perform additional experiments or more accurate experiments. In this case, we know that the throughput transform $(n+1, (n+1)B)$ is sufficient, as the corresponding t-statistics and error metrics illustrate. We can now use the utilization profiles for $(n+1, (n+1)B)$ to accurately calculate the utilization profiles for $(1, n, B, nB)$. Figure 5.5 summarizes the utilization profiles of the bottleneck resources. The utilization profiles for the disks and SCSI strings are calculated indirectly from regression on a simpler throughput transform.

5.5.3 Analysis of Reconstruct-Writes

This section defines and calculates the utilization profiles for the reconstruct–write access mode on RAID-II. As described in Section 2.3, the reconstruct-write access mode is serviced by reading the part of the parity stripe that is not being written, xoring it with

Resource: CPU.
Design Subset: $N = 24$ and $B \leq 8\,\mathrm{KB}$.

$\vec{f}$	$\vec{k}$	$\vec{t}$ (t-stat for $\vec{k}$)
$(1, n, B, nB)$	$(11.7\,\mathrm{ms}, 1.9\,\mathrm{ms}, -15.4\,\mathrm{ns}, 13.4\,\mathrm{ns})$	$(72.4, 82.0, -0.45, 2.7)$
$(1, n, -, -)$	$(11.6\,\mathrm{ms}, 2.0\,\mathrm{ms}, 0, 0)$	$(117, 136, -, -)$

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9950	0.0050	0.078	0.029
$(1, n, B, nB)$	0.9919	0.0081	0.099	0.037
$(1, n, -, -)$	0.9907	0.0093	0.088	0.038

Resource: Disks.
Design Subset: $N = 8$, $B \geq 16\,\mathrm{KB}$ and $n \geq 2$.

$\vec{f}$	$\vec{k}$	$\vec{t}$ (t-stat for $\vec{k}$)
$(1, n, B, nB)$	$(70.2\,\mathrm{ms}, 15.4\,\mathrm{ms}, 1.5\,\mathrm{us}, 1.3\,\mathrm{us})$	$(2.8, 1.5, 2.6, 5.5)$
$(n+1, (n+1)B)$	$(30.7\,\mathrm{ms}, 1.3\,\mathrm{us})$	$(17.4, 32.7)$

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9900	0.0100	0.070	0.061
$(1, n, B, nB)$	0.9895	0.0105	0.079	0.066
$(n+1, (n+1)B)$	0.9817	0.0183	0.122	0.084

Resource: SCSI Strings.
Design Subset: $N = 24$, $B \geq 32\,\mathrm{KB}$ and $n \geq 6$.

$\vec{f}$	$\vec{k}$	$\vec{t}$ (t-stat for $\vec{k}$)
$(1, n, B, nB)$	$(193\,\mathrm{ms}, -8.8\,\mathrm{ms}, -2.6\,\mathrm{us}, 1.0\,\mathrm{us})$	$(5.1, -2.0, -3.6, 12.1)$
$(n+1, (n+1)B)$	$(11.8\,\mathrm{ms}, 642\,\mathrm{ns})$	$(12.7, 35.9)$

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9864	0.0136	0.099	0.062
$(1, n, B, nB)$	0.9846	0.0154	0.135	0.059
$(n+1, (n+1)B)$	0.9763	0.0237	0.158	0.096

Figure 5.4: Analysis of Read-Modify-Write for each Resource.

$$\vec{f} = (1, n, B, nB)$$

Resource	$\vec{k}$
CPU	(11.6 ms, 2.0 ms, 0, 0)
Disk	(30.7 ms, 30.7 ms, 1.3 us, 1.3 us)
String	(11.8 ms, 11.8 ms, 642 ns, 642 ns)

$$1/\hat{T}_{max} = \max(\vec{k}_{cpu} \cdot \vec{f}, \vec{k}_{disk} \cdot \vec{f}/N, \vec{k}_{string} \cdot \vec{f}/8)$$

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9909	0.0091	0.327	0.067
$1/\hat{T}_{max}$	0.9440	0.0560	0.454	0.177

