ABSTRACT

STUDIES OF DISK ARRAYS TOLERATING TWO DISK FAILURES AND A PROPOSAL FOR A HETEROGENEOUS DISK ARRAY

by
Chunqi Han

There has been an explosion in the amount of generated data in the past decade. Online access to these data is made possible by large disk arrays, especially in the RAID (Redundant Array of Independent Disks) paradigm. According to the RAID level a disk array can tolerate one or more disk failures, so that the storage subsystem can continue operating with disk failure(s). RAID5 is a single disk failure tolerant array which dedicates the capacity of one disk to parity information. The content on the failed disk can be reconstructed on demand and written onto a spare disk. However, RAID5 does not provide enough protection for data since the data loss may occur when there is a media failure (unreadable sectors) or a second disk failure during the rebuild process. Due to the high cost of downtime in many applications, two disk failure tolerant arrays, such as RAID6 and EVENODD, have become popular. These schemes use $2/N$ of the capacity of the array for redundant information in order to tolerate two disk failures. RM2 is another scheme that can tolerate two disk failures, with slightly higher redundancy ratio. However, the performance of these two disk failure tolerant RAID schemes is impaired, since there are two check disks to be updated for each write request. Therefore, their performance, especially when there are disk failure(s), is of interest.

In the first part of the dissertation, the operations for the RAID5, RAID6, EVENODD and RM2 schemes are described. A cost model is developed for these RAID schemes by analyzing the operations in various operating modes. This cost model offers a measure of the volume of data being transmitted, and provides a

device-independent comparison of the efficiency of these RAID schemes. Based on this cost model, the maximum throughput of a RAID scheme can be obtained given detailed disk characteristic and RAID configuration. Utilizing M/G/1 queuing model and other favorable modeling assumptions, a queuing analysis to obtain the mean read response time is described. Simulation is used to validate analytic results, as well as to evaluate the RAID systems in analytically intractable cases.

The second part of this dissertation describes a new disk array architecture, namely Heterogeneous Disk Array (HDA). The HDA is motivated by a few observations of the trends in storage technology. The HDA architecture allows a disk array to have two forms of heterogeneity: (1) device heterogeneity, i.e., disks of different types can be incorporated in a single HDA; and (2) RAID level heterogeneity, i.e., various RAID schemes can coexist in the same array. The goal of this architecture is (1) utilizing the extra resource (i.e. bandwidth and capacity) introduced by new disk drives in an automated and efficient way; and (2) using appropriate RAID levels to meet the varying availability requirements for different applications.

In HDA, each new object is associated with an appropriate RAID level and the allocation is carried out in a way to keep disk bandwidth and capacity utilizations balanced. Design considerations for the data structures of HDA metadata are described, followed by the actual design of the data structures and flowcharts for the most frequent operations. Then a data allocation algorithm is described in detail. Finally, the HDA architecture is prototyped based on the DASim simulation toolkit developed at NJIT and simulation results of an HDA with two RAID levels (RAID1 and RAID5) are presented.

STUDIES OF DISK ARRAYS TOLERATING TWO DISK FAILURES AND A PROPOSAL FOR A HETEROGENEOUS DISK ARRAY

by

Chunqi Han

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

May 2004

APPROVAL PAGE

STUDIES OF DISK ARRAYS TOLERATING TWO DISK FAILURES AND A PROPOSAL FOR A HETEROGENEOUS DISK ARRAY

Chunqi Han

Dr. Alexander Thomasian, Dissertation Advisor
Professor of Computer Science, NJIT

Date

Dr. James Calvin, Committee Member
Associate Professor of Computer Science, NJIT

Date

Dr. Joseph Leung, Committee Member
Distinguished Professor of Computer Science, NJIT

Date

Dr. Wojciech Rytter, Committee Member
Professor of Computer Science, NJIT

Date

Dr. Jian Yang, Committee Member
Associate Professor of Industrial and Manufacturing Engineering, NJIT

Date

BIOGRAPHICAL SKETCH

Author:	Chunqi Han
Degree:	Doctor of Philosophy
Date:	May 2004

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2004
- Master of Science in Computer Science,
  Shanghai Jiao Tong University, Shanghai, China, 1998
- Bachelor of Science in Electrical Engineering,
  Shanghai Jiao Tong University, Shanghai, China, 1995

Major: Computer Science

Presentations and Publications:

Alexander Thomasian, Chunqi Han, Gang Fu, and Chang Liu, "A Performance Evaluation Tool for RAID Disk Arrays", *submitted to QEST'04.*

Chunqi Han, Alexander Thomasian and Chang Liu, "Affinity Based Routing in Mirrored Disks with Zoning", *to appear SPECTS'04.*

Gang Fu, Alexander Thomasian, Chunqi Han and Spencer Ng, "Rebuild Strategies for Clustered Redundant Disk Arrays", *to appear in SPECTS'04.*

Gang Fu, Alexander Thomasian, Chunqi Han, and Spencer Ng, "Rebuild Strategies for Redundant Disk Arrays," *Conference on Mass Storage Systems and Technologies '04*, College Park, MD, April 13-16 2004.

Chunqi Han and Alexander Thomasian, "Performance of Two Disk Failure Tolerant Disk Arrays," *SPECTS03*, Montral, August, 2003.

Alexander Thomasian, Junilda Spirollari, Chang Liu, Chunqi Han, Gang Fu, "Mirrored Disk Scheduling," *SPECTS03*, Montral, August, 2003.

R. Boian, A. Sharma, C. Han, G. Burdea, A. Merians, S. Adamovich, M. Recce, M. Tremaine and H. Poizner, "Virtual Reality-Based Post-Stroke Hand Rehabilitation," *Proceedings of Medicine Meets Virtual Reality 2002*, IOS Press, pp. 64-70, Newport Beach, CA, January 23-26 2002.

This work is dedicated to my beloved wife and family

ACKNOWLEDGMENT

I would like to extend my sincere gratitude to my advisor Dr. Alexander Thomasian. His invaluable guidance and encouragement have contributed significantly to the work presented in this dissertation.

I would like a warm thanks to Dr. Joseph Leung, Dr. Wojciech Rytter, Dr. James Calvin and Dr. Jian Yang for their guidance and abundant help throughout this research.

I would like to thank all the members in the Integrated System Lab, Gang Fu, Chang Liu, Yue Li and Lijuan Zhang, for their great help and support through the four years of my research and for the joyful days in the Lab. Special thanks go to Gang Fu for his great work in developing the DASim simulation toolkit.

I will always be indebted to my parents. Without their moral and intellectual guidance through my life, all these would be impossible. Also I wish to thank my brother and parents-in-law for their continuous support and encouragement.

I dedicated this dissertation to my wife, Li Zhang, for her love, understanding, help, and support.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter Page

TABLE OF CONTENTS
(Continued)

Chapter Page

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

This dissertation consists of two parts. The first part is the performance analysis and comparison of the disk arrays that can tolerate two disk failures. The second part describes the Heterogeneous Disk Array architecture and presents simulation results to quantify its performance. This chapter provides the motivation and background information for both studies.

1.1 Motivations for Performance Analysis Study on Double Failure Tolerant Disk Arrays

RAID5 is a popular disk array scheme, which utilizes one parity disk to protect against single disk failure. When a disk fails, the data on the failed disk can be reconstructed by exclusive-ORing the data on the surviving disks that are in the same parity group, and writing the data onto a spare disk. The mean time to data loss (MTTDL) of RAID5 is proportional to the square of mean time between failures (MTBF) of a single disk and inversely proportional to the square of number of disks and the mean time to reconstruct (MTTR) [48]. Such a system is reliable when the number of disk is small and the MTTR is short. However, single disk failure tolerant is not enough in some cases because of the following reasons [9]:

1. The reliability formula in [48] does not take into account the uncorrectable error rates of a hard disk. The uncorrectable error rate for the current state-of-the-art hard drive is 1 error out of 10^{15} bits [39]. For a RAID5 array consisting of twenty one 100 GB disks, the amount of data to be read at the occurrence

of a disk failure is 2000 Gigabytes or 1.6 × 10^{13} bits. The probability of a successful reading all those bits without an error is 98.4%, which means each disk failure will risk a data loss in 1.6% of cases. This may be unacceptable in some applications.

2. Since the data fault can not be detected without the attempt to read the data, the localized faults on dormant data sectors or tracks can not be detected until the rebuild process tries to read the data. Therefore, such a hard to detect localized fault leads to data loss.

3. During the rebuild process, the data are unprotected. A second disk failure before the completion of reconstruction will lead to a large amount of data loss.

The fast-increasing disk capacity exacerbates all three problems: Given the increased number of blocks on disk, more erroneous blocks will be created. Seldomly accessed blocks potentially harbor hidden faults. Large capacity increases rebuild time (MTTR) and make a second disk failure more possible.

The n-disk-failure-tolerant arrays are a solution to this problem, but only $n = 2$ is considered, because it is deemed to be sufficient to make data loss highly unlikely [9]. StorageTek's Iceberg is an early 2DFT product [15], which uses two check disks with P+Q coding (a Reed-Solomon code) [8, 38]. HP's RAID 5DP (double parity) also uses P+Q coding [35]. These two systems are referred to as RAID6 in this dissertation. Two schemes with minimal or low level redundancy: EVENODD [9] and RM2 [47], are also considered in this study. The advantage of these schemes over P+Q coding is that they use only parity and do not require specialized hardware or more lengthy computations required for Reed-Solomon coding. The more recent RDP coding also uses parity only and is similar to EVENODD from viewpoint of disk access [18]. Extensions of both RAID6 and EVENODD methods to three or more redundant disks have been proposed in [3] and [10], respectively. The methods in [30] are not considered because of their higher levels of redundancy.

1.2 Hard Disk Structure and Disk Array Organization

This section describes the structure and organization of modern hard drives as well as disk arrays. Subsequent chapters assume intimate knowledge of disk structures and array layouts.

1.2.1 Structure of Hard Disk Drive

Figure 1.1 shows the structure of a typical hard drive. A disk drive consists of one or more *platters* mounted on a common spindle. Each platter has both sides coated with magnetic material whose polarity changes with strong localized magnetic field. The polarity is retained until the next write occurs. The platters rotate at a fixed speed, which is typically measured as *revolutions per minute – RPM.* For each platter, there is a corresponding *read/write head* mounted at the end of a disk arm. Disk arms are mounted to a common shaft called *actuator.* A small directional current on the actuator motor causes it to move in either direction, and therefore moves the read/write heads inbound or outbound. This movement, together with the rotations of platters, allow the access to the data on each platter. Although there are multiple read/write heads, in most contemporary hard drives, there is only one active head at any time. This is because it is very hard or impossible to position two head precisely on corresponding tracks at the same time due to thermal variations of the disk arm and platters.

Data on disk are organized into *sectors*, *tracks*, and *cylinders.* A *sector* is a fixed amount (almost always 512 bytes) of sequential user data plus a header and

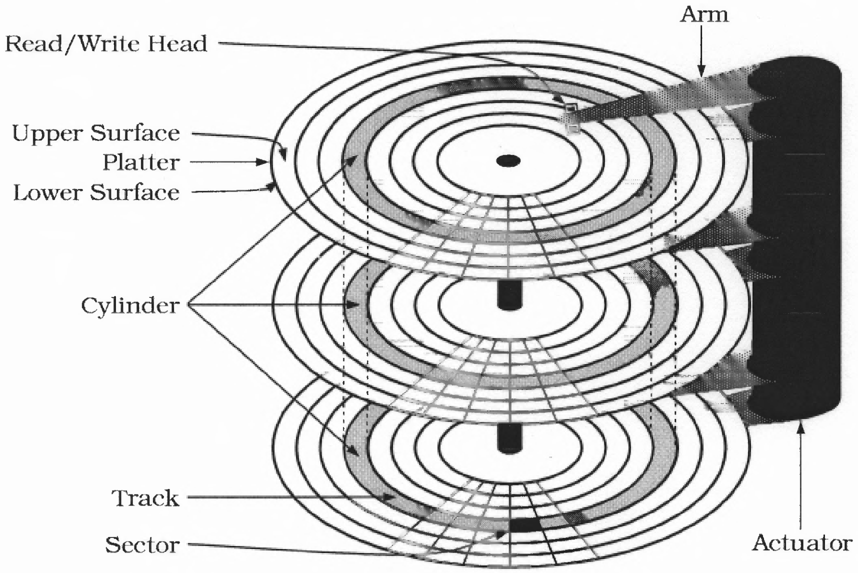


Figure 1.1 The structure of a hard disk. (Source: [59])

trailer. The sector header contains the sector id and clock synchronization information. The sector trailer contains the error correcting code computed over both the header and data. A *track* consists of a set of sectors on a data surface that makes a circle centered at the spindle. Tracks with the same radius constitutes a *cylinder*. All the sectors are numbered sequentially as block addresses and constitute a linear address space to the user.

When a user request arrives, the block address contained in the request is translated into cylinder and track numbers by the firmware of the disk. The actuator then moves the disk arms toward the target cylinder and corresponding read/write head is activated. The time incurred in this movement is called the *seek time*. After the head is put on the right track, the disk waits the first requested sector passes under the read/write head. This waiting time is called *rotational latency* or sometimes *latency* for short. The sum of the seek time and rotational latency is called the *positioning time*, since it is the time required to search for the target sector. After positioning, the constant rotation of the platters makes data sectors pass the read/write head consecutively. The time elapsed for all the requested data sectors passing under the read/write head is called the *transfer time.*

In some cases, a transfer would span two tracks. Therefore, two read/write heads need to be activated one after the other to fulfill the access. The head switching takes a short time ($\approx$ 1 millisecond) and is call the *head switching time.* Consequently, to ensure the data of the next track can be read right after the head switching, the first sector of the next track is positioned from the last sector of the previous track at an angle equal to the rotation speed times head switching time. A *track skew* is then defined to be the number of sectors that takes up this angle. Similarly, when a transfer spans two cylinders, a *cylinder switch time* (which is the seek time of one cylinder plus none-overlapped head switch time) occurs and corresponding *cylinder skew* is defined.

When user requests a whole track's worth of data, the rotational latency can be avoided by starting the reading the data sectors right away, after positioning the read/write head on the track, rather than waiting until the first sector rotates under the head. This policy is called the *zero-latency* operation. For requests that are not a full track, but consist of multiple sectors, the zero-latency operation can also be applied. However, the improvement is not as significant as for full-track accesses.

Since the tracks on the outer cylinders have a greater circumference than those on the inner cylinders, it is natural to put more sectors on a track on outer cylinders. Hence the idea of *zoning.* In a zoned disk, adjacent cylinders are grouped into a zone. The sectors per track remains the same in a zone while it differs between zones. Outer zones have more sectors per track.

At current recording density, a typical $3\frac{1}{2}$ inch disk has about 1000 sectors per track, 2 to 8 tracks per cylinder (i.e. 1 to 4 platters) and about 10^4 cylinders per disk. The revolution speed varies from 5400 to 15000 RPM. Mean seek time varies from 4 to 10 milliseconds. The mean latency is about half the rotation time and hence 2 to 5.6 milliseconds.

1.2.2 Single Failure Tolerant Disk Arrays

In the past decade, the performance of processors has been growing steadily. Their computing power in terms of Million Instruction per Second (MIPS) has been doubling approximately every two years. However, the I/O performance has been far behind the CPU processing power. The overall performance of the computer systems is therefore curbed by the performance of the I/O subsystem, according to Amdahl's law [31].

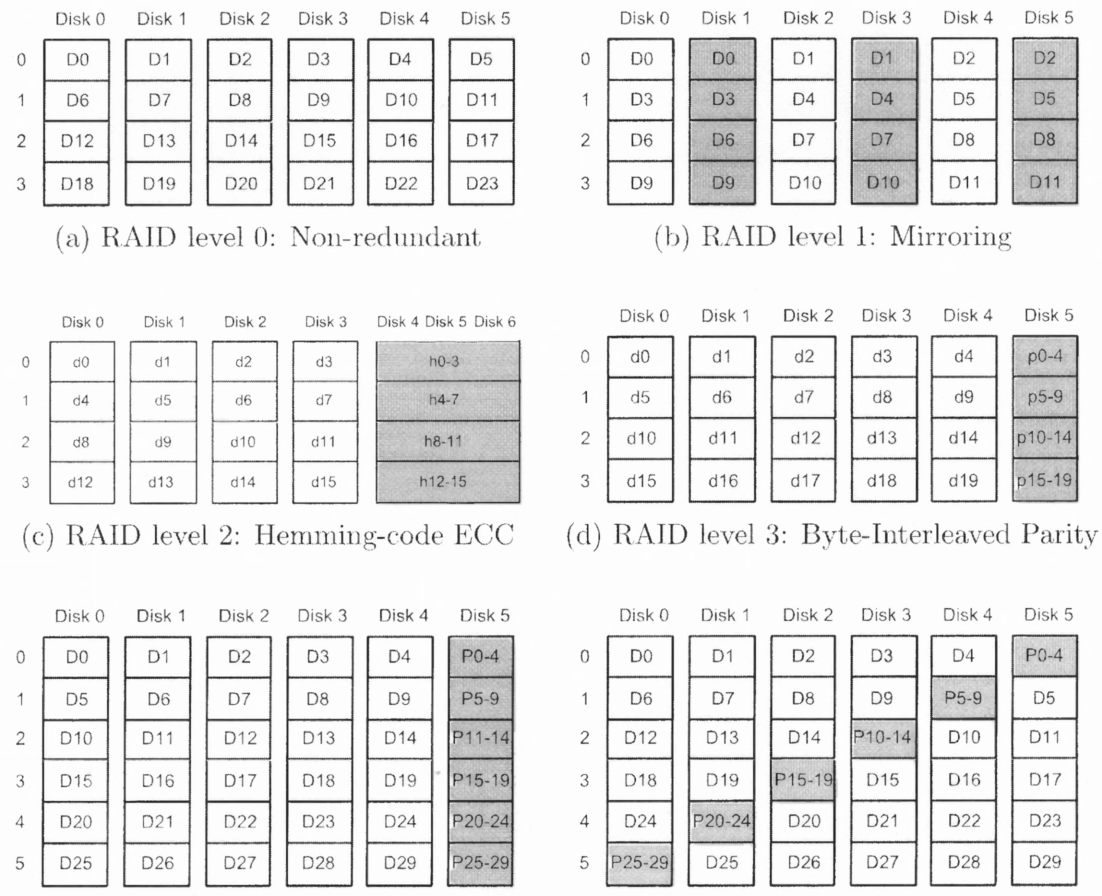
Hard disk drives are still the dominating choice of data storage. Since the mechanical parts (read/write arms, rotating platters) are involved, the accessing speed of the hard disks is bounded by mechanical limits. Thus the idea of multi-programming has been applied to increase the performance of I/O subsystem.