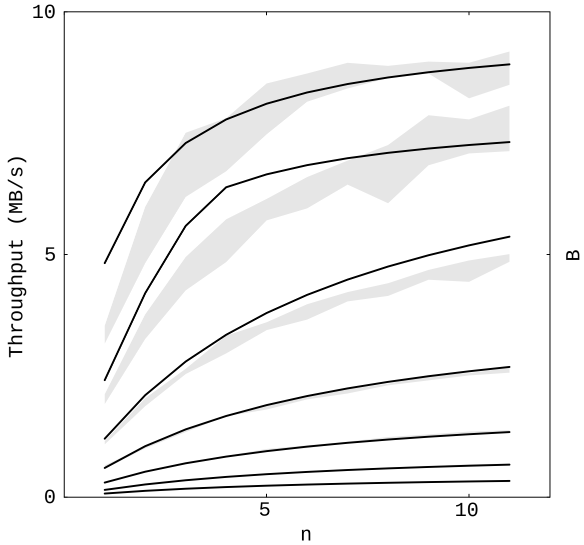


Figure 5.5: Final Analysis of Read-Modify-Write. The graph plots the maximum throughput predicted by $\hat{T}_{max}$ in MB/s and compares it to the range of measured maximum throughputs for $N = 24$ and various values of B. From bottom to top, each line represents the predicted maximum throughput for $B = 1, 2, 4, 8, 16, 32, 64$, respectively. For the smaller values of B, the measured ranges are so small that they overlap with the prediction lines.

the new data to regenerate the new parity, and then writing the new data and new parity to disk.

For many of the resources in the system, we can assume that the relevant overheads are the per array request overhead, per write disk overhead, per read disk overhead, per write byte overhead and per read byte overhead. The corresponding throughput transform is

$$\vec{f'} = (1, N - n - 1, n + 1, (N - n - 1)B, (n + 1)B). \tag{5.36}$$

That is, for each reconstruct-write array access, $N - n - 1$ disks must be read, n data disks plus 1 parity disk must be written, and the corresponding numbers of bytes must be read and written. If the write and read overheads are similar, the throughput transform can be simplified to $(1, N, NB)$ corresponding to the per array request overhead, per disk overhead and per byte overhead respectively.

As was the case for read-modify-writes, the xor engine's overheads are different and consists of overheads per array request, per stripe-unit xored, per parity stripe-unit generated, per byte xored, per parity byte generated.

$$\vec{f''} = (1, N - 1, 1, (N - 1)B, B). \tag{5.37}$$

The above discussion suggests the use of $(1, n, N, B, nB, NB)$ as a general throughput transform. However, preliminary analysis of the data suggests that n is an insignificant factor—the best basis that excludes n is only slightly less accurate than the best basis that includes it. Thus, we will use the following simpler throughput transform:

$$\vec{f} = (1, N, B, NB). \tag{5.38}$$

Figure 5.6 summarizes the analysis for each bottleneck resource for the reconstruct-write access mode. Since the CPU does not explicitly transfer bytes of data, the t-statistics for the CPU indicate that the overheads per byte overheads are statistically insignificant. Analogously, since the disks and SCSI strings do not see array requests, the t-statistics for those resources indicate that the fixed, or per array request overheads, are statistically insignificant. The factor B is statistically insignificant for all three resources. Figure 5.7 summarizes the utilization profiles of the bottleneck resources.

5.6 Applications of Utilization Profiles

The previous sections have used utilization profiles to characterize the performance of RAID-II for the read, read-modify-write and reconstruct-write access modes. To illustrate the usefulness of utilization profiles, this section will use the utilization profiles derived in the earlier sections to answer the following performance-oriented questions:

- What is the maximum throughput at a given system configuration and workload?
- What is the bottleneck resource at a given system configuration and workload?
- Given that the disks are the bottleneck, what is the effect of increasing the number of disks in the system?

Resource: CPU.
Design Subset: $N \geq 12$ and $B \leq 8\,\mathrm{KB}$.

$\vec{f}$	$\vec{k}$	$\vec{t}$ (t-stat for $\vec{k}$)
$(1, N, B, NB)$	$(12.4\,\mathrm{ms}, 825\,\mathrm{us}, -119\,\mathrm{ns}, 14.9\,\mathrm{ns})$	$(33.9, 41.7, -1.5, 3.6)$
$(1, N, -, -)$	$(12.0\,\mathrm{ms}, 883\,\mathrm{us}, 0, 0)$	$(53.8, 73.5, -, -)$

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9116	0.0884	0.144	0.075
$(1, N, B, NB)$	0.8737	0.1263	0.188	0.091
$(1, N, -, -)$	0.8624	0.1376	0.193	0.092

Resource: Disks.
Design Subset: $N \leq 3$.

$\vec{f}$	$\vec{k}$	$\vec{t}$ (t-stat for $\vec{k}$)
$(1, N, B, NB)$	$(-2.2\,\mathrm{ms}, 15.7\,\mathrm{ms}, 28\,\mathrm{ns}, 591\,\mathrm{ns})$	$(-1.0, 19.0, 0.38, 20.4)$
$(-, N, -, NB)$	$(0, 14.9\,\mathrm{ms}, 0, 602\,\mathrm{us})$	$(-, 92.1, -, 107)$

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9930	0.0070	0.427	0.051
$(1, N, B, NB)$	0.9908	0.0092	0.546	0.045
$(-, N, -, NB)$	0.9907	0.0093	0.553	0.049

Resource: SCSI Strings.
Design Subset: $N \geq 20$ and $B \geq 32\,\mathrm{KB}$.

$\vec{f}$	$\vec{k}$	$\vec{t}$ (t-stat for $\vec{k}$)
$(1, N, B, NB)$	$(83.8\,\mathrm{ms}, 766\,\mathrm{us}, 6.7\,\mathrm{us}, 45.6\,\mathrm{ns})$	$(2.4, 0.48, 9.7, 1.5)$
$(-, N, -, NB)$	$(0, 4.5\,\mathrm{ms}, 0, 346\,\mathrm{ns})$	$(-, 12.3, -, 48.4)$

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9909	0.0091	0.129	0.043
$(1, N, B, NB)$	0.9833	0.0167	0.141	0.057
$(-, N, -, NB)$	0.8910	0.1090	0.231	0.126

Figure 5.6: Analysis of Reconstruct-Write for each Resource.

$$\vec{f} = (1, N, B, NB)$$

Resource	$\vec{k}$
CPU	$(12.0\,\text{ms}, 883\,\text{us}, 0, 0)$
Disk	$(0, 14.9\,\text{ms}, 0, 602\,\text{us})$
String	$(0, 4.5\,\text{ms}, 0, 346\,\text{ns})$

$$1/\hat{T}_{max} = \max(\vec{k}_{cpu} \cdot \vec{f}, \vec{k}_{disk} \cdot \vec{f}/N, \vec{k}_{string} \cdot \vec{f}/8)$$

$\vec{f}$	R^2	$1 - R^2$	*max err*	*90 % err*
BEST	0.9941	0.0059	0.427	0.062
$1/\hat{T}_{max}$	0.9045	0.0955	0.553	0.180

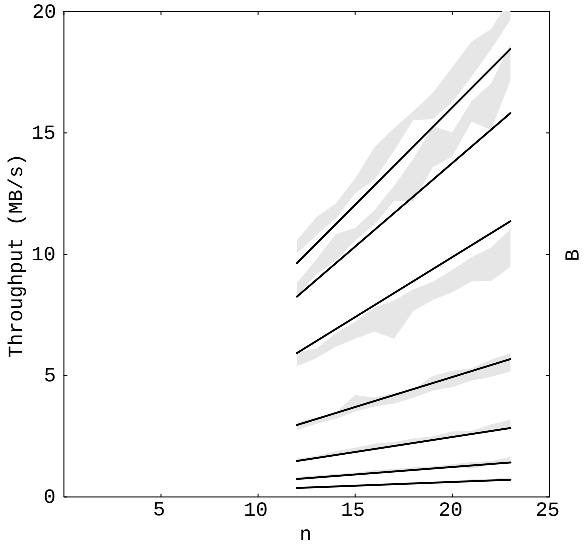


Figure 5.7: Final Analysis of Reconstruct-Write. The graph plots the maximum throughput predicted by $\hat{T}_{max}$ in MB/s and compares it to the range of measured maximum throughputs for $N = 24$ and various values of B. From bottom to top, each line represents the predicted maximum throughput for $B = 1, 2, 4, 8, 16, 32, 64$, respectively. For the smaller values of B, the measured ranges are so small that they overlap with the prediction lines.