The Redundant Array of Independent Disks (RAID) [48] offer the advantage of fragmenting the total storage space into multiple inexpensive smaller disks, which allows cost-effective solutions. It also benefits from the higher aggregate bandwidth of the component disks, and smaller seek latencies associated with shorter arms.

A typical disk array consists of a bunch of *identical hard disks* connected to an *array controller* through a common parallel bus (e.g. SCSI [7]). Recently serial links (e.g. Fibre Channel) are receiving more attention over parallel connection such as SCSI buses. Serial links require smaller mechanical interface and are expected to be more popular since the form factor of disk are getting smaller [28, 29]. The array controller is connected to a host computer using high-bandwidth links. The responsibility of the array controller is maintaining address mapping and redundant information, controlling individual disks, translating host requests and recovering from disk or link failures. The array controller provides a linear address space to the host. The redundant information is maintained by the disk array controller and is transparent to the user. The mapping of this host side linear address space to individual disk address space is referred to as the *data layout.*

One of the fundamental concepts of RAID disk arrays is *striping* [48, 26, 42]. Striping is to break the linear address space exported by the array controller into smaller blocks. Each block is called a *stripe unit* or *striping unit.* Consecutive stripe units are placed onto different drives, so that the stripe unit is the maximum amount of consecutive data assigned to a single disk. The benefits of striping include automatic load balancing and high bandwidth for large sequential transfers. However, these two benefits do not come together with higher concurrency: a smaller stripe unit size may increase the bandwidth for a single transfer by involving more disks, while at the same time, this reduces the number of concurrent requests the disk array can handle.

Traditional single failure tolerant disk arrays are classified into five types, namely RAID level 1 through 5 [48]. This terminology has gained wide acceptance and is used throughout this dissertation. Although not part of the original RAID classification, RAID level 0 is often used to indicate a non-redundant disk array with striping. Briefly, RAID1 is block interleaved dedicated mirroring. RAID2 is bit or byte-interleaved and using Hamming error correcting code [49]. RAID3 is byte-interleave parity with one disk dedicated to parity. RAID4 is block-interleave parity with one disk dedicated to parity. RAID5 is rotated block-interleaved parity with the parity blocks distributed over all disks. In Figure 1.2, the data and redundancy information organizations for RAID levels 0 through 5 are illustrated. The RAID5 design shown uses *left-symmetric* organization [37], which is first placing the parity stripe units on the diagonal and then placing consecutive data stripe units on consecutive disks. The group of disks that a parity is computed over is called a *parity group.* For the samples show in Figure 1.2, there is only one parity group for each RAID level. It is possible that more than one parity group exist in a RAID. This technique is called *declustering* and the result RAID scheme is called *clustered RAID* [44, 33].



	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D0	D1	D2	D3	D4	D5
1	D6	D7	D8	D9	D10	D11
2	D12	D13	D14	D15	D16	D17
3	D18	D19	D20	D21	D22	D23

(a) RAID level 0: Non-redundant

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D0	D0	D1	D1	D2	D2
1	D3	D3	D4	D4	D5	D5
2	D6	D6	D7	D7	D8	D8
3	D9	D9	D10	D10	D11	D11

(b) RAID level 1: Mirroring

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4 Disk 5 Disk 6
0	d0	d1	d2	d3	h0-3
1	d4	d5	d6	d7	h4-7
2	d8	d9	d10	d11	h8-11
3	d12	d13	d14	d15	h12-15

(c) RAID level 2: Hemming-code ECC

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	d0	d1	d2	d3	d4	p0-4
1	d5	d6	d7	d8	d9	p5-9
2	d10	d11	d12	d13	d14	p10-14
3	d15	d16	d17	d18	d19	p15-19

(d) RAID level 3: Byte-Interleaved Parity

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D0	D1	D2	D3	D4	P0-4
1	D5	D6	D7	D8	D9	P5-9
2	D10	D11	D12	D13	D14	P11-14
3	D15	D16	D17	D18	D19	P15-19
4	D20	D21	D22	D23	D24	P20-24
5	D25	D26	D27	D28	D29	P25-29

(e) RAID level 4: Dedicated Block Parity

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D0	D1	D2	D3	D4	P0-4
1	D6	D7	D8	D9	P5-9	D5
2	D12	D13	D14	P10-14	D10	D11
3	D18	D19	P15-19	D15	D16	D17
4	D24	P20-24	D20	D21	D22	D23
5	P25-29	D25	D26	D27	D28	D29

(f) RAID level 5: Rotated Block Parity

Figure 1.2 Data layout in RAID levels 0 through 5. The shaded blocks are parities. d_i means data are bit or byte interleaved over disks. D_i means data are block interleaved. p_{i-j} means the parity is computed over d_i through d_j, P_{i-j} is defined similarly.

Among those RAID levels, RAID 2 and 4 are of less interest. RAID2 uses Hamming code, which introduces higher redundancy than necessary. In disk arrays, it is easy to check whether a disk is failed or not by sending special commands or checking the ECC codes in the sector trailers. This feature makes a disk an *erasure* channel, to distinguish from an *error* channel, in which the location of an error is unknown. RAID4 differs from RAID3 only in that it is block-interleaved. There is a load imbalance problem for RAID4, since the disk that is dedicated to parity can be overloaded if a large fraction of requests are writes. RAID5 offers a better solution by distributing the parity stripe units over all disks, such that the load is balanced and all disks can contribute to the read throughput. RAID3 is suitable for the special scenario when the disk array is dedicated to a single application and the process demands large amount of data at high bandwidth.

The concept of disk array offers a solution for highly reliable parallel data storage. For single disk tolerant disk arrays, the reliability can be measured in the form of *mean time to data loss(MTTDL)*. A simple expression for the MTTDL for a redundant disk array that can tolerate one disk failure is given in [48]:

$$MTTDL = MTTF_{RAID} = \frac{(MTTF_{disk})^2}{N(G-1)MTTR_{disk}}$$

where N is the total number of disks in the array, G is the number of disks in a RAID group (i.e. a set of disks over which a parity is computed), $MTTF_{disk}$ is the mean time to failure of a component disk, typically 200,000 to 300,000 hours. $MTTR_{disk}$ is the mean time to repair of a component disk, typically a few hours.

1.3 RAID Performance Studies

Since the advent of RAID, there have been numerous performance studies dealing with various RAID performance metrics. Some studies investigate the maximum throughput attainable for various RAID levels and different workloads, such as in [15, 26, 48]. Other studies use mean response time as the main performance metric. The M/M/1 queueing model with Poisson arrivals and exponential service times was used in some early studies [40, 44]. However, since the disk service time can be more accurately modeled with a general distribution, the M/G/1 queuing model was introduced later [59, 65, 43, 36, 64, 17]. Among those, some studies also evaluated RAID5 performance in degraded and rebuild modes [43, 65, 64, 40, 44].

Beside analytical analyses, simulations are often used in the performance study. The simulations take two roles: firstly it is used to validate the accuracy of various approximations in the analytical model, e.g. [65, 64], secondly it is used to solve some complicated problems that are too difficult to solve by queueing theory [52, 13, 34]. Simulations can be driven by a random number generator or a trace collected from a real system. While pseudo random number driven simulations offer more flexibility and control on workloads, results from trace driven simulation are more meaningful and have more credibility, since it makes a closer emulating of real workloads. However, trace files are often hard to obtain, and more importantly, they are hard to be adapted according to various arrival rates and configurations. Since the performance for various arrival rates and configurations need to be investigated, pseudo random number rather than trace driven simulations are used in this dissertation.

The caches have a major effect on disk array performance. It operates as a filter and reduces the disk load by satisfying a fraction of read requests directly. This fraction is called the cache hit ratio. The performance of the RAID controller cache can be evaluated by trace driven simulations [73, 66]. The main target for such simulations is to estimate the cache hit ratio. However, the cache performance depends heavily on the workload characteristic and varies from case to case. As a result, most performance studies on redundant disk array investigate the performance of a disk array without caching. In this dissertation, only throughput provided by disk arms is of interest and the cache effect is not considered for the same reason. The effects of caching will be discussed in more detail in Section 2.2.3.

1.4 Motivations for Heterogeneous Disk Array

The second part of this dissertation is concerned with devising a *self-managed heterogeneous storage system.* The heterogeneity has two aspects: different disk drives and different RAID levels in a *single* disk array. In this section, the motivations for the introduction of heterogeneous disk array are discussed, followed by description of related work in the next section.

The heterogenous disk array is motivated by the observations about several trends in the data storage technology, which are described below.

1.4.1 Growth in Disk Capacity

Rapid improvements in magnetic material and recording technology have lead to an exponential growth in disk capacity. Since '90s, the areal density of the hard disk drives has been increasing at 60% per year, which is even faster than before [29, 28]. Figure 1.3 shows the trend of the price per megabyte for both magnetic hard disks and semiconductor storages (i.e. DRAM and flash memory) in the past two decades. Note that the Y axis is logarithmic, which means the capacity of a hard drive has been increasing exponentially, since the price per disk remains steady. In Table 1.1, the typical seek time and bandwidth for a hard disk in different years are shown. The data sources are [4], [22] and [39] for the disk models before year 1994, year 1999 and year 2003, respectively.

However, this rapid increase in disk capacity has the following implications. Consider the scenario: one of the disks fails after operating for several years, but the same model is not readily available or cost effective when compared with more modern disks. Therefore, it is replaced with a new disk drive. The new disk drive is larger in capacity, and usually has a lower seek time, higher rotational speed and

Table 1.1 The Trend in Disk Bandwidth.

Characteristic	1987	1990	1994	1999	2003
Seek time(ms)	16.7	11.5	8(r)/9(w)	5	3.2
RPM	3600	5400	7200	10000	15000
Rotational Latency(ms)	8.3	5.5	4.2	3.0	2.0
Transfer rate (MB/s)	2.5	3–4.4	6–9	18-22.5	75
8KB transfer time (ms)	28.3	19.1	13.1	9.6	5.3
1MB transfer time (ms)	425	244	123	62	18.6

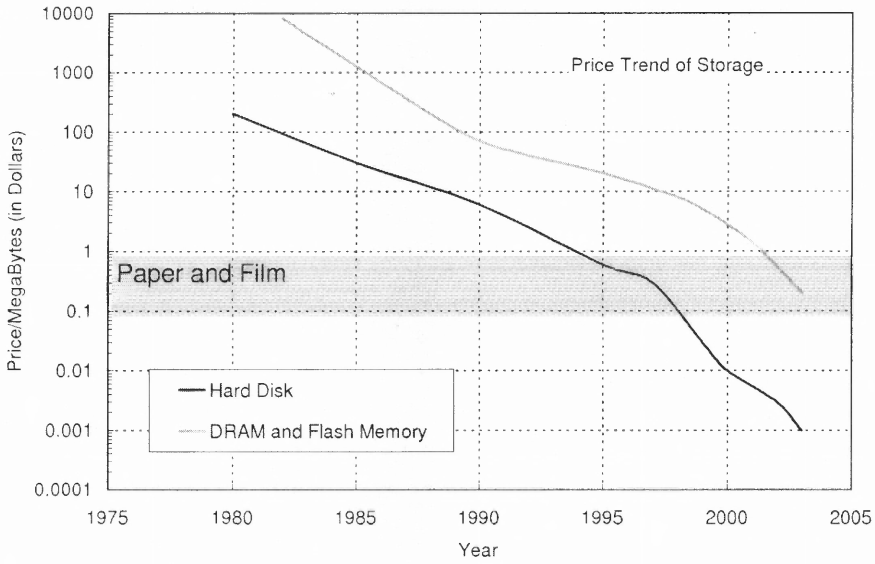


Figure 1.3 The trend of storage price in the last two decades. The shaded area is the price range for paper and film media. This figure is a simplified reproduction of the figure in [29].

transfer rate. In a traditional disk array configuration, only part of the capacity that equals that of the old disk drive is used. The extra capacity and bandwidth are simply wasted [19, 20].

A similar situation arises when a disk array runs out of storage space after a few years due to the growth in the data or introduction of new applications. Since the identical disk model is no longer manufactured at that time, one can either add new disks to the disk array (if it allows such an expansion) or buy a new disk array altogether. However, in the first approach a large fraction of capacity (perhaps more than 50%) and bandwidth of the new disks is wasted. The second approach might be too costly.

A storage system would be more attractive if it allows the user to add new disk models when there is a shortage of space or bandwidth. The extra capacity and bandwidth should be fully utilized, so that the user can benefit from the fast improving technology and enjoy a higher performance-price ratio. This means the storage system should consist of heterogeneous disks.

1.4.2 Complex Application Requirements

A disk array is usually shared by many applications. Each application accesses one or multiple datasets. Each dataset can have a number of requirements. The requirements include, but are not limited to, capacity, throughput, and reliability.

As far as reliability and performance are concerned, there is no single RAID level that can meet all of the requirements. Each RAID level has different characteristics and performs well only for a relatively narrow range of workloads. It is desirable, for example, to use RAID6 rather than RAID5 for critical data since RAID6 provides more protection over the data, RAID1 is suitable for hot data since the throughput of mirrored data doubles that of a single disk for read requests, RAID0 can be used for high volume temporary data since it has the lowest storage overhead and it does not suffer the small write penalty.

A general purpose storage system should be able to combine different RAID levels. i.e., RAID0/1/5/6 etc., to meet the requirements of individual stores. In other words, the storage system should be heterogeneous in terms of the RAID level.

1.4.3 Management Cost

In traditional storage systems, many configuration parameters must be specified for each new dataset. To achieve the right balance between cost, availability, and performance is a challenging task. Storage administrators are skilled and highly paid personnel since they need to make decisions which impact performance, based on inadequate information. Recent studies have indicated that the cost of large storage system, over the course of their lifetime is dominated by storage management costs [1, 46].

Even the array itself is not cheap. It is common for an enterprise class storage system to cost more than one million dollars. Because of the rules of thumb used by the administrators, and the not-so-good scalability of most storage system, the resulting systems are often over-provisioned, which makes them unnecessarily expensive.

An ideal storage should be self-managed. It should be able to choose the right configuration for different datasets based on their characteristics. It should be painlessly expandable, so that there is no need to provide too much spare storage for future use.

1.4.4 Summary

As the previous discussion shows, the hard disk technology has been making great strides in the past two decades and will continue to do so. Disk capacities have been doubling every 18 months, and their bandwidths have been increasing steadily as well. This means the price per megabyte decreases at a fast pace and it is more cost-effective to use more recent disk products to replace a failed disk or to expand disk array capacity. Moreover, in some circumstances, a failed disk has to be replaced by a newer model because the same old model is not available. Consequently, this mixture of disk model leads to heterogeneity in physical devices.

However, this heterogeneity introduces a few problems. In traditional disk array, e.g. RAID1/5/6, all disks are identical. If a new disk with larger capacity is introduced, only part of the capacity that is equal to the old model is utilized. The rest of capacity, which may exceed 50%, is wasted. As a result, a disk array scheme that can utilize the extra capacity is attractive.

Besides capacity, the disk heterogeneity has another side effect – on the bandwidth side.

In this dissertation, the bandwidth is defined as the number of small accesses a disk can handle in a second. A small access is a read or write to a 4KB block. Although the disk bandwidth is increasing steadily as shown in Table 1.1, the improvement is not so fast such that it can not catch up with the pace of disk capacity. This is because there are mechanical components (e.g. disk arms, platters), which have mass and momentum that make them difficult to move fast. Also the growing recording density leads to more tracks per inch (TPI) but makes it more difficult to locate a track.

As an example, in Table 1.2, three disks are given with their capacity and bandwidth. It can be observed that the bandwidth/capacity ratio is decreasing very

Table 1.2 The Trend of Bandwidth-Capacity Ratio

Year	1997	1999	2003
Model	Seagate barracuda	Seagate Atlas 10K	Maxtor 15K
Capacity (GB)	2.0	9.1	73
Bandwidth (4KB accesses per second)	74.5	116.8	188
Bandwidth/Capacity Ratio	37.25	12.84	2.58

fast over the years. In other words, if the data of the same genre are put into two disks produced in year 1997 and 2003, the bottleneck of the older model tends to be at the capacity side, while the bottleneck of the new model tends to be at the bandwidth side. Therefore, to alleviate the bottlenecks on both disks, it is natural to put data with higher access rate/MB on the smaller drive while putting the data with lower access rate/MB on the larger drive.

However, this requires knowledge of access rate *at the time of* allocation, which is *before* the data are actually accessed. This kind of knowledge is hard to obtain, but it is not hopeless. Most applications have their pattern of accessing data. For example, the files created by a logger are important but rarely accessed. Files created by email clients are accessed periodically and should be moderately protected. With the help of operating system, these pattern can be discovered and forwarded to the storage system to help making proper allocation decisions.

Inevitably, these access rate estimations are approximate. The inaccuracy may lead to an *unbalanced load* problem, which means some disks are over-utilized while some others are under-utilized in terms of either bandwidth or capacity. To further optimize the placement of data, it is desirable to have a scheme that can balance the load automatically based on the information gathered by continuously monitoring the system performance.

To sum up, the Heterogeneous Disk Array architecture to be described in this dissertation can provide several features that are not available in traditional disk arrays:

1. It consists of different disk types.
2. Multiple RAID levels coexist in a single physical array.
3. Utilizes the available storage capacity to the maximum extent.
4. Utilizes the available disk bandwidth to the maximum extent.
5. System performance is monitored to make automatic load balancing possible.

1.5 Related Work for Heterogeneous Disk Array

This section describes some previous work related to various aspects of the heterogeneous disk array. These aspects include proper placement of files to achieve a balanced load, utilizing the extra capacity and bandwidth offered by new disks and incorporating multiple RAID levels in a single disk array.

1.5.1 The File Placement Problem

The objective of the file placement problem is to balance disk workloads (by eliminating disk access skew) or to meet response time requirements for certain applications. The inputs to the file placement problem are a set of file sizes and their access frequency and the system configuration: disk capacities, maximum disk access rates, disk bandwidths (transfer rates), data path characteristics, etc. There have been numerous studies in this field, some of which predate the advent of RAID [23].

A placement optimization program to minimize file access times in multi-disk multi-computer system is described in [69]. The placement decisions are made in two steps: first via a macro model, then a micro model.

In the macro model, non-linear programming algorithm such as the Rosenbrock algorithm [51] is used with an open Queuing Network Model (QNM) as its objective function evaluator. The output of the macro model consists of optimal relative disk access rates, which are access rates normalized by the CPU throughputs[1]. In fact the macro model heuristic does not guarantee a globally optimal solution, though an extremely close to optimal solution is invariably found and the same is true of the micro model heuristic.

The micro model is a Binary Linear Programming Model (BLPM). The QNM is also involved to help determine the BLPM stopping criteria. The objective function measures, for each device, the distance between the optimal relative access rates as computed by the macro model and the sum of the individual file access rates for files assigned to that device. The user can impose constraints, which either assign or restrict the assignment of certain files to certain devices.

The BLPM is solved using a *greedy* heuristic. First, it chooses a fast, but reasonably good starting assignment of files to devices. Then, after imposing a finite nested sequence of increasing neighborhoods about points in the space of file assignments, it searches the neighborhoods in that sequence for an improved assignment. The algorithm is reasonably fast. Given a rather stringent objective, e.g., the difference between the mean response times of accessing two files should less than 0.0001 second, the algorithm converges in a few minutes.

[1] The model presumes multiple computers with different CPU processing powers.

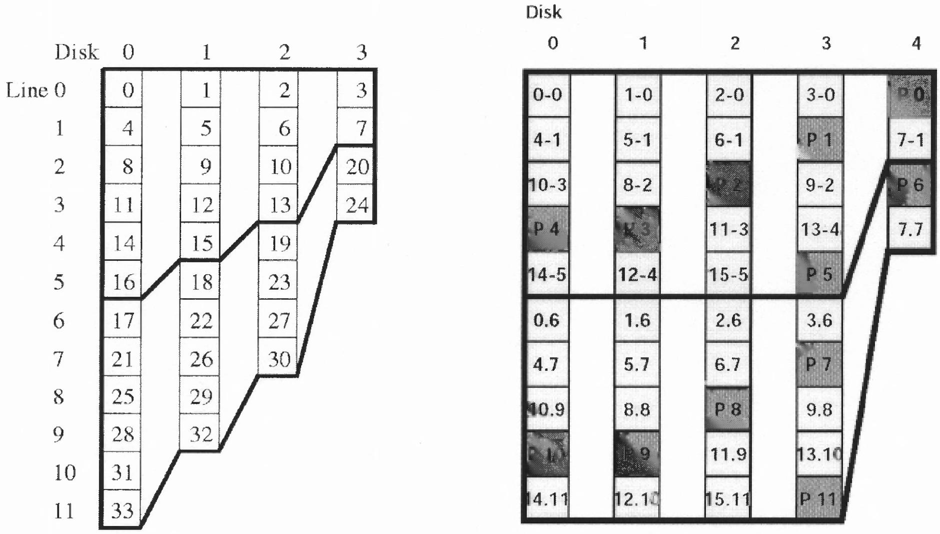
Another model for the file placement problem is described by Hill [32]. Files are modeled as 2-dimensional vectors (size and access rate), and disks are modeled as a container that has a maximum capacity and a maximum throughput (in accesses per second). The problem of file placement then reduces to a vector scheduling problem. However, an appropriate algorithm is not clearly specified.

1.5.2 Techniques to Cope With Disk Heterogeneity

Methods to utilize the extra space for larger disks in a RAID0 or RAID5 systems, named AdaptRaid0 and AdaptRaid5, are described in [19, 20], respectively. Their methods are based on intuitive ideas. Sample data layouts in the two cases are given in Figure 1.4 and are self-explanatory.

Although these algorithms fully utilize disk capacity, they do not take load balancing into account. For example, it is obvious that disk 3 in Figure 1.4(b) has a heavier load than disk 2. This is because there are two parity blocks in a base layout (the layout circumscribed by thick lines) on disk 3, which protect a total of 6 data blocks, while there is only one parity block on disk 2 which protects only 2 data blocks. Therefore, since every data update is accompanied by a parity update, the frequency with witch the parities on disk 3 are updated is much higher than that of disk 2. Another unsolved problem is how to evolve to a new data and parity layout once new disks are added to the system.

More studies of heterogeneous disks are carried out in a multimedia environment, e.g., for video on demand – VOD. A system that can ensure the continuous display using heterogeneous disk-subsystems is described in [72, 71]. The underlying technique is called *disk merging*, which is to organize heterogeneous disks into a group of identical logical disks. The algorithm first chooses how many logical disks should be mapped to each of the *slowest physical disks*. Then the characteristic of the multimedia streams



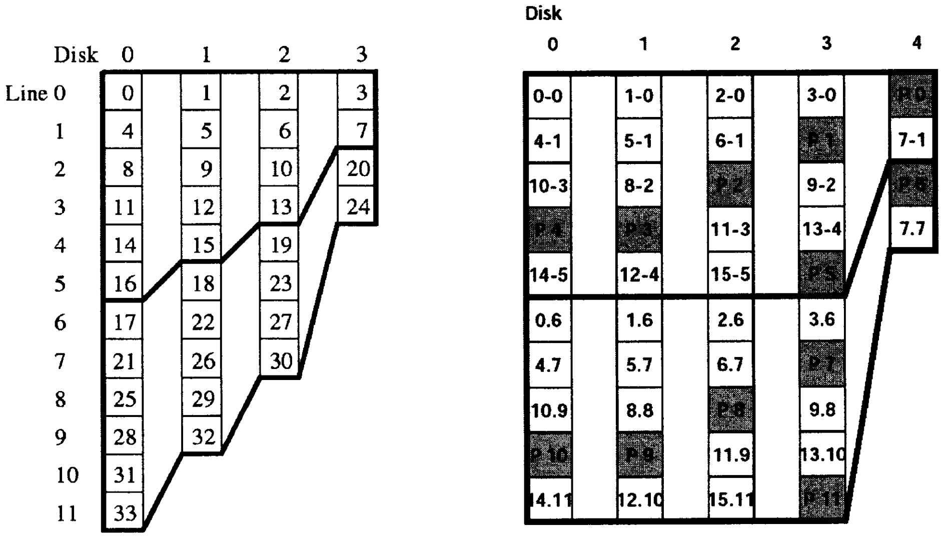
(a) Data Layout in AdaptRaid0 (b) Data and Parity Layout in AdaptRaid5

Figure 1.4 Data Layouts for AdaptRaid0 and AdaptRaid5. Figures are extracted from [19] and [20].

are used to estimate parameters, which are then used to determine how many logical disks map to the other faster disk types in the system (usually these numbers are proportional to the ratios of the bandwidths of those disks to the slowest disk).

Comparing the two methods, it is clear that the latter method tackles the bandwidth aspect of the heterogeneity. Although the latter method can create some identical logical disks out of heterogeneous physical disks and therefore balances the workloads on all disks, some capacity of the larger disks are wasted (unless the bandwidth-capacity ratio for all disks are the same). It should be noted that the workloads in a multimedia system are much more predictable than in a general purpose computer system, in which requests with high concurrency and high variation are expected.

A different approach which is based on Bandwidth to Space Ratio (BSR) is proposed in [21], which attempts to utilize both bandwidth and capacity to the maximum extent. The *BSR deviation* of a device is defined as the deviation of the



(a) Data Layout in AdaptRaid0 (b) Data and Parity Layout in AdaptRaid5

Figure 1.4 Data Layouts for AdaptRaid0 and AdaptRaid5. Figures are extracted from [19] and [20].

are used to estimate parameters, which are then used to determine how many logical disks map to the other faster disk types in the system (usually these numbers are proportional to the ratios of the bandwidths of those disks to the slowest disk).

Comparing the two methods, it is clear that the latter method tackles the bandwidth aspect of the heterogeneity. Although the latter method can create some identical logical disks out of heterogeneous physical disks and therefore balances the workloads on all disks, some capacity of the larger disks are wasted (unless the bandwidth-capacity ratio for all disks are the same). It should be noted that the workloads in a multimedia system are much more predictable than in a general purpose computer system, in which requests with high concurrency and high variation are expected.

A different approach which is based on Bandwidth to Space Ratio (BSR) is proposed in [21], which attempts to utilize both bandwidth and capacity to the maximum extent. The *BSR deviation* of a device is defined as the deviation of the

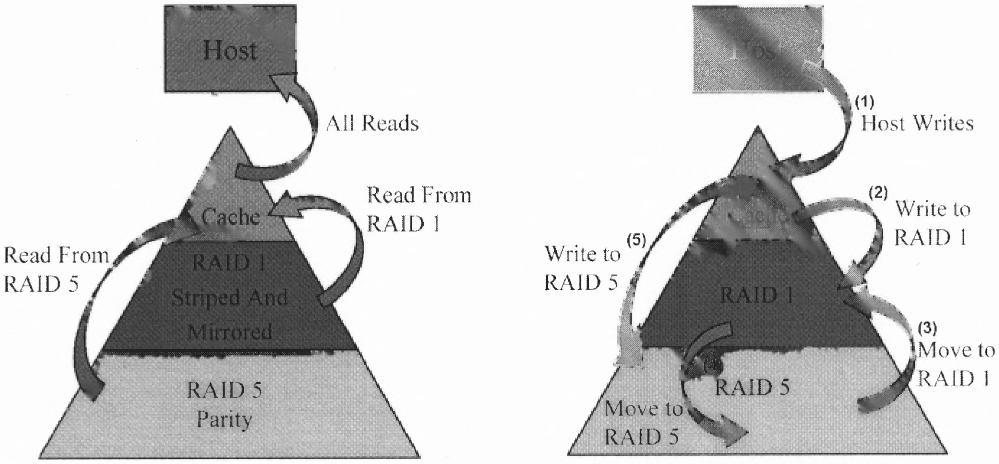
BSR of the video objects on the device from the BSR of that device. The objective function is to minimize the BSR deviation. The heuristic used to select drives for replicas is a greedy algorithm in nature. When placing a video replica, the devices are considered in decreasing order of BSR deviation. Then select from the list the first device whose BSR deviation can be reduced by the video replica being placed.

This algorithm solves the problem fairly well in some circumstances. The Zipf distribution is used to characterize the video access rate in their experiment. The results show both bandwidth and space capacity are almost 100% utilized at the same time, which is ideal. However, their experiment uses disks that have the same BSR, which is not the case in reality. As shown in Section 1.4.4, since disk capacity are growing at an exponential rate, while disk bandwidth are growing much more slowly, the BSRs for more recent disks are smaller than the old ones. Whether their algorithm still performs well in this situation is uncertain.

Another problem is that there are some situations in which the algorithm will result in very skewed space utilization. For example, consider the following situation: there are two disks with BSRs 0.6 and 0.3, the current cumulative BSR of the objects on the two disks are 0.4 and 0.3, respectively. If there are continuous incoming video allocation request with constant BSR 0.5, all the video objects will be placed on the first disk.

1.5.3 System that Have Multiple RAID Levels

1.5.3.1 HP AutoRAID system. The HP AutoRAID hierarchical storage system is a two-level storage hierarchy implemented inside a single disk array controller [68]. At the upper level of this hierarchy, two copies of active data are stored to provide full redundancy and excellent performance via mirroring. At the lower level, RAID5 parity protection is used to provide improved storage cost for less active data, at somewhat lower performance. The hierarchical system automatically and transparently manages migration of data blocks between these two levels as access patterns change. The result is a fully-redundant storage system that is easy to use, suitable for a wide variety of workloads, largely insensitive to dynamic workload changes, and that performs much better than disk arrays with comparable numbers of spindles and much larger amounts of front-end RAM cache [68, 67].



(a) AutoRAID Reads

(b) AutoRAID Writes

Figure 1.5 The reads and writes operations in HP AutoRAID. Figures are extracted from [67].

The reads and writes are handled as shown in Figure 1.5. Reads are served by directly accessing the RAID1 or RAID5 partition of the array, if the data is not in cache. Writes are more complicated. The data are first stored into the write cache (arrow 1). When the data in the cache needs destaging and the old data are in the RAID1 partition, then just update it (arrow 2); if the old data are in RAID5 partition, usually the data are first promoted to the RAID1 level (arrow 3) and then updated (arrow 2).

While promoting data from RAID5 to RAID1, it is possible that the RAID1 partition runs out of space. Therefore, some of the data in the RAID1 level are downgraded to RAID5 level. The selection of data to be downgraded is based on the access frequency and aging policy. In following situations, the cache may destage directly to RAID5 level (arrow 5): (i) if a high rate of sequential I/O's is detected, since a "full stripe" write can be used to update the data; (ii) if the writes to data that are in RAID5 occur very quickly such that the movement of data to RAID1 and associated aging process would consume too much of the controller's bandwidth.

In effect, the AutoRAID uses RAID1 as a cache for RAID5. The current working set are kept in RAID1 partition. When a working set becomes dormant, and there is need for space for a new working set, the old one is downgraded into the RAID5 partition. The scheme works well when the size of working set is not large and can fit into the RAID1 partition. When the working set is larger than RAID1 partition, the constant promoting and downgrading impair the system performance a great deal. When the working set changes frequently, there is also the need to switch data in and out of RAID1 partition, which has the same effect as a large working set.

1.5.3.2 Attribute Managed Storage Design Tools. The Minerva [2] and Ergastulum [5, 6] are two storage system design tools developed at Hewlett-Packard Laboratories. They are two versions of a solution to the *attribute mapping problem* raised in 1995 [27, 11] and formalized in 1996 [58].

The target for both systems is to create a self-configuring and self-managing storage system. The storage is given a specification of the workload it has to support, the data it needs to store, and of the storage devices at its disposal. It then decides how many of each kind of storage device to use, and how to balance the load and data across them. Both the workload behavior and the device capabilities are specified by the *attributes* of the load and the device, respectively.

The workload attributes include performance requirements, such as mean throughput, maximum latency, and jitter; resiliency needs such as availability, reliability, and fault models; cost bounds; data sizes and so on. Device attributes are expressed similarly.

The components in the attribute managed storage system are shown in Figure 1.6. *Storage objects* (also named *stores*) are the basic persistent unit that applications access, and that must be assigned to storage devices. These objects could be files, tables, or parts of tables in a database, a media clip or blocks of a scientific data set.

A *stream* represents the application workload on the object and the resources that the workload uses. It captures the dynamic component of a workload and summarizes its I/O request pattern. The *assignment* metadata are maintained by the system for the mapping of objects to devices. The *mapping engine* takes the requirements of objects and capabilities of devices as input, negotiates and finds out a good mapping of objects to devices such that all the requirements can be satisfied with the lowest cost.

Ergastulum is described here, since it is an improved version of Minerva. The architecture of Ergastulum is shown in Figure 1.7. The workloads are described in terms of *stores* and *streams* as mentioned above.

Ergastulum consists of three main components: a data structure, called the *device tree*, that keeps track of the current design, previous designs and possible configuration changes; a search algorithm that uses various strategies to find a near-optimal design; and a state management component, called *speculation*, that allows Ergastulum to easily roll back to a previously generated design with low overhead.

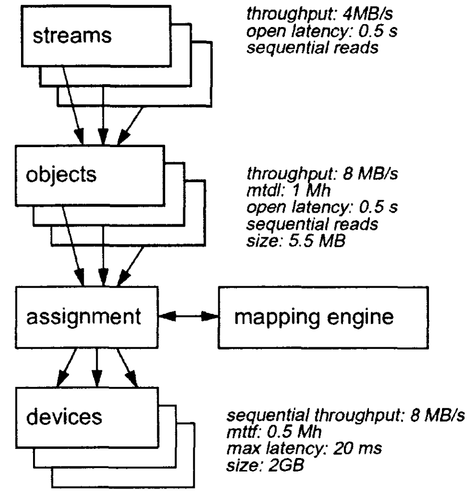


Figure 1.6 The components in the attribute managed storage model.

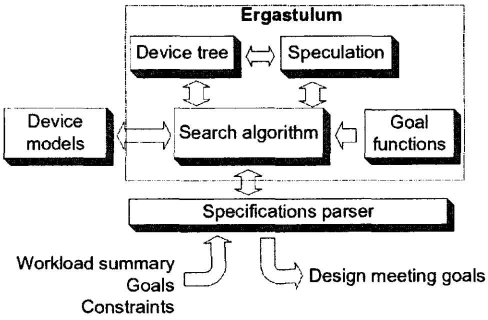


Figure 1.7 Ergastulum's architecture.

In Ergastulum, the mapping engine uses a search heuristic, which is a generalized version of best-fit bin packing with randomization. The algorithm has two phases. The initial assignment starts with empty devices. The list of stores is randomized, each store is assigned into the device tree using a best-fit search of the tree. In order to escape from local minima, in the second phase, a random subset of stores are removed from the device tree and re-assigned.

Attribute managed storage model is a static model and therefore the Minerva and Ergastulum storage system design tools can give a reasonably good design only if the attributes of workloads are known in advance and do not vary often. However, for a general purpose storage system, the workloads may change over time, new applications may be installed and thus introduce new streams. The static design approach is not suitable for such systems. System upgrading is another problem that the static design model cannot solve. Although the static design tool can be executed again after system upgrading, the new solution can be totally different from the old one, which means that all of the old data need to be moved. This usually leads to a long outage of service and can be prohibitively expensive.

1.6 Dissertation Overview

The dissertation consists of two parts. The first part is a description, performance analysis, and comparison of disk arrays that can tolerate two disk failures. The RAID schemes considered are RAID6, EVENODD and RM2. The performance of RAID5 and RAID0 is also analyzed to evaluate the cost of fault tolerance. In Chapter 2 the operations for various RAID schemes are described and analyzed. Then analytical model for the estimation of throughput and response time are described. Chapter 3 provides simulation results for various RAID schemes. The performance for these schemes are compared. The simulation also serves as a validation of the analytical model described in Chapter 2.

The second part of this dissertation is the Heterogeneous Disk Array (HDA) architecture. Chapter 4 describe the architecture of the HDA. The functions of each module are defined. The data structures and algorithms used in the architecture are given. Chapter 5 gives the simulation result for a HDA. Chapter 6 concludes the dissertation.

Appendix A gives the queuing formulas used in the analytical model for the response time of double disk failure tolerant disk arrays.

CHAPTER 2

DESCRIPTION AND PERFORMANCE ANALYSIS OF DOUBLE DISK FAILURE TOLERANT DISK ARRAYS

This chapter describes the analytical model for the performance of double disk failure tolerant disk arrays. It starts with describing the methodology used in the model. Then the workload assumptions and primitive operations are defined. After that the cost models for RAID5, RAID6, EVENODD and RM2 with various number of failures are described. For each disk array scheme, firstly its data layout and operations are analyzed, which is followed by its case graphs and cost functions. The RAID5 analysis is provided for the purpose of drawing a baseline for the comparison and investigating the performance-wise cost of maintaining two check blocks in double failure tolerant arrays. Finally, a queuing analysis utilizing M/G/1 model is described.