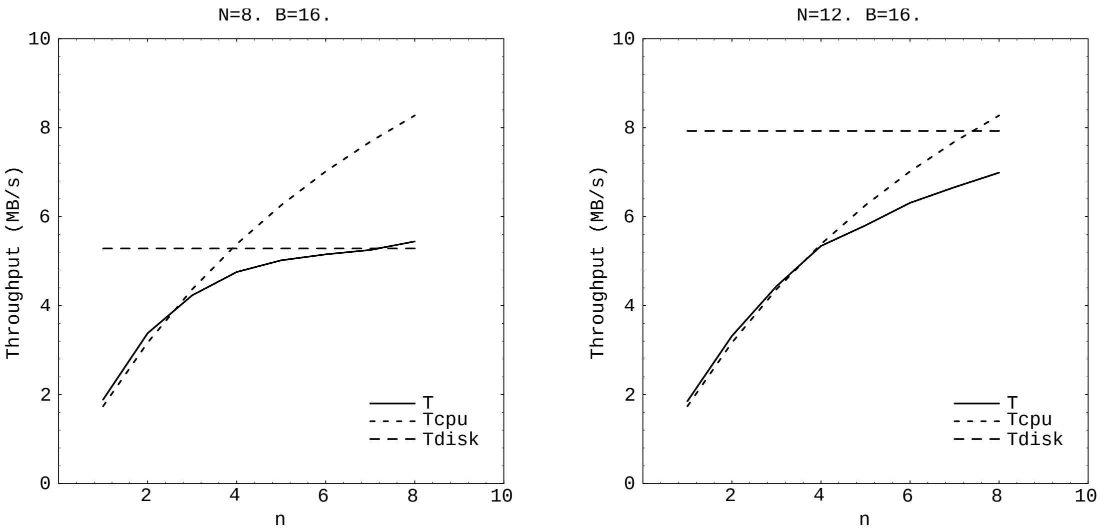


Figure 5.8: Maximum Throughput Envelopes for Reads. The solid line illustrates the measured maximum throughput envelope. The dotted lines illustrates the maximum throughputs predicted by the CPU and disk utilization profiles.

- What throughput can the system support before certain resources become a certain percent utilized?
- How does RAID-II compare with RAID-I?

Figure 5.8 plots the measured maximum throughput envelope and the maximum throughputs predicted by the CPU and disk utilization profiles for the specified values of N, the number of disks in the system, and B, the stripe unit size, as a function of n, the request size in stripe units. The predicted envelope roughly corresponds to the measured envelope. Thus, utilization profiles can be used to determine for each system configuration and workload parameter, the maximum throughput that the system can achieve and the resource that is the bottleneck. We can also use utilization profiles to quickly evaluate changes in the system configuration. For example, the first graph illustrates the throughput envelope when there are eight disks in the system. In this case, the bottleneck throughput when the request size equals six disks is approximately 5.2 MB/s and the bottleneck resource is the disks. The second graph illustrates what would happen if we increase the number of disks to twelve. In such a case, the bottleneck throughput increases to 6.2 MB/s and the bottleneck resource becomes the CPU. Thus, increasing the number of disks from eight to twelve, effectively eliminates the disks as a bottleneck for the given stripe unit and request size.

We have so far applied utilization profiles for workloads when the system is operating close to its bottleneck throughputs. Utilization profiles, however, can also be used answer questions about the system at non-bottleneck throughputs. For example, suppose that we want to limit the utilization of certain resources, such as disks, to be less than 30 % to ensure that queueing delays do not exceed a certain threshold. What throughput can the system support without violating this criteria? Because, we can use utilization profiles to determine throughput envelopes for arbitrary levels of utilization, this question can be answered much the same way as our previous questions.

In general, utilization profiles mathematically relate the utilization of individual resources in the system to the overall system throughput as a function of the system and workload parameters. They can be used to study the effect of varying the system throughput, utilization or system and workload parameters on either the system throughput or utilization of a given resource in the system.

The throughput transforms are abstract quantities that are independent of a particular system's implementation; it is reasonable for all RAID level 5 implementations to have a per array request overhead, a per disk overhead and a per byte overhead. Thus, we can use the same throughput transform to analyze "similar" systems, where the systems will differ is in the values of their utilization profiles.

Because utilization profiles accurately summarize the performance of a given system over a wide range of system and workload parameters, it is especially convenient to compare the performance of two or more similar system by directly comparing their utilization profiles. As an example, the table below tabulates the CPU and disk utilization profiles of RAID-II and RAID-I, our first disk array prototype, for read requests.

$$\vec{f} = (1, n, nB)$$

Resource	k_{II}	k_I	k_{II}/k_I
CPU	$(8.5\,\mathrm{ms}, 916\,\mathrm{us}, 0)$	$(3.7\,\mathrm{ms}, 1.0\,\mathrm{ms}, 388\,\mathrm{ns})$	$(2.3, 0.92, 0)$
Disk	$(0, 14.8\,\mathrm{ms}, 608\,\mathrm{ns})$	$(0, 25.4\,\mathrm{ms}, 805\,\mathrm{ns})$	$(-, 0.58, 0.76)$

Unlike RAID-II, the CPU utilization profile for RAID-I has a non-zero per byte overhead. Because RAID-I is constructed using completely off-the-shelf parts, all data goes through the host CPU's memory system. Moreover, for each I/O request, the data is copied by the CPU to/from user address space from/to kernel address space. Thus, the per byte overhead, reflects the CPU overhead in copying the data and also contention for memory with I/O devices when the cache is being filled or flushed.

The per array request CPU overhead for RAID-II is 2.3 times higher than that for RAID-I. This difference is primarily an artifact of the different measurement programs that we used to gather statistics. On RAID-I, we used a measurement tool that relied on lightweight processes within a common address space whereas with RAID-II we used heavyweight processes with separate address spaces; we made the change to heavyweight processes to increase generality in the measurement tool since not all UNIX systems support lightweight processes. As can be seen, however, it has the detrimental consequence of significantly increasing the per array request overhead since context switches between heavyweight processes are significantly more expensive. In reality, the per array request overheads for both RAID-II and RAID-I are similar as is their per disk overhead. Thus, the main difference between the CPU utilization profiles of RAID-II versus RAID-I is in the per byte overheads. Since RAID-II's CPU does not manipulate bytes, its per byte overhead is dramatically smaller than RAID-I's per byte overhead. Comparison of the disk utilization profiles tells us is that the disks used for RAID-II both positions and transfers data significantly faster than the disks used for RAID-I.

5.7 Summary

In this section, we have presented an analysis technique that combines empirical performance models with statistical techniques to summarize the performance of disk arrays over a wide range of system and workload parameters. The analysis technique is generalizable to a wide variety of systems, requires minimal information about the system to be analyzed, does not require measurements of internal system resources, and produces results that are compact, intuitive, and easily compared with those of similar systems. We then applied the technique to analyze the performance of RAID-II, our second disk array prototype and used the results of the analysis to answer several performance questions about RAID-II. Finally, we used utilization profiles to compare the performance of RAID-II relative to RAID-I, our first disk array prototype. We showed that such comparisons between similar systems immediately brings to attention the relative strengths and weaknesses of the two systems.

Chapter 6

Summary and Conclusions

In this dissertation, we have presented an analytic performance model for non-redundant disk arrays, applied the analytic model to derive an equation for the optimal size of data striping, developed an analysis technique based on utilization profiles, applied the technique to analyze RAID-II, our second disk array prototype, and used the results of the analysis to determine throughput limits, bottlenecks, and to compare RAID-II versus RAID-I.

In deriving the analytic model, we modeled disk arrays as a closed queueing system consisting of N disks and a fixed number, L, of processes continuously issuing requests of fixed size p. We later extended the model to handle distributions of request sizes. The resulting model predicts the expected utilization of a disk in the modeled system, U, as $\frac{1}{1+\frac{1}{L}(1/\overline{p}-1)}$ where $\overline{p}$ is the average size of requests as a fraction of the number of disks in the disk array. We then derived the expected response time and throughput as a function of utilization. We showed via simulation that the simulated utilization is generally within 10%

of the utilization predicted by the analytic model. We also examined the error introduced by each approximation made in the derivation of the analytic model to better understand the validity of the approximations. Finally, we validated two results of the analytic model, namely that utilization is insensitive to the disk service time distribution and that utilization can be accurately represented by an equation of the form $U = 1/(1 + \frac{1}{L}f(p, N))$ where $f(p, N)$ represents an arbitrary function of p and N.

Next, we used the analytic model to show that the optimal size of data striping, which simultaneously maximizes throughput and minimizes response time, is equal to $\sqrt{\frac{PX(L-1)Z}{N}}$ where P is the average disk positioning time, X is the average disk transfer rate and Z is the request size. We used simulation to validate the optimal stripe unit equation and showed that our results correspond well with those obtained by Chen [7]. An advantage of our analytically derived optimal stripe unit equation versus that of Chen's empirically derived rules-of-thumb is that we explicitly take into account both the number of disk in the disk array and the request size whereas Chen does not.