2.1 Methodology

To compare the performance of RAID6, EVENODD, RM2 systems with each other and RAID0 and RAID5, firstly several basic disk operations are defined: reads, writes, read-modify-writes, and more complex VSR accesses (see Section 2.3) for RM2. These basic disk operations are the building blocks of other more complicated operations. Given a disk model with its detailed specifications and a RAID configuration, the cost (or mean service time) of these basic operations can be precisely calculated. Then the more complicated operations in various operating modes (e.g, normal mode and degrade mode with single or double disk failures) can be expressed as linear functions of those basic operations. These linear functions are called the cost functions in this dissertation. For a given workload and RAID configuration, the cost functions are utilized with given disk characteristics to estimate the maximum throughput. However, this analysis assumes FCFS (i.e. First Come First Serve) scheduling policy. A much higher maximum throughput can be obtained by using SATF (Shortest Access or positioning Time First) policy, as reported in Chapter 3.

Since the cost functions are not based on assumption of any particular disk model or RAID organization, the performance of various schemes can be compared using this cost model in a disk independent manner. These cost functions combined with request arrival rates can also be used to estimate I/O bus bandwidth requirements. By incorporating M/G/1 queuing model with cost functions, mean user response time can be estimated. Multiple non-preemptive priorities can be solved by this analytical model. For example, read requests can have a higher priority than write requests since application response time is usually defined by underlying read response times. A detailed simulation tool which is call DASim is built from ground up to validate the analytical model as well as to investigate the performance of SATF and other local scheduling policies which are difficult to solve analytically. These results are presented in Chapter 3.

2.2 Workload Assumptions

This section describes the workload assumptions used in the cost model as well as queueing model in subsequent sections.

2.2.1 Request Sizes and Placements

Storage Performance Council's SPC Benchmark-1^{TM} is characterized by "predominantly random I/O operations as typified by multi-user OLTP, database and email server environments" [60]. An analysis of I/O traces from OLTP applications for airline reservations showed that 96% of requests are to 4 KB blocks and the rest to 24 KB blocks [50]. In other words, I/O requests tend to be relatively small blocks of data.

The access time of "modern" disks is dominated by the positioning time for random requests, so that exact sizes of smaller requests have very little effect on performance. When the stripe unit is much larger than the maximum block size being accessed, the possibility that a request will cross stripe unit boundaries and access two disks is quite small.

Based on these facts, the requests are assumed to be randomly distributed over all disk blocks in this dissertation. In fact, it provides a lower bound to performance. However, as noted at a later point, it is relative rather than absolute performance that are of interest. A similar statement applies to the using of the FCFS, rather than the SATF policy in the performance analysis.

The analytic and simulation models presented in this dissertation consider the processing of discrete requests generated by an infinite number of sources, which differ from *continuous requests* usually generated by a finite number of sources and accessing successive blocks of data. In fact, modern disk drives with high data transfer rates are well equipped to handle such requests.

2.2.2 The Arrival Process

Most I/O trace analyses have shown that the arrival rate of requests varies drastically over time. One study recommends to pick peek arrival periods and model their arrival process as Poisson arrivals [70]. Almost all analytical studies of RAID postulate Poisson arrivals, see, e.g. [16], [43], [65].

In reality the arrival process can be modeled more precisely with M/G/1//S model. Disk requests are generated by a finite set of S transactions executing concurrently. Each transaction generates a disk request after a think-time Z. As the queue-length (q) of disk requests at all disks of the disk array increases, the arrival rate decreases according to $(S - q)/Z$. However, the M/G/1 model with infinite sources rather than the M/G/1//S model is used in the analysis because:

1. In OLTP applications, the number of concurrent transactions is large. Therefore, it is close to infinite sources.
2. The throughput can be varied by changing one parameter only (the arrival rate λ) with M/G/1 model, which brings great flexibility in the simulation and modeling.
3. The M/G/1 model is easier to analyze but still provides results that are accurate enough.

For a given disk utilization, the mean response time $R_{M/G/1} > R_{M/G/1//S}$, but this difference gets smaller with increasing S and smaller service time variability [12]. Therefore, M/G/1 model can give a prudent estimation of system response time.

2.2.3 The Effect of Caching

In this dissertation, the performance of disk arrays are compared without considering the disk array controller cache. In effect, the comparisons are done when the disk array is processing *read misses and destages of write requests from the disk array controller cache.* In other words, only the throughput provided by disk arms rather than cache hits are examined. However, caches have great impact on the disk array performance, which are discussed below, but they affect all disk array schemes equally.

The rate of read requests to the disk array is reduced due to hits in the caches. Write requests are considered completed as soon as a dirty block is written into a duplexed NVRAM cache, so that the destaging of dirty blocks from NVRAM can be deferred. If a dirty block is overwritten α times in the NVRAM then the destaging rate is reduced by a factor $1 + \alpha$. Batched destaging of dirty blocks can be optimized to minimize destaging time. Those optimizations are not considered in this study, since α and information about the locality of requests to be destaged are not available.

If the caching of parity blocks is allowed, the cache would be more beneficial for RAID5 than RAID6 and RM2, since RAID6 has twice as many parity blocks as RAID5 and this number is slightly higher for RM2. However, the caching of check blocks is not recommended in [66], since they do not contribute to read hits.

The hard disk's onboard cache hit ratio is expected to be negligibly small for random accesses and is therefore ignored.

To summarize, the effect of caching is ignored. The performance of different RAID systems are compared when they are subjected to the same arrival rates for reads and destages.

2.3 Basic Operations in RAID

This section describes the basic operations used in the cost model. These operations are considered atomic and non-preemptive, which means once an operation is scheduled to run, it can not be interrupted by a higher priority request. The costs for executing read (or write) requests for a RAID scheme are expressed as functions of various types of basic operations specified below.

1. *SR/SW*: Single Read / Single Write of a single block. SW seeks are slightly longer than SR's due to head settling time.

2. *RMW*: Read-Modify-Write of a single block. A RMW is an SR is followed by a full disk rotation to update the data block. A RMW is treated as an atomic disk access in our study.

3. $VSR(M, k)$. Variable-distance Simple Reads are used in RM2 recovery process. It involves accessing $1 \leq k \leq M$ blocks out of M neighboring stripe units on the same disk, which constitute a segment. The blocks are at the same offset in the stripe unit. A sample is shown in Figure 2.1. This operation is introduced since a $VSR(M, k)$ access requires much less seek than k separate SR requests due to the proximity of VSR blocks.

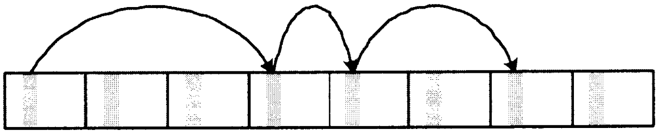


Figure 2.1 A sample $VSR(8,4)$ request. Read 4 blocks out of 8 neighboring stripe units. Each block are at the same offset in its stripe unit.

Hereafter, the notations D_{SR}, D_{SW}, D_{RMW} and $D_{VSR(M,K)}$ are used to denote the cost (i.e. service time) of the corresponding disk operations. In Section 2.8.1 the disk service times are obtained for the aforementioned basic operations.

2.4 Cost of Operations for RAID5 Disk Array

RAID5 is single disk failure tolerant. The cost of operations for RAID5 is discussed in this section to serve as a baseline of comparison. It is also useful when investigating the performance-wise side effect of keeping redundant information.

2.4.1 RAID5 Organization and Operations

RAID5 [48] uses exclusive-OR to calculate the parity information and the parity information is distributed on all hard disks in order to balance the load. There are several different ways to distributed the parity information. The most common way is the left symmetric layout, as show in Figure 1.2(f) on page 8. The parity blocks are on the diagonal. Each parity block is the exclusive-OR of all the data blocks on the same stripe. i.e.

$$P = \bigoplus_{i=1}^{N-1} D_i \tag{2.1}$$

where N is the total number of disks in the array. When the k^{th} data block fails, its content can be reconstructed by rewriting equation 2.1 as:

$$D_k = P \oplus \left(\bigoplus_{i=1, i \neq k}^{N-1} D_i \right) \tag{2.2}$$

The write operation requires to update both the data and parity block. When the size of the data to be written is small, instead of computing the parity all over again using equation 2.1, it is more efficient to do it incrementally as follows:

$$P_{new} = D_{new} \oplus D_{old} \oplus P_{old} \tag{2.3}$$

Thus, one write request will result in four disk accesses: read old data, read old parity, write new data and write new parity. Therefore, the service demand for a small write request is more than doubled compared to a non-redundant system, hence the name *small write penalty.*

When the size of write request is large enough to span at least half of the stripe, the *reconstruct write* strategy is more efficient. In a reconstruct write, the data blocks that are *not* to be updated are read from the same stripe, then they are XORed with the new data to get the new parity by equation 2.1. Then the new parity and data are updated on corresponding disks. However, since OLTP workload with small request size is assumed, the reconstruct write strategy is not used except in cases when there are disk failures.

2.4.2 Cost of Operations in RAID5

This section investigates two operating modes for a RAID5 disk array: normal mode and degraded mode. In normal mode, there is no disk failure, while in degraded mode there is one disk failure. Rebuild mode is a special degraded mode with rebuild process reconstructing the data onto a spare disk. The rebuild process usually runs in a low priority to minimize the impact on user requests. Rebuild mode is not considered in this dissertation.

The costs for read and write operations in normal and degraded modes are analyzed with a case-by-case approach as follows:

2.4.2.1 Normal Mode.

Read Operation In normal mode, all hard disk are working. The cost to read a single block is just a simple read. Therefore,

$$C_{r0} = D_{SR}$$

The subscript "$r0$" means reading with no disk failure.

Write Operation As discussed earlier, the new parity is computed using equation 2.3. Four accesses are required to complete a write request. These four accesses consists two *read-modify-writes*, as shown in Figure 2.2. The cost for write operation is two read-modify-writes.

$$C_{w0} = 2D_{RMW}$$

The subscript "$w0$" means writing with no disk failure.

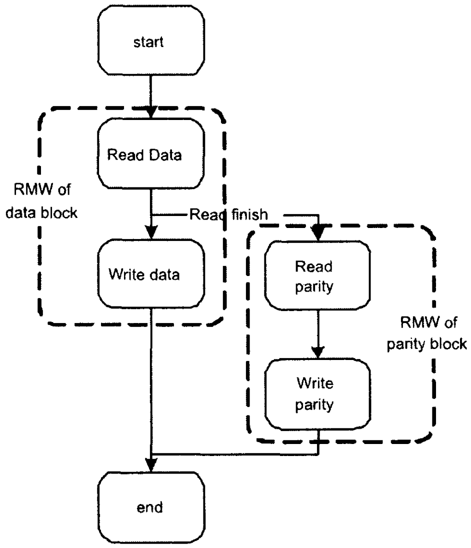


Figure 2.2 Write operation in RAID5 normal mode. Only disk operations are shown, the numerical operations (e.g, exclusive-OR) are omitted since they are not the bottleneck of the performance. The read and write operations on each disk forms an read-modify-write (RMW). The RMW of parity block starts after the read part of RMW on data block is finished.

2.4.2.2 Degraded Mode.

Read Operation In degraded mode, a request may read from a failed disk or non-failed disk.

Case 1 Read from a non-failed disk. The cost in this case is one simple read.

$$c_1 = D_{SR}$$

The probability for this case is:

$$p_1 = \frac{N-1}{N}$$

Case 2 Read from the failed disk.

Then reconstruction is required. Using equation 2.2, the failed block can be reconstructed by reading and exclusive-ORing all the surviving data blocks in the same stripe and the parity block, which totals $(N-1)$ simple reads.

$$c_2 = (N-1)D_{SR}$$

The probability that we read from a failed disk is:

$$p_2 = \frac{1}{N}$$

The cost of read operation with single disk failure is then a weight sum of the two cases:

$$\begin{aligned} C_{r1} &= p_1c_1 + p_2c_2 \\ &= \left(\frac{N-1}{N} + \frac{1}{N}(N-1)\right) D_{SR} \\ &= \frac{2(N-1)}{N} D_{SR} \end{aligned}$$

Write Operation Similarly, there are two cases:

Case 1 Write to non-failed disk, with probability

$$p_1 = \frac{N-1}{N}$$

Depends on whether the associated parity block is on a failed disk or not, there exists two subcases:

Subcase 1.1 The parity block is on the failed disk.

The fraction of this case is:

$$p_{1.1} = \frac{1}{N-1}$$

Since the parity block is unavailable, the parity read and write are not necessary, and the reading of the old value of the data block is omitted as well. Therefore, the cost will be only one simple write.

$$c_{1.1} = D_{SW}$$

Subcase 1.2 The parity block is on a non-failed disk.

Fraction of this case is:

$$p_{1.2} = \frac{N-2}{N-1}$$

Then the data block and parity block need to be updated as in normal mode.

$$c_{1.2} = C_{w0} = 2D_{RMW}$$

Cost of write in case 1 is:

$$c_1 = \frac{1}{N-1}D_{SW} + \frac{2(N-2)}{N-1}D_{RMW}$$

Case 2 Write to failed disk, with probability

$$p_2 = \frac{1}{N}$$

Since the data block is failed, only the parity block need to be updated. The parity is computed using equation 2.1, which means all the remaining data block in the same stripe need to be read, and then exclusive-ORed with the new data, follows by a simple write to the parity disk. The cost is therefore:

$$c_2 = (N-2)D_{SR} + D_{SW}$$

The cost of write with single disk failure is:

$$\begin{aligned} C_{w1} &= p_1c_1 + p_2c_2 \\ &= \frac{N-1}{N}\left(\frac{1}{N-1}D_{SW} + \frac{2(N-2)}{N-1}D_{RMW}\right) + \frac{1}{N}\left((N-2)D_{SR} + D_{SW}\right) \\ &= \frac{2}{N}D_{SW} + \frac{N-2}{N}D_{SR} + \frac{2(N-2)}{N}D_{RMW} \end{aligned}$$

2.4.2.3 Summary. As discussed in the previous subsection, the cases with their probabilities and operating strategies can be represented in a case-breakdown graph. For RAID5 with degraded mode, the case-breakdown graph is shown in Figure 2.3. The fractions in the parentheses are the probability of the corresponding case. The costs for RAID5 read and write operations in normal and degraded mode are summarized in Table 2.1. For the sake of brevity, the lengthy case by case analyses will be omitted for other schemes. Only the most complicated cases will be described. The case-breakdown graph will be shown to sketch the cases and the cost of operations will be given.

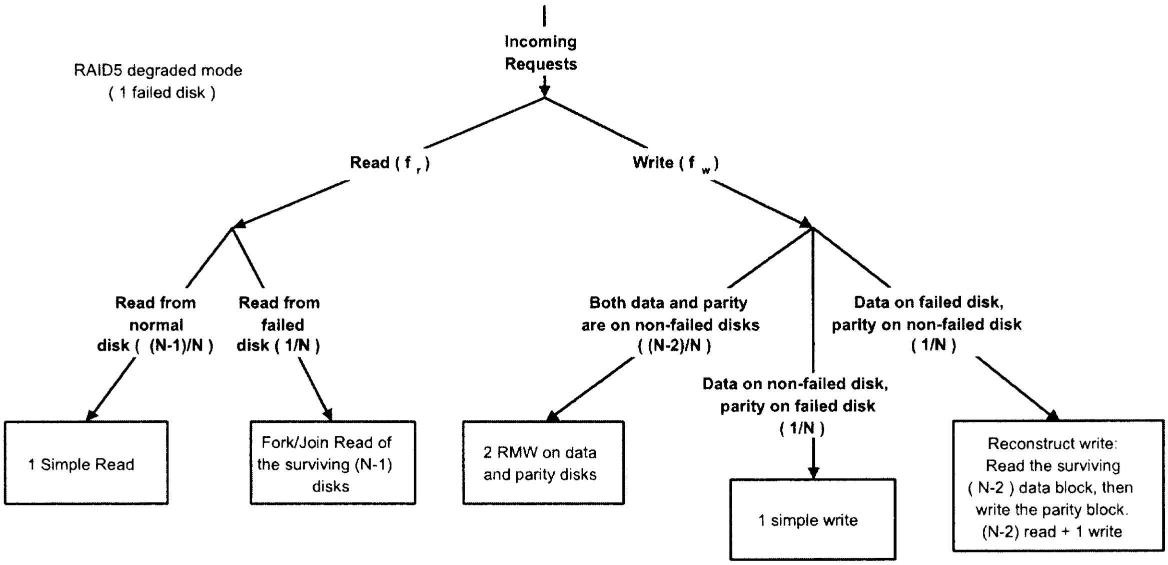

RAID5 degraded mode
(1 failed disk)
Incoming Requests
Read (f r)
Write (f w)
Read from normal disk ((N-1)/N)
Read from failed disk (1/N)
Both data and parity are on non-failed disks ((N-2)/N)
Data on non-failed disk, parity on failed disk (1/N)
Data on failed disk, parity on non-failed disk (1/N)
1 Simple Read
Fork/Join Read of the surviving (N-1) disks
2 RMW on data and parity disks
1 simple write
Reconstruct write: Read the surviving (N-2) data block, then write the parity block. (N-2) read + 1 write

Figure 2.3 The cases for RAID5 degraded mode. N is the total number of disks in the array. f_r and f_w are the fraction of read and write requests, $f_r + f_w = 1$. The fractions in the parentheses are the probabilities for the corresponding cases.

Table 2.1 Cost of Operations for RAID5 with N Disks

Mode	Read	Write
Normal	D_{SR}	$2D_{RMW}$
Degraded	$\frac{2(N-1)}{N}D_{SR}$	$\frac{2}{N}D_{SW} + \frac{N-2}{N}D_{SR} + \frac{2(N-2)}{N}D_{RMW}$

2.5 Cost of Operations for RAID6 Disk Array

RAID6 uses Reed-Solomon code to maintain two redundant disks and therefore can survive two disk failures. The scheme can easily be extended to tolerate more than two disk failures by using more check disks.

2.5.1 RAID6 Organization

RAID6 uses two check disks to tolerate two disk failures, but more generally a disk array can utilize the Reed-Solomon code on n data disks and k check disks ($N = n+k$), with data words (d_i) and checksum words (c_i), as shown below

d_1	d_2	d_3	d_4	$\ldots$	$\ldots$	d_n	c_1	c_2	$\ldots$	c_k

to tolerate up to k disk failures. If the words are w bits wide, the constraint $N \leq 2^w$ applies, which is not a problem in practice.