Due to limitation in the applicability of the analytic model, we next developed an analysis technique based on utilization profiles. The technique combines intuitive performance models with statistical techniques and metrics to summarize the performance of a system over a wide range of system and workload parameters with a relative small set of performance metrics called *utilization profiles*. We showed that the analysis technique is applicable to a wide variety of systems, requires minimal information about the system to be analyzed, does not require measurements of internal system resources, and produces results that are compact, intuitive, and easily compared with those of other similar systems

Finally, we used the analysis technique to examine the performance of RAID-II, our second disk array prototype. We used the results of the analysis to determine the performance limits of RAID-II, the bottleneck resource for a given system configuration and workload, and to compare the performance of RAID-II to RAID-I. We showed that utilization profiles are good in characterizing the overall performance of a system and can be used to characterize the performance effects of changes in the system configuration or workload, making them ideal for performance tuning and capacity planning.

In conclusion, we have found that both analytic models and empirical performance analysis techniques are invaluable in understanding and analyzing the performance of real systems. The analytic models allow the formulation of abstract relationships that are difficult to determine directly from empirical measurements and aid in differentiating important factors from insignificant factors. Analytic models, however, are frequently very limited in their application to real systems. Thus, empirical analysis techniques must be applied to validate the analytic models and also to study aspects of the real system that cannot be modeled analytically. Since accurate empirical studies require the analysis of a large amount of data, methods, such as utilization profiles, for summarizing and characterizing the raw performance measurements are necessary.

There are several areas for future work based on the material presented in this dissertation. First, the analytic model's workload can be extended to handle CPU think time. In such a system, processes, instead of simply issuing I/O requests, would alternate between computation and I/O. Second, the model can be extended to handle write requests in RAID's. It would be particularly interesting to see if the general properties we validated for non-redundant disk arrays, such as the insensitivity of disk utilization to the disk service time distribution, hold for RAID's when servicing writes. Third, the model can be applied to solve other problems in the design and configuration of disk arrays, such as computing price/performance ratios for various disk array and workload parameters, and quantifying tradeoffs between the performance and reliability of disk arrays. Fourth, the analysis technique based on utilization profiles can be used to analyze other types of systems such as networks or time sharing systems. This would verify that utilization profiles is a general-purpose technique and is applicable to a wide variety of systems including disk arrays. Fifth, and finally, utilization profiles can be applied to solve real problems in systems with real users. The examples we have provided in this dissertation are, admittedly, artificial; it would be difficult to assess the practical usefulness of utilization profiles without using it to solve real problems.

Bibliography

[1] Francois Baccelli. Two parallel queues created by arrivals with two demands. Technical Report 426, INRIA—Rocquencourt France, 1985.

[2] Francois Baccelli, Armand M. Makowski, and Adam Shwartz. The fork-join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds. In *Adv. Appl. Prob.*, volume 21, pages 629–660, September 1989.

[3] Anupam Bhide and Daniel Dias. Raid architectures for oltp. Technical Report RC 17879 (78489), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10598, March 1992.

[4] Dina Bitton and Jim Gray. Disk shadowing. In *Proc. Very Large Data Bases*, pages 331–338, August 1988.

[5] Peter M. Chen, Garth A. Gibson, Randy H. Katz, and David A. Patterson. An evaluation of redundant arrays of disks using an Amdahl 5890. In *Proc. SIGMETRICS*, pages 74–85, May 1990.

[6] Peter M. Chen, Edward K. Lee, Ann L. Drapeau, Ken Lutz, Ethan L. Miller, Srinivasan Seshan, Ken Shirriff, David A. Patterson, and Randy H. Katz. Performance and design evaluation of the RAID-II storage server. In *International Parallel Processing Symposium*, April 1993.

[7] Peter M. Chen and David A. Patterson. Maximizing performance in a striped disk array. In *Proc. International Symposium on Computer Architecture*, pages 322–331, May 1990.

[8] Shenze Chen and Don Towsley. A queueing analysis of RAID architectures. Technical Report COINS 91-71, University of Massachusetts at Amherst, 1991.

[9] Ann L. Chervenak and Randy H. Katz. Performance of a disk array prototype. In *Proc. SIGMETRICS*, pages 188–197, May 1991.

[10] L. Flatto. Two parallel queues created by arrivals with two demands II. *SIAM J. Appl. Math.*, 45:861–878, October 1985.

[11] L. Flatto and S. Hahn. Two parallel queues created by arrivals with two demands I. *SIAM J. Appl. Math.*, 44:1041–1053, October 1984.

[12] Neal Glover and Trent Dudley. *Practical Error Correction Design for Engineers.* Data Systems Technology, Corp., 1988.

[13] Jim Gray, Bob Horst, and Mark Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *Proc. Very Large Data Bases*, pages 148–161, August 1990.

[14] Philip Heidelberger and Kishor S. Trivedi. Queueing network models for parallel processing with asynchronous tasks. *IEEE Trans. on Computers*, C-31:1099–1109, November 1982.

[15] Philip Heidelberger and Kishor S. Trivedi. Analytic queueing models for programs with internal concurrency. *IEEE Trans. on Computers*, C-32:73–82, January 1983.

[16] Robert Y. Hou, Jai Menon, and Yale N. Patt. Balancing I/O response time and disk rebuild time in a RAID5 disk array. In *Proc. Hawaii International Conference on Computer Systems*, January 1993.

[17] Raj Jain. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, Inc., 1991.

[18] Cheeha Kim and Ashok K. Agrawala. Analysis of the fork-join queue. *IEEE Trans. on Computers*, 38:250–255, February 1989.

[19] Michelle Y. Kim. Synchronized disk interleaving. *IEEE Trans. on Computers*, C-35:978–988, November 1986.

[20] Michelle Y. Kim and Asser N. Tantawi. Asynchronous disk interleaving: Approximating access delays. *IEEE Trans. on Computers*, 40:801–810, July 1991.

[21] Edward K. Lee and Randy H. Katz. An analytic performance model of disk arrays and its application. Technical Report UCB/CSD 91/660, University of California at Berkeley, November 1991.

[22] Edward K. Lee and Randy H. Katz. Performance consequences of parity placement in disk arrays. In *Proc. ASPLOS*, pages 190–199, April 1991.

[23] Edward K. Lee and Randy H. Katz. An analytic performance model of disk arrays. In *Proc. SIGMETRICS*, pages 98–109, May 1993.

[24] Miron Livny, S. Khoshafian, and H. Boral. Multi-disk management algorithms. In *Proc. SIGMETRICS*, pages 69–77, May 1987.

[25] Jai Menon and Jim Kasson. Methods for improved update performance of disk arrays. In *Proc. Hawaii International Conference on Computer Systems*, January 1992.

[26] Jai Menon and Dick Mattson. Performance of disk arrays in transaction processing environments. Research Report RJ 8230, IBM Research Division, 1991.

[27] Jai Menon and Dick Mattson. Comparison of sparing alternatives for disk arrays. In *Proc. International Symposium on Computer Architecture*, pages 318–329, May 1992.

[28] Jai Menon and Dick Mattson. Distributed sparing in disk arrays. In *Proc. IEEE COMPCON*, pages 410–421, February 1992.

[29] Ethan L. Miller. Input/output behavior of supercomputing applications. Technical Report UCB/CSD 91/616, University of California at Berkeley, 1991.

[30] Richard R. Muntz and John C.S. Lui. Performance analysis of disk arrays under failure. Technical Report CSD 900005, University of California at Los Angeles, 1990.

[31] R. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Trans. on Computers*, 37:739–743, June 1988.

[32] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. ACM SIGMOD*, pages 109–116, June 1988.

[33] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface.* Morgan Kaufmann, 1993.

[34] A. L. Narasimha Reddy and Prithviraj Banerjee. An evaluation of multiple-disk I/O systems. *IEEE Trans. on Computers*, 38:1680–1690, December 1989.

[35] A. L. Narasimha Reddy and Prithviraj Banerjee. Gracefully degradable disk arrays. *Proc. International Symposium on Fault-Tolerant Computing*, June 1991.

[36] K. Salem and H. Garcia-Molina. Disk striping. In *Proc. IEEE Data Engineering*, pages 336–342, February 1986.

[37] Daniel Stodolsky and Garth A. Gibson. Parity logging: Overcoming the small write problem in redundant disk arrays. In *Proc. International Symposium on Computer Architecture*, May 1993.

[38] J. Thisquen. Seek time measurements. Technical report, Amdahl Peripheral Products Division, May 1988.

[39] Ronald W. Wolff. *Stochastic Modeling and the Theory of Queues.* Prentice-Hall, 1989.