The algorithm has two main aspects[8]:

- using the Vandermonde matrix to calculate the checksum words;
- using Gaussian Elimination to recover from failures;

It should be noted that all the calculation are performed over Galois Fields.

To compute the checksum words, let vector $\mathbf{d} = (d_1, d_2, d_3, \ldots, d_n)$ be the input data words and vector $\mathbf{u} = (d_1, d_2, \ldots, d_n, c_1, c_2, \ldots, c_k)$ be the coded data words. The checksum words are linear combination of the input words, i.e. the vector $\mathbf{u}$ can be obtained by multiplying $\mathbf{d}$ with a *generator matrix* $\mathbf{G}$, i.e., $\mathbf{u} = \mathbf{dG}$. Since it is desirable to keep the first n words in $\mathbf{u}$ same as $\mathbf{d}$ so that no decoding is required for a reading, the generator matrix $\mathbf{G}$ should be in the form: $\mathbf{G} = [\mathbf{I}|\mathbf{P}]$. $\mathbf{I}$ is an $n \times n$ identity matrix and $\mathbf{P}$ is an $n \times k$ matrix.

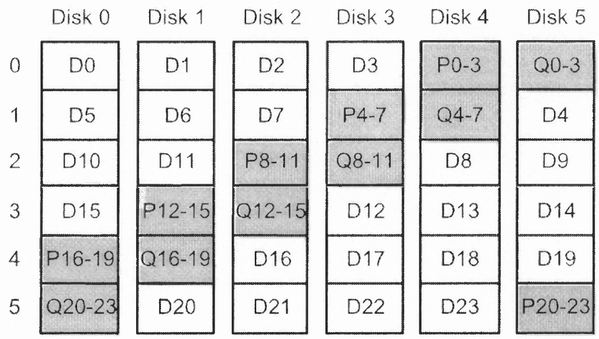
Thus each disk is represented by an element in $\mathbf{u}$ and a column in $\mathbf{G}$. The matrix $\mathbf{G}$ is generated algorithmically through elementary row transformation from an $n \times (n+k)$ Vandermonde matrix $\mathbf{V}$:

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 1^2 & \cdots & 1^{n+k} \\ 1 & 2 & 2^2 & \cdots & 2^{n+k} \\ 1 & 3 & 3^2 & \cdots & 3^{n+k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \cdots & n^{n+k} \end{bmatrix} \Longrightarrow \mathbf{G} = [\mathbf{I}|\mathbf{P}]$$

For a small write, when d'_j $(1 \leq j \leq n)$ overwrites d_j, all the checksum words need to be changed. It is easy to verify that: $c'_i = c_i + g_{j,n+i}\left(d'_j - d_j\right), (1 \leq i \leq k)$, where $g_{x,y}$ is an element in $\mathbf{G}$.

When disk failures occur, since each disk is represented by a column in $\mathbf{G}$ and $\mathbf{u}$, one can just delete the corresponding columns, leading to $\mathbf{G}'$ and $\mathbf{u}'$, such that $\mathbf{u}' = \mathbf{dG}'$. The generator matrix ensures that any subset of n columns are linearly independent. Therefore, the scheme can tolerate up to k failures and still $\mathbf{G}'$ is non-singular. Thus set of equations can be solved to compute $\mathbf{d}$. If there are less than k disk failure, just pick any n surviving disks. More details can be found in [8].

RAID6 is a special case of the above paradigm with $k = 2$ check disks. The left symmetric data layout in RAID5 can be applied on RAID6 as well, as shown in Figure 2.4.



	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
0	D0	D1	D2	D3	P0-3	Q0-3
1	D5	D6	D7	P4-7	Q4-7	D4
2	D10	D11	P8-11	Q8-11	D8	D9
3	D15	P12-15	Q12-15	D12	D13	D14
4	P16-19	Q16-19	D16	D17	D18	D19
5	Q20-23	D20	D21	D22	D23	P20-23

Figure 2.4 RAID6 data organization with left symmetric layout. The shaded area are check blocks. P_{i-j} and Q_{i-j} are the two check words computed over D_i through D_j.

However, the left symmetric data layout exhibits slightly load imbalance problem, especially when the total number of disks N is small [63]. Therefore, a modified left symmetric data layout based on pseudo random number generator is used to ensure that the load is balanced under all operating modes. Since this modified layout itself does not affect the analytical model that follows, it is omitted.

2.5.2 Cost of Operations in RAID6

The processing costs in RAID6 are similar to those in RAID5, except that two rather than one check block need to be updated in RAID6.

In normal mode, the operations of RAID6 are similar to operations in RAID5. A read is a simple read and a write will incur three read-modify-writes: one to the data block and two for both check blocks.

With single disk failure, a data block to be read from a failed disk (probability $f = 1/N$) can be reconstructed by reading the associated data and one of the two

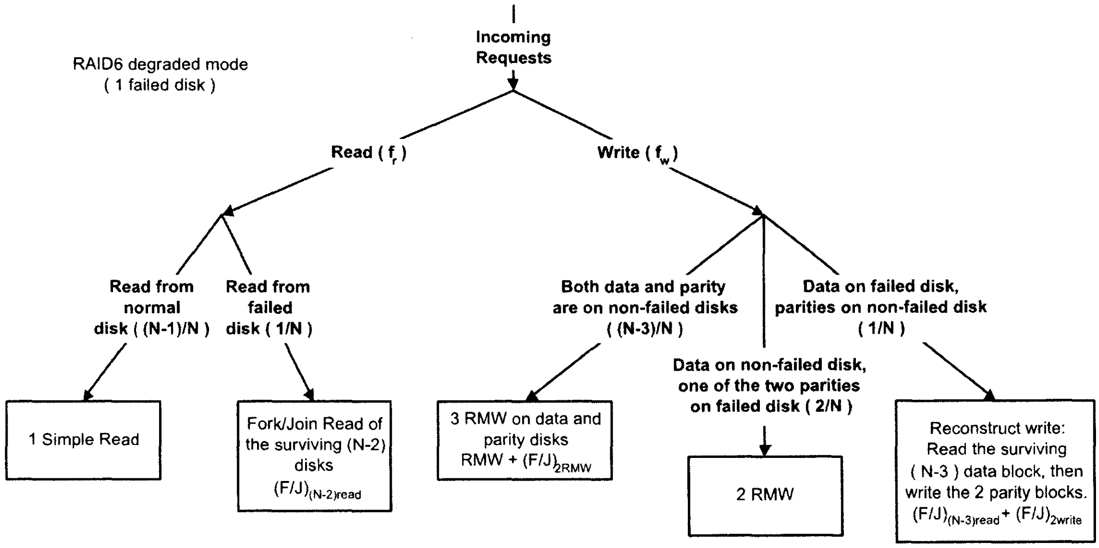

RAID6 degraded mode
(1 failed disk)
Incoming Requests
Read (f_r)
Write (f_w)
Read from normal disk ((N-1)/N)
Read from failed disk (1/N)
Both data and parity are on non-failed disks ((N-3)/N)
Data on failed disk, parities on non-failed disk (1/N)
Data on non-failed disk, one of the two parities on failed disk (2/N)
1 Simple Read
Fork/Join Read of the surviving (N-2) disks $(F/J)_{(N-2)read}$
3 RMW on data and parity disks RMW + $(F/J)_{2RMW}$
2 RMW
Reconstruct write: Read the surviving (N-3) data block, then write the 2 parity blocks. $(F/J)_{(N-3)read}$+ $(F/J)_{2write}$

Figure 2.5 The cases for RAID6 with one disk failure. N is the total number of disks. f_r and f_w are the fraction of read and write requests, $f_r + f_w = 1$. The fractions in the parentheses are the probability for the corresponding case. The symbol (F/J) stands for Fork-Join, with means sub-requests are sent to individual disks and the result will wait until all sub-requests are finished.

check blocks, which is $(N-2)$ SRs in total. A write to a failed block (probability $f = 1/N$) involves the reconstruct write strategy: first read the remain $N-3$ data blocks, then compute both check blocks and overwrite them with two SWs. A write to a non-failed block requires a RMW to the data block followed by one or two RMWs to the parity blocks depending on whether the parities are on the failed disk ($f = 2/N$) or non-failed ($f = (N-3)/N$) disks. These cases are shown in Figure 2.5.

With two disk failures, there are more cases. We list the cases with their probability, operating strategy and corresponding cost in Figure 2.6. Particularly, when both failures are on the check disks, the system is degraded into a RAID0; when one failure is on data while the other failure is on a check disk, the system is equivalent to a RAID5 disk array in degraded mode; when both failures are on data disks, a reconstruction is required only if the request wants to update one of the failed block.

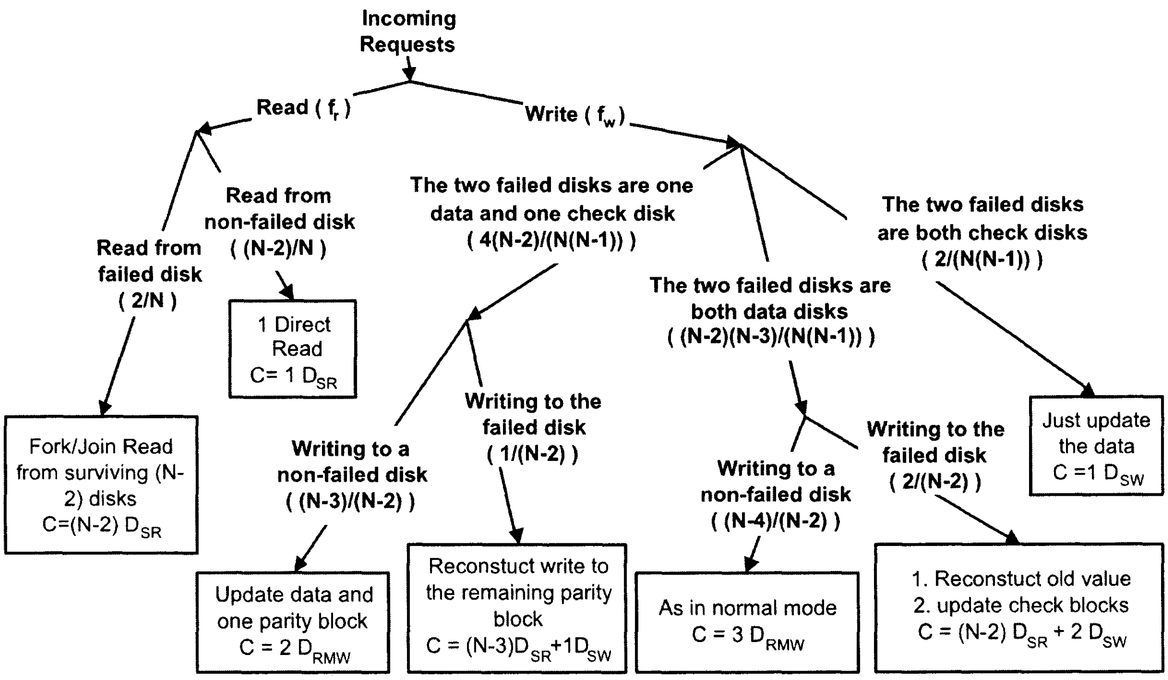

Incoming Requests
Read (f$_r$)
Write (f$_w$)
Read from failed disk (2/N)
Read from non-failed disk ((N-2)/N)
Fork/Join Read from surviving (N-2) disks C=(N-2) D$_{SR}$
1 Direct Read C= 1 D$_{SR}$
The two failed disks are one data and one check disk (4(N-2)/(N(N-1)))
Writing to a non-failed disk ((N-3)/(N-2))
Writing to the failed disk (1/(N-2))
Update data and one parity block C = 2 D$_{RMW}$
Reconstuct write to the remaining parity block C = (N-3)D$_{SR}$+1D$_{SW}$
The two failed disks are both data disks ((N-2)(N-3)/(N(N-1)))
Writing to a non-failed disk ((N-4)/(N-2))
Writing to the failed disk (2/(N-2))
As in normal mode C = 3 D$_{RMW}$
1. Reconstuct old value 2. update check blocks C = (N-2) D$_{SR}$ + 2 D$_{SW}$
The two failed disks are both check disks (2/(N(N-1)))
Just update the data C =1 D$_{SW}$

Figure 2.6 The cases for RAID6 with two disk failures. N is the total number of disks. f_r and f_w are the fraction of read and write requests, $f_r + f_w = 1$. The fractions in the parentheses are the probability for the corresponding case.

The cost of operations for RAID6 with none, one or two disk failures are summarized in Table 2.2.

2.6 Cost of Operations for EVENODD Disk Array

The EVENODD scheme [9] was introduced by M. Blaum et al, in 1995. EVENODD has two desirable features: Firstly, the coding is based solely on the XOR operation, so that its hardware implementation requires the same hardware as RAID5. A complexity comparison of XOR operations is provided in [9], which shows that the ratio of the number of XOR operations for RAID6 versus EVENODD tends to five as the number of disks increases. Secondly, it utilizes two parity disks to protect against two disk failures, which is the minimum redundancy possible. In this section, after a brief description of EVENODD, the cost functions for its operations are provided.

Table 2.2 Cost of Operations for RAID6 with N Disks

Mode	Read	Write
normal	D_{SR}	$3D_{RMW}$
single failure	$\frac{2N-3}{N}D_{SR}$	$\frac{3N-5}{N}D_{RMW} + \frac{N-3}{N}D_{SR} + \frac{2}{N}D_{SW}$
double failure	$\frac{3(N-2)}{N}D_{SR}$	$\frac{(N-3)(3N-4)}{N(N-1)}D_{RMW} + \frac{2(N-2)(N-3)}{N(N-1)}D_{SR} + \frac{4N-6}{N(N-1)}D_{SW}$

2.6.1 The EVENODD Data Layout

The EVENODD scheme organizes data as *symbols* in an $(m-1)\times(m+2)$ array, where m is the number of data disks. This array is referred to as a *segment*. Each column in the array represents a disk. Column m and $m+1$ hold redundant information referred to as the P and Q parities. The symbol size can be as small as one bit to as large as desired.

Parity P is the XOR of all the data symbols in the same row, which is similar to the parity in RAID5. The Q parities are computed over a diagonal that spans $m-1$ rows, as shown in Figure 2.7.

A special diagonal parity that is distributed over all parity Q symbols, denoted as S, is the exclusive-OR of those data symbols marked with ∞. i.e. $S = \bigoplus \infty$. The name of the scheme comes from the diagonal parity S. On the other hand, S can also be obtained as the exclusive-OR of all the symbols in columns m and $m+1$, which correspond to all symbols in parity P and Q, respectively.

The EVENODD scheme works only when the number of data disks m is a prime number [9]. When m is not prime, there is a simple way around it: select a

Disk0	Disk1	Disk2	Disk3	Disk4	Parity P	Parity Q
$\diamondsuit$	$\clubsuit$	$\heartsuit$	$\spadesuit$	∞		$\diamondsuit \oplus \infty$
$\clubsuit$	$\heartsuit$	$\spadesuit$	∞	$\diamondsuit$		$\clubsuit \oplus \infty$
$\heartsuit$	$\spadesuit$	∞	$\diamondsuit$	$\clubsuit$		$\heartsuit \oplus \infty$
$\spadesuit$	∞	$\diamondsuit$	$\clubsuit$	$\heartsuit$		$\spadesuit \oplus \infty$

Figure 2.7 The parity Q in EVENODD scheme. Sample shown uses 5 data and 2 parity disks, corresponding to $m = 5$. The notation $\diamondsuit \oplus \infty$ means this symbol is computed by exclusive-ORing all the data symbols marked with $\diamondsuit$ and ∞. This graph is a modified reproduction of the graph in [9].

prime number larger than the number of disks, and assume that the rest are virtual disks with all zero contents.

When there are one disk failure, the failed data can be reconstructed in the same way as in RAID5 since the parity P is same as in RAID5. When there are two disk failures and both failures happen with data disks, a complex process is required to reconstruct the failed symbols. However, as will be shown in the next section, this complexity does not affect our analysis on disk accesses when the symbol size is adequately small. Therefore, the lengthy discussion on the reconstruction process is omitted.

2.6.2 Cost of Operations in EVENODD

The EVENODD scheme is as efficient as RAID6 when symbol size is adequately small [9]. If the minimum block size is 4KB, the symbol size can be $4096/(m-1)$ so that each column in the segment will be one block. There are several good prime numbers for this purpose, e.g, $m = 17$ and $m = 257$. One can vertically stack as many segments as required, with the data allocated consecutively across segment boundaries on a disk, to make the striping unit as large as desired. Again, the actual number of disks in the array can be less than $m + 2$ since the rest can always be virtual disks.

With proper small symbol size, the reading of a block will correspond to the reading of a whole column, and the writing of a block in normal mode will be the updating of the data block and two parity columns, which are also one block each. The disk access pattern in EVENODD in normal mode is then exactly the same as in RAID6, so their performance will be indistinguishable from the viewpoint of disk access.

With single disk failure, the reading of a failed block will result in reading all the corresponding data blocks and the parity P block followed by an exclusive-OR of these blocks. As to the write operation, first reconstruct the old value and then update the column of parity P and Q by two RMWs. The access pattern is the same as in RAID6 and so is the cost.

When there are two failed disks, in order to recover from the failure, we need to read all the surviving symbols, which are equivalent to all surviving blocks in a row. This cost is again the same as the cost in RAID6.

To summarize, EVENODD and RAID6 are identical from the viewpoint of disk access pattern in all operating modes. Therefore, the cost functions for RAID6 in Table 2.2 are applicable to EVENODD with adequately small symbol size.

2.7 Cost of Operations for RM2 Disk Array

RM2 [47] was introduced by C. I. Park in 1995. It features

- Tolerate double disk failure.
- Use exclusive-OR only so that we can use the current hardware.

- Be able to get a layout for arbitrary redundancy ratio.

In this section, the RM2 redundancy scheme is described first. This is followed by descriptions of its operations and operating costs.

2.7.1 The RM2 Data Layout

The double failure data placement given in [47] is defined as: "Given a redundancy ratio p and the number of disks N, construct N parity groups each of which consists of $2(M-1)$ data blocks and one parity block such that each data block should be included in two groups, where $M = 1/p$." Each disk contains one parity and $M-1$ data units in a segment, so that the parity blocks are distributed evenly.

An algorithmic solution to this problem is based on an $N \times N$ *redundancy matrix (RM)*, where each column corresponds to a disk and each row corresponds to a parity group. The columns of RM are called *placement vectors.* Values of the elements of RM, $RM_{i,j}$, are defined as follows:

- $RM_{ij} = -1$ A parity block of the disk j belongs to parity group i.
- $RM_{ij} = 0$ Nothing (none of the blocks on disk j belongs to parity group i).
- $RM_{ij} = k$ The kth data block of disk j belongs to group i. $(1 \leq k \leq M-1)$

The redundancy matrix RM must have the following properties: (1) There is only one -1 in each row, i.e., each parity group (PG) has one parity block. (2) There is only one -1 in each column, i.e. one parity block for each disk in a segment. (3) Each column has $2(M-1)$ positive entries with each number appearing exactly twice, i.e., each data block is protected by exactly two parity blocks.

An RM2 data layout is defined by an RM, which can be constructed as follows:

1. Select the target redundancy ratio p and set $M = 1/p$.

2. Select the total number of disks N that satisfy: $N \geq 3M - 2$ if N is odd or $N \geq 4M - 5$ if N is even.

3. Construct a *seed placement vector* for M and N as (the prime here implies a transpose):

$$\underbrace{\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline -1 & M-1 & M-2 & \dots & 2 & 1 & 1 & 2 & \dots & M-2 & M-1 & 0 & \dots & 0 \\ \hline \end{array}}_{a\ total\ of\ N\ elements}{}'$$

4. Construct the $N \times N$ RM matrix column-by-column by rotating the seed placement vector, as shown below:

$$RM = \begin{bmatrix} -1 & 0 & \cdots & M-1 \\ M-1 & -1 & \cdots & M-2 \\ M-2 & M-1 & \cdots & M-3 \\ \vdots & \vdots & & \vdots \\ 1 & 2 & \cdots & 1 \\ 1 & 1 & \cdots & 2 \\ 2 & 1 & \cdots & 3 \\ \vdots & \vdots & & \vdots \\ M-1 & M-2 & \cdots & 0 \\ 0 & M-1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & -1 \end{bmatrix}_{N \times N}$$

	D0	D1	D2	D3	D4	D5	D6
PG0	-1	0	0	2	1	1	2
PG1	2	-1	0	0	2	1	1
PG2	1	2	-1	0	0	2	1
PG3	1	1	2	-1	0	0	2
PG4	2	1	1	2	-1	0	0
PG5	0	2	1	1	2	-1	0
PG6	0	0	2	1	1	2	-1

(a)

D0	D1	D2	D3	D4	D5	D6
P_0	P_1	P_2	P_3	P_4	P_5	P_6
$D_{2,3}$	$D_{3,4}$	$D_{4,5}$	$D_{5,6}$	$D_{0,6}$	$D_{0,1}$	$D_{1,2}$
$D_{1,4}$	$D_{2,5}$	$D_{3,6}$	$D_{4,0}$	$D_{5,1}$	$D_{6,2}$	$D_{0,3}$

(b)

Figure 2.8 A Sample RM2 Layout with M=3 and N=7: (a)The redundancy matrix (RM), (b)Corresponding Disk Layout. D0 ... D6 are hard disks, PG0 ... PG6 are parity groups. The notation $d_{i,j}$ means this data block is protected by parity blocks p_i and p_j.

For example, given $p = 1/3$, then $M = 1/p = 3$. $N = 7$ is the smallest number satisfying the inequalities. The seed placement vector is $(-1, 2, 1, 1, 2, 0, 0)'$. The RM matrix and data layouts are shown in Figure 2.8. Note that the parity group size for the RM2 scheme is $2M - 1$ blocks, since each parity block protects $2M - 2$ data blocks. The inequalities imply that $p = 1/M \geq 3/(N + 2)$, which means that RM2 has a higher redundancy ratio than RAID6, which is always $2/N$.

The physical position of a block can be different within a disk. For example, in disk 0, the position of P_0 and $D_{2,3}$ can be exchanged and still can tolerate double disk failure.

2.7.2 Cost of Operations in RM2

In normal mode the cost is $C = D_{SR}$ for reads and $C = 3D_{RMW}$ for writes, since two parity blocks in addition to the data block need to be updated.

With one failed disk, a data block on the failed disk can be reconstructed by using either of its parity groups. Since the parity group size is $2M - 1$, the reconstruction requires reading $2M - 2$ blocks. For write requests, corresponding parity blocks need to be updated if they are available. In the case that the data block is failed, reconstruct write strategy is applied on one of the two parities, which involves $2M - 3$ reads. For the other parity, instead of using reconstruct write and reading another $2M - 3$ blocks, the old data is first reconstructed and the second parity is updated in a incremental manner. An example for this case is shown in Figure 2.9. The cases and their corresponding costs are shown in Figure 2.10 for RM2 with one disk failure.

When there are two disk failures, depending on whether the target block is failed or not, there are different strategies as described below.

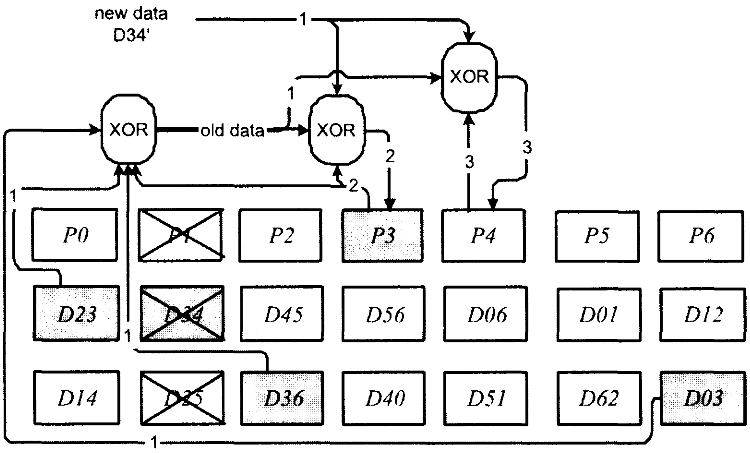


Figure 2.9 An example of writing a failed block in RM2 with one disk failure. D_{34} is on a failed disk and is being updated. D_{34} is protected by parities P_3 and P_4. The shaded blocks show the parity group of P_3. The steps to update D_{34} are: (1) Reconstruct old value of D_{34}. (2) Update parity $P3$. (3) Update parity $P4$ using Read Modify Write. Steps 2 and 3 can be done in parallel.

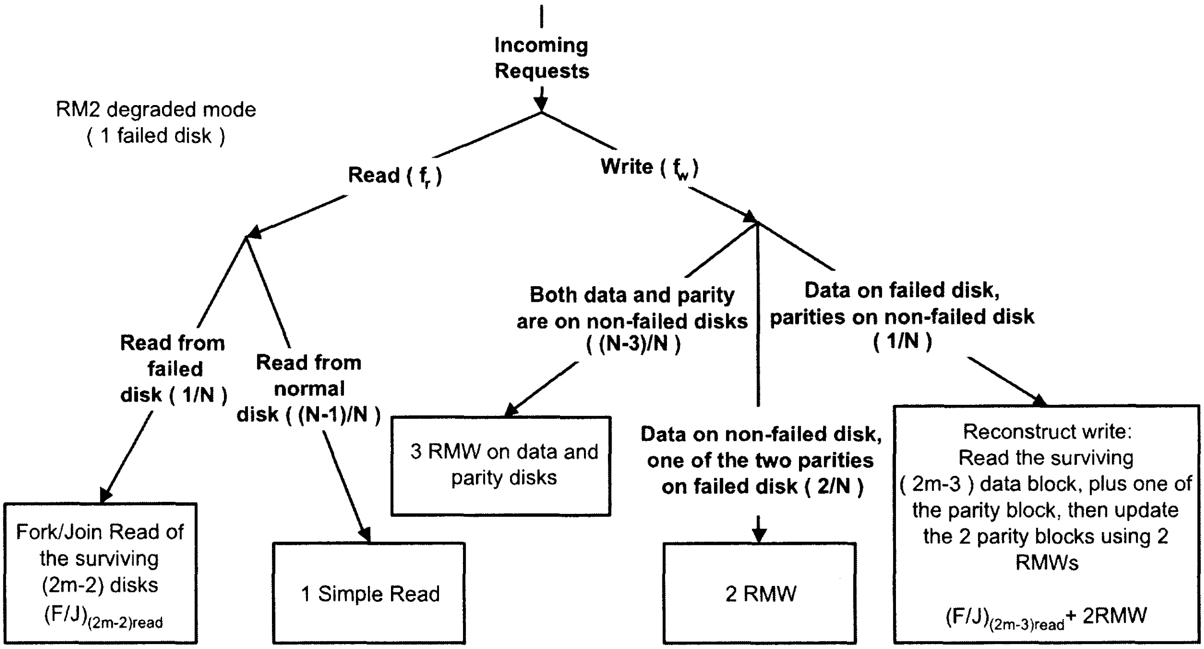

Incoming Requests
RM2 degraded mode
(1 failed disk)
Read (f_r)
Write (f_w)
Read from failed disk (1/N)
Read from normal disk ((N-1)/N)
Both data and parity are on non-failed disks ((N-3)/N)
Data on failed disk, parities on non-failed disk (1/N)
Data on non-failed disk, one of the two parities on failed disk (2/N)
Fork/Join Read of the surviving (2m-2) disks (F/J)(2m-2)read
1 Simple Read
3 RMW on data and parity disks
2 RMW
Reconstruct write: Read the surviving (2m-3) data block, plus one of the parity block, then update the 2 parity blocks using 2 RMWs
(F/J)(2m-3)read+ 2RMW

Figure 2.10 The cases for RM2 with one disk failure. N is the total number of disks. f_r and f_w are the fraction of read and write requests, $f_r + f_w = 1$. The fractions in the parentheses are the probability for the corresponding case. F/J is the short for fork/join.

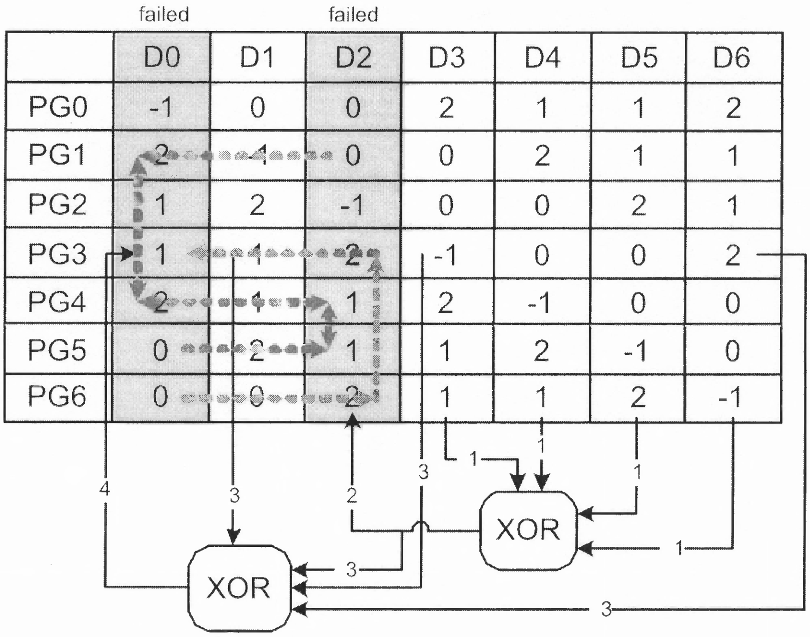
With two disk failures, a read request to a block is successful with probability $f = (N-2)/N$ and the cost is $C = D_{SR}$. The target block is unavailable with probability $2/N$, in which case a *recovery path* is needed and the number of steps varies from 1 to $2M-2$.

As an example, in Figure 2.8 consider an access to $d_{2,3}$ with disk D0 and D3 failed. Utilizing p_2 and the associated parity group then $d_{2,3} \leftarrow p_2 \oplus d_{2,5} \oplus d_{6,2} \oplus d_{1,2}$, which is quite similar to the process in RAID5. However, if D2 fails instead of D3 then $d_{2,3}$ cannot be reconstructed using p_3 directly, since $d_{3,6}$ is not available. However, $d_{3,6}$ can be reconstructed using p_6, so we have the following two steps: (1) $d_{3,6} \leftarrow p_6 \oplus d_{5,6} \oplus d_{0,6} \oplus d_{6,2}$. (2) $d_{2,3} \leftarrow p_3 \oplus d_{3,4} \oplus d_{3,6} \oplus d_{0,3}$. Therefore, the recovery path to rebuild block $d_{2,3}$ is $d_{3,6} \rightarrow d_{2,3}$, whose length is two. It is easy to ascertain that with D0 and D1 failed, the path to reconstruct $d_{1,4}$ is $d_{2,5} \rightarrow d_{2,3} \rightarrow d_{3,4} \rightarrow d_{1,4}$ and 2.4 disk accesses are required per (surviving) disk. Figure 2.11 shows this recovery process and recovery paths.

Read requests in RM2 with two disk failures are processed as the $VSR(M,k)$ request type (introduced in Section 2.3), where $1 \leq k \leq M$ blocks from M stripe units are read from a disk. The mean number of blocks to be read to reconstruct a failed block is denoted by F. Clearly, F is a function of N and M. To determine this function, instead of embarking on a lengthy combinatorial analysis, all possible cases for array configurations with $N \leq 100$ and $M \leq 20$ are enumerated, and then a surface-fitting with respect to N and M is done using Matlab. The surface is shown in Figure 2.12. The result is:

$$\begin{aligned} F = \; & 0.002N^2 + 0.1406M^2 - 0.0364NM \\ & -0.0847N + 3.7359M - 5.323 \end{aligned}$$

Although with the original RM2 data layout, the load on each disk is not balanced when there are two disk failures. However, this can be solved by shuffling the columns according to a pseudo random number generator [63]. Therefore, the load is balanced and those F blocks are assumed to be evenly distributed over the



	D0	D1	D2	D3	D4	D5	D6
PG0	-1	0	0	2	1	1	2
PG1	2	1	0	0	2	1	1
PG2	1	2	-1	0	0	2	1
PG3	1	1	2	-1	0	0	2
PG4	2	1	1	2	-1	0	0
PG5	0	2	1	1	2	-1	0
PG6	0	0	2	1	1	2	-1

Figure 2.11 Recover from double disk failure in RM2. D0 and D2 are the two failed disks. The path on the left is bidirectional, which means the reconstruction can start from either end, while the path on the right has only one direction. To recover block b_0^1 (the first data block on disk 0, which is represented by the two 1's in the D0 column, or the block $d_{2,3}$ in Figure 2.8(b)), which is covered by parity group 2 and 3: (1) Read blocks b_3^1, b_4^1, b_5^2, b_6^0. (2) Reconstruct block b_2^2 by equation $b_2^2 = b_3^1 \oplus b_4^1 \oplus b_5^2 \oplus b_6^0$. (3) Read blocks b_1^1, b_3^0, b_6^2. (4) Reconstruct block b_0^1 by equation $b_0^1 = b_2^2 \oplus b_1^1 \oplus b_3^0 \oplus b_6^2$.

$N-2$ surviving disks, the number of accesses per disk is $F/(N-2)$. Those $F/(N-2)$ block accesses consists a single VSR operation. Therefore, the overall cost is $C = (N-2)D_{VSR(M,\frac{F}{N-2})}$.

The case analysis for write requests is similar to the read, except it has more cases. In general, a lost data block is reconstructed as in read and then the available parity blocks are written. Those cases as well as the cases in processing read requests are shown in Figure 2.13.

The cost for a read or write operation with two disk failure is then a weighted sum of the costs show in the figure. The costs of operations for an RM2 disk array with none, one or two disk failures are summarized in Table 2.3.

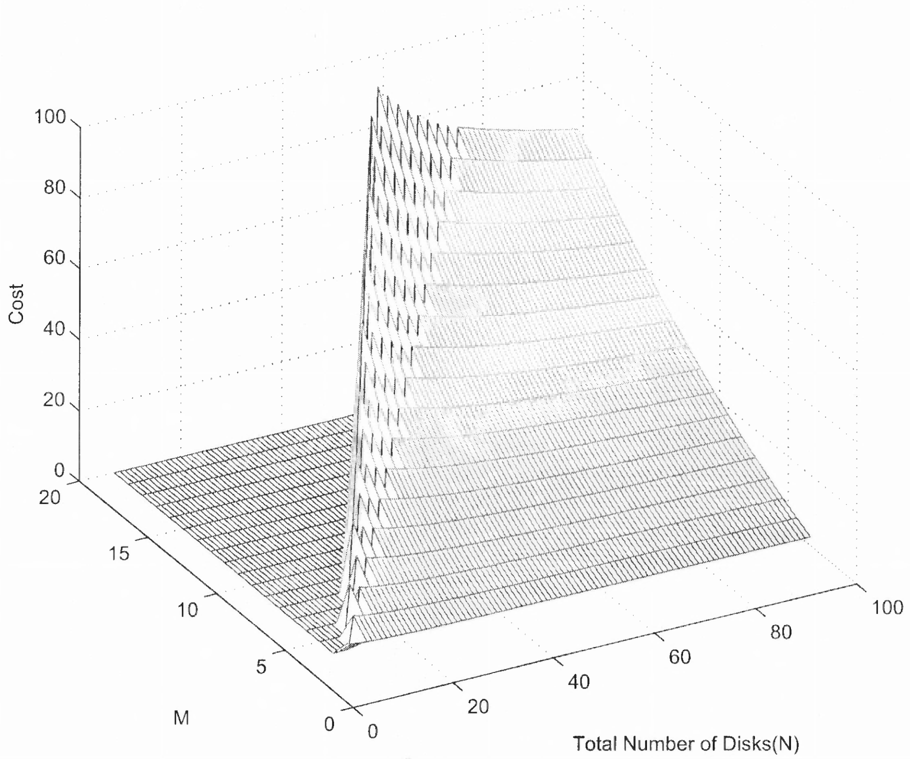


Figure 2.12 Surface fitting for the number of blocks to be read (F) in order to reconstruct a failed block in RM2 disk array with two disk failures. For each N and M, the recovery length and the number of blocks to be read are calculated my enumerate all possible cases with a program.

Table 2.3 Cost of Operations for RM2 with N disks

Mode	Read	Write
Normal	D_{SR}	$3D_{RMW}$
One failure	$\frac{N+2M-3}{N}D_{SR}$	$\frac{2M-3}{N}D_{SR}+\frac{3N-3}{N}D_{RMW}$
Two failures	$\frac{2(N-2)}{N}D_{SR}+\frac{N-2}{N}D_{VSR(M,\frac{F}{N-2})}$	$\frac{2}{N-1}D_{SW}+\frac{(3N-2)(N-3)}{N(N-1)}D_{RMW}+$ $\frac{2(N-2)}{N}D_{VSR(M,\frac{F}{N-2})}$

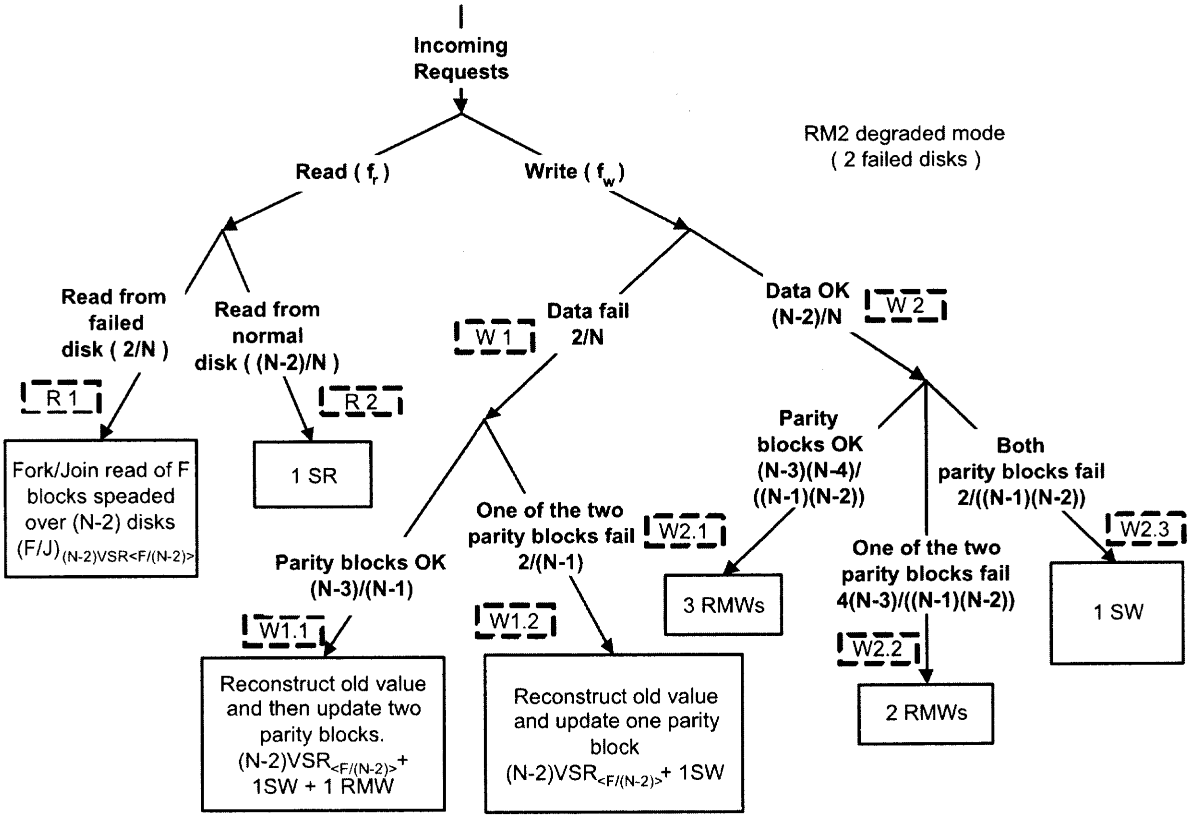

Incoming Requests
RM2 degraded mode (2 failed disks)
Read (f_r)
Write (f_w)
Read from failed disk (2/N)
Read from normal disk ((N-2)/N)
R 1
R 2
Fork/Join read of F blocks speaded over (N-2) disks $(F/J)_{(N-2)VSR<F/(N-2)>}$
1 SR
W 1
Data fail 2/N
Data OK (N-2)/N
W 2
Parity blocks OK (N-3)/(N-1)
W1.1
One of the two parity blocks fail 2/(N-1)
W2.1
W1.2
Reconstruct old value and then update two parity blocks. $(N-2)VSR_{<F/(N-2)>}$+ 1SW + 1 RMW
Reconstruct old value and update one parity block $(N-2)VSR_{<F/(N-2)>}$+ 1SW
Parity blocks OK (N-3)(N-4)/ ((N-1)(N-2))
3 RMWs
One of the two parity blocks fail 4(N-3)/((N-1)(N-2))
W2.2
2 RMWs
Both parity blocks fail 2/((N-1)(N-2))
W2.3
1 SW

Figure 2.13 The cases for RM2 with two disk failures.

2.8 Analytical Model

In this section, the maximum throughput and read response time for a disk array are obtained given its scheme, number of disks, stripe unit size, etc. Characteristics of the disks constituting the array are specified by detailed information on disk geometry and a seek time table. The workload assumptions are described in Section 2.2. The scheduling policy is FCFS or FCFS with read priority.

The stripe unit size is selected to be large compared to most request sizes to ensure that the possibility of stripe crossover and multiple disk accesses is negligibly small. With increasing disk capacities and higher data volumes, much larger stripe units can be sustained without introducing an access skew.

This section is organized as follows. Firstly the moments of service time for basic operations are calculated, which is followed by the derivation of service times in RAID disk arrays. Finally the equations are provided for estimating the mean response time for read requests in normal and degraded mode, which includes fork-join requests.

The notation x with different subscripts is used to denote corresponding service time. W is mean waiting time, R is mean response time, f_r and f_w are fraction of read/write requests and X is maximum throughput.

2.8.1 Service Time for Basic Operations

The analysis presented here is applicable to modern disk drives with zoning, such as the IBM 18ES (model DNES-309170W)[1], whose detailed characteristic, as well as many others, are available at [55]. This drive is used to set the parameters in the analytic and simulation tools. The basic parameters for this drive are listed in Table 2.4.

Table 2.4 IBM 18ES Specifications

Model	IBM DNES-309170W
Capacity	9170M
No. of cylinders (cyl)	11474
Tracks per cylinder	5
Rotation speed	7200 rpm
Time to rotate (T_{rot})	8.3 ms
Sectors per track	247-390
Number of zones	11
Mean seek time	7.16 ms

[1] `http://www.storage.ibm.com/hdd/prod/ultrastar.htm`

Disk service time has three components: seek, latency and transfer time, which are represented by the random variables x_s, x_l and x_t, respectively. Service time for SR, SW, and RMW requests are:

$$x_{SR} = x_s + x_l + x_t$$

$$x_{SW} = x_{SR} + T_h$$

$$x_{RMW} = x_{SR} + T_{rot}$$

where T_h is the head settling time for write, and $T_{rot} = 60/RPM$ is the disk rotation time. The transfer delays in path elements, parity calculation and disk controller time are ignored because: firstly, they are small and overlapping each other; secondly, such hardware details are not available.

The i^{th} moment of seek time $(\overline{x_s^i})$ requires the seek time characteristic and seek distance distribution $P_D(d)$. The seek time characteristic as well as detailed zoning information can be obtained by using the DixTrac tool developed at CMU [55]. The characteristic for several hard drives are available at [22].

For non-zoned disks, every cylinder stores the same amount of data, therefore, $P_D(0) = 1/cyl$ and $P_D(d) = \frac{2(cyl-d)}{cyl(cyl-1)}$, $1 \leq d \leq cyl - 1$, where cyl is the number of cylinders [65]. The mean seek distance is 1/3 of total cylinder number. For zoned disks, all cylinders do not contain the same amount of data. Therefore, this equation does not hold. The seek distance distribution for zoned disk is calculated as follows.

Given uniform access assumption, the probability that the read/write head is at cylinder k, denoted as $P_K(k)$, is proportional to the volume of data on that cylinder:

$$P_K(k) = \begin{cases} \frac{\text{\# sectors on cylinder k}}{\text{\# sectors on disk}} & 1 \leq k \leq cyl \\ 0 & otherwise \end{cases}$$

where *cyl* is the number of cylinders. This model can be easily extended to handle nonuniform disk accesses, e.g., hot data sets residing on certain disk cylinders, but this is beyond the scope of this discussion.

The probability of a seek with distance d when the read/write head is currently at cylinder k is:

$$P_{D|K}(d|k) = \begin{cases} P_K(k+d) + P_K(k-d) & d \neq 0 \\ P_K(k) & d = 0 \end{cases}$$

Then the seek distance distribution $P_D(d)$ can be obtained by unconditioning with respect to $P_K(k)$:

$$P_D(d) = \sum_{k=1}^{cyl} P_{D|K}(d|k) \times P_K(k).$$

Given the seek time characteristic, the i^{th} moment of seek time is:

$$\overline{x_s^i} = \sum_{d=0}^{cyl-1} P_D(d) \times [\texttt{seek\_time}(d)]^i$$

The analysis above takes into account spare cylinders but assumes that there are no bad sectors or tracks.

The latency is uniformly distributed over $(0, T_{rot})$ when request sizes are small with respect to track size regardless of whether zero-latency accesses are possible or not. The first three moments of latency are

$$\overline{x_l^i} \approx \frac{T_{rot}^i}{i+1} \qquad (i = 1, 2, 3)$$

The transfer time of a block consisting of j sectors on cylinder k is obtained as

$$x_t(j|k) = \frac{j \times T_{rot}}{\#\text{ of sectors on track } k}$$

The probability that the block is on cylinder k is $P_K(k)$ as before, so that the i^{th} moment of transfer time is:

$$\overline{x_t^i}(j) = \sum_{k=1}^{cyl} x_t^i(j|k) P_K(k)$$

Given the composition of request sizes, it is possible to obtain the moments over different transfer sizes. However, when all transfer sizes are small, the average transfer time can be treated as a constant.

The three random variables x_s, x_l, x_t are independent for small requests, which implies the expectation of the product of two variable is the product of their individual expectations (e.g. $\overline{x_s \cdot x_l} = \overline{x_s} \cdot \overline{x_l}$). Therefore, the i^{th} moment for the service time of SR requests, for example, is obtained by taking the expectation of both sides:

$$\begin{aligned} \overline{x^2}_{SR} &= (x_s + x_l + x_t)^2 \\ &= \overline{x_s^2} + \overline{x_l^2} + \overline{x_t^2} + 2\overline{x_s} \cdot \overline{x_l} + 2\overline{x_s} \cdot \overline{x_t} + 2\overline{x_l} \cdot \overline{x_t} \end{aligned}$$

Higher moments of SWs and RMWs can be obtained similarly.

The moments for $VSR(M, k)$ requests are difficult to obtain analytically, therefore lightweight simulation is applied to estimate their first three moments.

2.8.2 Service Time for RAID

The costs of processing user level reads and writes on RAID5, RAID6, and RM2 disk arrays are linear functions of the costs for SR, SW, RMW, and VSR(M,k) operations. For a given mixture of reads and writes with frequency f_r and f_w, the overall cost is a weighted sum. An additional weighted sum is required to take into account different request sizes. Only one request size is considered in this study to simplify the discussion.

The cost functions can be used in estimating the volume of data to be transferred on I/O buses. For example, in a RAID5 disk array, an user level read incurs an SR request, while a logical write incurs two RMWs. Assuming that disks have XOR functionality, the data disk generates the update mask $d_{diff} = d_{new} \oplus d_{old}$, which is transmitted to the parity disk. The parity disk then computes and writes $p_{new} = d_{diff} \oplus p_{old}$. Given that the parity and data disks are on different buses, two data transfers are required on the "data" bus and one on the "parity" bus. A similar enumeration is possible when the data, parity, or both data and parity blocks are cached by the disk array controller.

Reads and writes in RAID0 result in disk level SRs and SWs, so that $\overline{x}_{disk} = f_r\overline{x}_{SR} + f_w\overline{x}_{SW}$ and the maximum throughput is $X_{RAID0} = N/\overline{x}_{disk}$. For the disk drive under consideration (IBM18ES), $\overline{x}_{SR} = 11.54$ ms and $\overline{x}_{SW} \approx \overline{x}_{SR}$, which yields $X_{RAID0} \approx 86$ accesses per second.

A read request in RAID5/0F (with no failures) corresponds to an SR, while a write requires two RMW accesses. The fractions of the types of requests are $f_{SR} = f_r/(f_r + 2f_w)$ and $f_{RMW} = 1 - f_{SR}$. The i^{th} moment of disk service time is $\overline{x^i}_{disk} = f_{SR}\overline{x^i}_{SR} + f_{RMW}\overline{x^i}_{RMW}$. The maximum throughput in this case is $X_{RAID5/0F} = N/\overline{x}_{disk}$.

In general, for a RAID scheme with given operation mode and fraction of reads (f_r), the maximum throughput of the array can be obtained by the following few steps: (1) check the corresponding cost table (Table 2.1, 2.2 and 2.3); (2) calculate the fractions for different types of primitive operations; (3) compute the mean service time $\overline{x}_{disk}$ as a weighted sum of those operations; (4) the maximum throughput is then obtained as $(N - i)/\overline{x}_{disk}$, where i is the number of failed disks.

2.8.3 Single Disk Mean Response Time Analysis

The analysis in this section is extended from the analysis described in [62]. The mean and second moment of waiting time for the M/G/1 model are [61]:

$$W = \frac{\lambda\overline{x^2_{disk}}}{2(1-\rho)} \qquad \overline{W^2} = 2W^2 + \frac{\lambda\overline{x^3_{disk}}}{3(1-\rho)}$$

where λ is the arrival rate and ρ is the disk utilization. The utilization $\rho = \lambda \overline{x}_{disk}$ should be less than one to ensure a non-saturated system. This formula applies to any of the disks, since their loads are balanced.

The mean and second moment of response time for read requests with FCFS scheduling policy are:

$$R_r = W + \overline{x}_{SR}\ , \qquad \overline{R_r^2} = \overline{W^2} + \overline{x_{SR}^2} + 2W \cdot \overline{x}_{SR}$$

When reads have a nonpreemptive priority over writes, then the mean response time for a read request is

$$\overline{R_r} = \overline{x}_{SR} + W_r$$

where

$$W_r = \frac{\lambda \overline{x_{disk}^2}}{2(1-\rho_r)}$$

ρ_r is the disk utilization due to read requests only, x_{disk} is the overall service time for a request. It can be observed from the equations that effectively read requests only compete against each other [61].

A complete list of the equations are shown in Appendix A. The validations for the above analyses are shown in Section 3.2.

2.8.4 Fork-Join Response Time Analysis

Estimating the fork-join response time is listed as a challenging problem in [15]. In fact, an approximate expression for the mean response time of n-way fork-join requests ($R_{\mathrm{F/J}}^{\langle n \rangle}$) was developed under Markovian assumptions (Poisson arrivals and exponential service times) in [45] and utilized in several RAID studies, e.g., [40]. Here an easy is described to compute approximate expression for n way fork-join response time $R_{\mathrm{F/J}}^{\langle n \rangle}$ for general service times. This approximate solution is validate it against simulation [24].

Reconstruction of a data block on a failed disk in RAID5/1F corresponds to an $N-1$ way fork-join. However, this is not a pure fork/join system since each disk processes its own requests, which interfere with fork/join requests. This non-pure fork-join response time is denoted as $R'^{\langle n \rangle}_{\mathrm{F/J}}$. When fork-join requests are processed in FCFS order together with interfering requests, the queue-lengths they encounter at each server tend to be more variable than the queue-lengths they would have encountered in a "pure" fork-join system. In effect the response times at different queues are more independent than they would be in a pure fork-join system.

It is known from [45] that $R^{\langle n \rangle}_{\mathrm{F/J}} \leq R^{\langle n \rangle}_{\mathrm{max}}$, where $R^{\langle n \rangle}_{\mathrm{max}}$ is the maximum of the response times of n requests constituting the fork-join request. Experimental results show that for the same overall utilization, the inequalities $R^{\langle n \rangle}_{\mathrm{F/J}} \leq R'^{\langle n \rangle}_{\mathrm{F/J}} \leq R^{\langle n \rangle}_{\mathrm{max}}$ hold in most cases. The second inequality is violated when interfering requests result in an overall service time which is much more highly variable than that of fork-join requests, but this is usually not true for RAID workloads. Therefore, it is reasonable to use $R'^{\langle n \rangle}_{\mathrm{F/J}} \approx R^{\langle n \rangle}_{\mathrm{max}}$, since it is unclear how to interpolate between the two bounds and $R^{\langle n \rangle}_{\mathrm{max}}$ is a tight upper bound.

Consider a fork-join request that involves reading of all surviving disks and XORing them as in RAID5 with one disk failure. To compute $R^{\langle n \rangle}_{\mathrm{max}}$ for those read requests, first the response times of the component SR requests are approximated with the extreme-value distribution. The reason of choosing extreme-value distribution is because its expected value of the maximum is easy to compute. The extreme-value distribution is given as

$$P[Y < y] = exp(-e^{\frac{-(y-a)}{b}})$$

with a mean $\overline{Y} = a + \gamma b$ and variance $\sigma_Y^2 = (\pi b)^2/6$ and

$$Y^n_{max} = (a + \gamma b) + bln(n)$$

where $\gamma \approx 0.57721$ is the Euler-Mascheroni constant.

By matching the mean R_r and the variance ($\sigma^2_{R_r} = \overline{R^2_r} - R^2_r$) of SR requests to the mean and variance of extreme-value distribution, the two parameters a and b can be computed easily. It follows that

$$R^{\langle n \rangle}_{\max} = R_r + \sqrt{6/\pi}\sigma_{R_r} ln(n)$$

Calibration against simulation results shows that $R'^{\langle n \rangle}_{\mathrm{F/J}}$ can be estimated more accurately by dividing the second term by 1.27 [24], i.e.

$$R'^{\langle n \rangle}_{\mathrm{F/J}} \approx R_r + \frac{\sqrt{6/\pi}\sigma_{R_r} ln(n)}{1.27} \quad \text{for FCFS}$$

In the case of SATF requests this coefficient is 1.10 [24], i.e.

$$R'^{\langle n \rangle}_{\mathrm{F/J}} \approx R_r + \frac{\sqrt{6/\pi}\sigma_{R_r} ln(n)}{1.10} \quad \text{for SATF}$$

This approximation is validated with simulation results, as reported in [24].

The 95^{th} percentile of read response time (in normal mode) can be expressed as $R^{95\%}_r \approx R_r + 2\sigma_{R_r}$. This formula can be derived for an exponential service time distribution. The response time distribution for M/M/1 with arrival rate λ and service rate $\mu > \lambda$ is also exponential: $R(t) = 1 - e^{-t/R}$, where $R = 1/(\mu - \lambda)$ is the mean response time. Setting $R(t) = 0.95$, then $R^{95\%} = ln(20)R \approx 3R = R + 2\sigma$, since the standard deviation of response time $\sigma = R$ when $R(t)$ is exponential. This approximation may be poor at lower arrival rates where σ follows the disk service time distribution (it has $c_v < 1$ in this case). However, $R(t)$ tends to an exponential distribution, regardless of service time distribution, since its coefficient of variation $c_v \rightarrow 1$ as $\rho \rightarrow 1$ [65].

CHAPTER 3

PERFORMANCE COMPARISON OF DOUBLE DISK FAILURE TOLERANT DISK ARRAYS

This chapter describes the simulation configuration and results. The simulation results are used to validate the analytical model described in Chapter 2 and to compare the performance for various disk array schemes that can tolerate double disk failures.

The chapter is organized as follows. In the first section the configuration of RAID systems under consideration is described. Next the validation of the analytical model is provided. Then, the maximum throughput and the mean response time for read requests are used to compare the performance of RAID6 and RM2 with each other and also RAID5 and RAID0 with FCFS scheduling policy. Finally, the response time with SATF policy is provided. Note that EVENODD performance is exactly the same as RAID6 in all modes, so that all the discussion and results for RAID6 apply to EVENODD as well.

3.1 Configuration

The total number of disks N is varied to investigate the scalability of the schemes. A fixed N implies the same normal mode maximum throughput in processing read requests, so that it is a fair comparison of the penalty of updating check disks. A small, intermediate, and large RAID configuration, are considered with $N = 7\,/\,19\,/\,39$ disks respectively, as shown in Table 3.1. The stripe unit size is 512 KB.

Table 3.1 Configurations Used in Comparison (The three numbers in each box correspond to the configurations with N =7, 19 and 39 disks.)

scheme	redundancy ratio (%)	parity group size	scheme parameter
RAID0	0 / 0 / 0	N/A	N/A
RAID5	14 / 5.3/ 2.6	7/19/39	N/A
RAID6	28 / 11 / 5.1	6/18/38	N/A
EVENODD	28 / 11 / 5.1	6/18/38	m=257
RM2	33 / 14 / 7.7	5/13/25	M=3/7/13

A parity group in RAID5 is a set of disks over which the parity is computed. Parity group (or more precisely redundancy group) sizes have a major impact on degrade mode performance. For RAID5 there is one parity group whose size is N. For RAID6, there are two groups of size $N - 1$, one associated with P and the other with Q. The parity group size for RM2 is $2M - 1 \leq (2N + 1)/3$ when N is odd, since $N \leq 3M - 2$ (see Section 2.7), i.e., it is smaller than that of RAID6, which favors the performance of RM2 versus RAID6 when both have single disk failures.

To summarize the workload model used in the simulation: arrivals are Poisson, requests are fixed size and are uniformly distributed over all available address space. Two cases when the fraction of reads $f_r = 0.75$ and $f_r = 0.50$ are considered.

3.2 Validation of Analytical Models

Since there are several approximations involved in the analytical model, a validation is necessary to show the accuracy of the approximations. A detailed disk array simulator called DASim[1] [63] is used to validate the analysis. The simulator takes into account

[1] DASim is developed by the Integrated System Lab of NJIT.

most hard drive details such as zoning, track and cylinder skews, spare cylinders and track aligned access. It uses disk specification extracted by the DixTrac project [55], which contains detailed disk geometric information as well as the seek time table.

The simulation is run with various array schemes and configurations shown in Table 3.1. The validation results with $N = 19$ for the mean read response time (R_r) are presented in Figures 3.1, 3.2, and 3.3 for the three modes with zero, one, and two disk failures, respectively. The result shows that the analysis describe in Chapter 2 is highly accurate for RAID0, RAID5, RAID6 (also EVENODD) and RM2 at all operating modes.

3.3 Performance Comparison with FCFS Policy

3.3.1 Normal Mode

The performance of a disk array is a function of the characteristics of disk drives constituting it, so that it is the relative rather than absolute values of throughput that are of interest.

The maximum throughput of a scheme is denoted as X_{scheme} with proper subscript. A good measure of the efficiency of a scheme operating in normal mode is the ratio of its maximum throughput (X_{scheme}) to that of RAID0, denoted as $E_{\text{scheme}} = X_{\text{scheme}}/X_{\text{RAID0}}$. It is observed that $E_{\text{scheme}} = 1$ for all schemes when the fraction of write $f_w = 0$. But due to the small write penalty, it decreases rapidly with increasing f_w and more parity units in a stripe.

E_{scheme} for RAID5, RAID6 and RM2 operating in normal mode are summarized in Table 3.2 using the throughput model described in Chapter 2. Note that these values are independent of N and apply to all three configurations. It follows that RAID6 and RM2 have the same performance in normal mode, while RAID5 performs

Table 3.2 Efficiency of RAID Schemes (E_{scheme}) in Normal Mode

scheme	75% read	50% read
RAID5	0.62	0.45
RAID6	0.49	0.32
RM2	0.49	0.32

better since it updates one check disk rather than two. Performance deteriorates rapidly at higher values of f_w, showing the high cost of parity updating. This is a justification for techniques to reduce the write penalty by aforementioned caching techniques or using log-structured arrays [41].

While X_{scheme} is a good indicator of the performance of a scheme, its mean read response time (R_r) is also important since it may be unacceptable at higher arrival rates. Figures 3.1, 3.2 and 3.3,are plots of R_r versus the arrival rate for $N = 19$, $f_r = 0.75$ with zero, one, and two disk failures. For applications that impose response time limits, the maximum throughput with respect to the highest acceptable response time (say 100 ms) can be estimated.

In Figure 3.4,the utilization factors are plotted for systems with zero, one and two failures. Since utilization is arrival-pattern independent, it can give a general comparison of the efficiency of different schemes. The lines are grouped into three bunches, corresponding to the operation mode with zero, one and two failures. The distance between the bunches shows the cost of recovery. Within a bunch, the slopes show the efficiency of different schemes: a lower slope means a higher throughput.

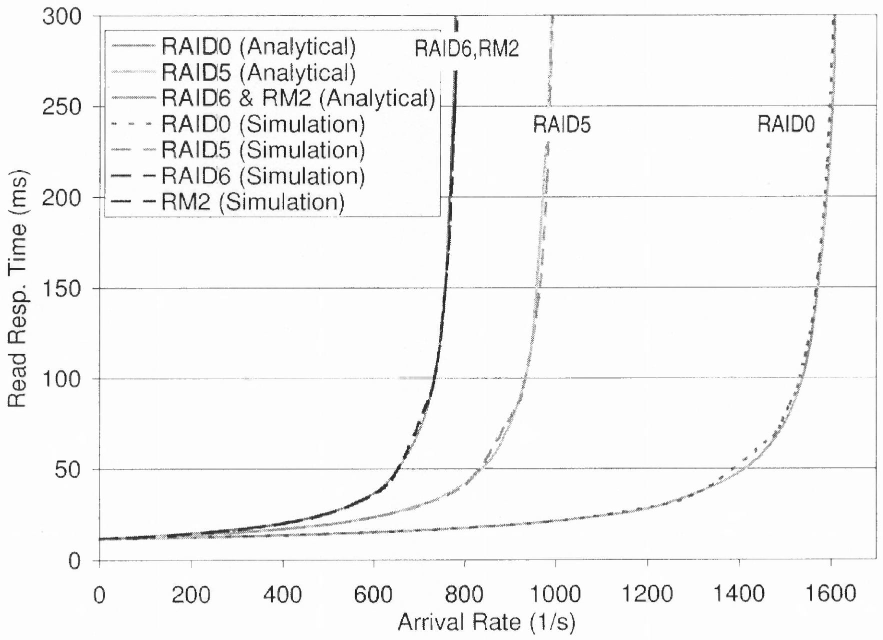


Figure 3.1 Mean read response time in normal mode, with $N = 19, f_r = 75\%$, and FCFS scheduling policy on each disk. Stripe unit size is 128KB.

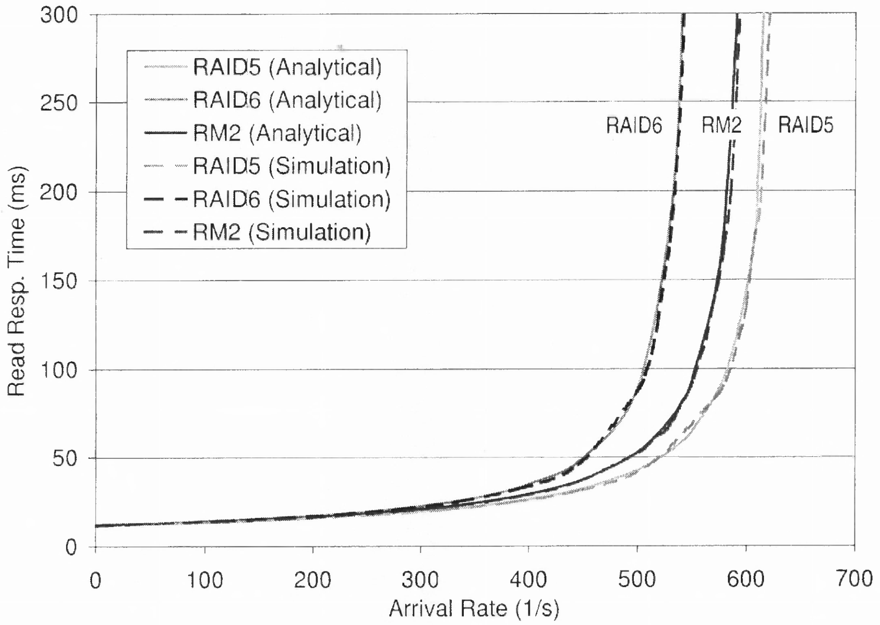


Figure 3.2 Mean read response time with one disk failure, $N = 19, f_r = 75\%$, and FCFS scheduling policy on each disk. Stripe unit size is 128KB.

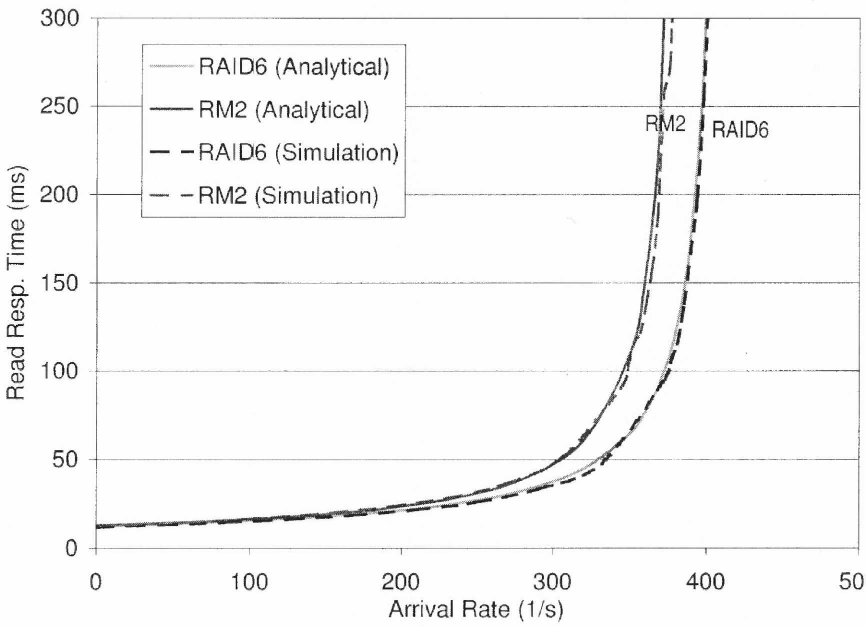


Figure 3.3 Mean read response time with two disk failures, $N = 19, f_r = 75\%$, and FCFS scheduling policy on each disk. Stripe unit size is 128KB.

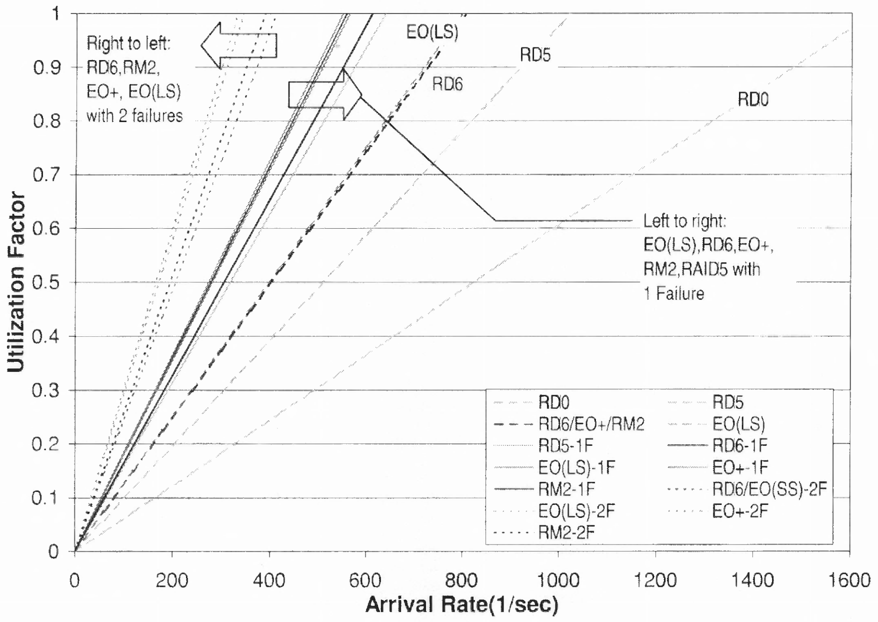


Figure 3.4 Mean disk utilization in normal mode, with $N = 19, f_r = 75\%$, and FCFS scheduling policy on each disk. Stripe unit size is 128KB.

3.3.2 Degraded Mode with One Disk Failure

With one disk failure the system performance deteriorates due to the overhead associated with reconstructing the data on the failed disk. The *performance degradation factor* is defined as the ratio of the maximum throughput with one (or two) disk failures to the maximum throughput with no disk failure. Since the maximum throughput for RAID6 and RM2 are the same in normal mode, this performance degradation factor is also a good indicator of their relative performance.

Table 3.3 gives the performance degradation factor for RAID5, RAID6 and RM2.

Table 3.3 Performance Degradation Factor with One and Two Disk Failures

f_r		75% read			50% read		
N		7	19	39	7	19	39
single failure	RAID5	0.65	0.63	0.63	0.73	0.71	0.69
	RAID6	0.74	0.69	0.68	0.82	0.78	0.77
	RM2	0.77	0.76	0.77	0.84	0.83	0.83
double failure	RAID6	0.53	0.51	0.51	0.64	0.62	0.61
	RM2	0.51	0.48	0.49	0.61	0.58	0.59

The loss of throughput with a single disk failure with respect to normal mode for all schemes is around 30% for $f_r = 0.75$ and 20% for $f_r = 0.5$. RM2 has the smallest throughput loss and outperforms RAID6. This is because RM2 is effectively a clustered RAID and its parity group size is smaller than RAID6 (see Table 3.1). An clustered RAID6 (as in [3]) can be appropriately configured to match RM2's performance with the same level of overhead.

The performance degradation for all schemes is affected very little by with increasing N, which implies their maximum throughputs (X_{scheme}) are almost linear functions of N. The reason is that the mean cost of operation with single disk failure divided by the number of disks is $O(1)$ for all schemes. For example, as in RAID5 array, a disk failure will double the load on each disk regardless of the number of disks in the array.

3.3.3 Degraded Mode with Two Disk Failures

When there are two failed disks, the performance degradation varies with the scheme as shown in Table 3.3. All schemes suffer a throughput loss of at least 36%. RM2 retains about 48-61%, while RAID6 and EVENODD retain 51-64% of their normal mode throughput.

RAID6 (also EVENODD) shows good scalability, since the mean cost of operation on each disk is $O(1)$. On the other hand, the performance loss for RM2 increases with the number of disks (N). This is due to its chain-like rebuild procedure, in which recovery paths of length $O(N)$ are involved. The details are discussed below.

RM2 uses a *chain-like* recovery procedure in the degraded mode with double disk failure, which means that the recovery of each block in the chain requires the recovery of the previous block (if any) plus accessing the remainder of its parity group (See Section 2.7 and Figure 2.11). This property leads to the reading of more than one block from a disk, which forms a VSR type basic operation (see Section 2.3). Therefore, the cost of operation in RM2 with double failures is inherently higher than that in RAID6.

Without going into detail, multiple blocks can be retrieved more efficiently if the stripe unit size is small, since they are more likely to be on the same cylinder/track.

As a result, an RM2 scheme with smaller stipe unit size can offer a higher maximum throughput. Multiple runs of simulation are used to investigate the impact of stripe unit size on maximum throughput. The configuration is $N = 19$ with $f_r = 0.75$ with various stripe unit sizes from 4KB to 1024KB. Throughputs normalized with respect to RAID6 are given in Table 3.4.

It is observed that RM2 performance is not affected by stripe unit size in normal and degraded mode with one disk failure. However, with two disk failures, its maximum throughput drops 16% when the stripe unit size increases from 4KB to 1024KB. At very small stripe unit size, the VSR operations are very likely equivalent to a simple read since multiple stripe units are on the same track. Therefore, the effect of declustering (i.e. smaller parity group size) overrides the effect of chain-like recovery pattern at very small stripe unit size and makes the maximum throughput of RM2 higher than that of RAID6.

However, in practice, it is desirable to keep the stripe unit size large so that the possibility that a request cross the boundary of a stripe unit is low enough. For example, imagine a stripe unit size of 4KB, and a 12KB write will span three stripe units. These three stipe units are covered by six parity groups. Therefore, six 4KB parity stripe units on six disks need to be updated, which is prohibitively expensive.

Table 3.4 The Impact of Stripe Unit Size on Maximum Throughput in RM2 (Value shown are $X_{\text{RM2}}/X_{\text{RAID6}}$ with different stripe unit sizes.)

Stripe Unit	4K	16K	64K	512K	1M
No failure	1	1	1	1	1
One failure	1.09	1.09	1.09	1.09	1.09
Two failures	1.08	1.04	0.98	0.94	0.92

3.4 Performance Comparison with SATF Policy

In this section, the simulation results for disk arrays with SATF scheduling policy on individual disks are reported. The mean read response times versus arrival rate with $N = 19$ and $f_r = 0.75$ are plotted in Figure 3.5, 3.6 and 3.7, for various RAID schemes with none, one and two disk failures, respectively. In Figure 3.4, the change of utilization versus arrival rate is shown for RAID5, RAID6 and RM2 running in normal mode.

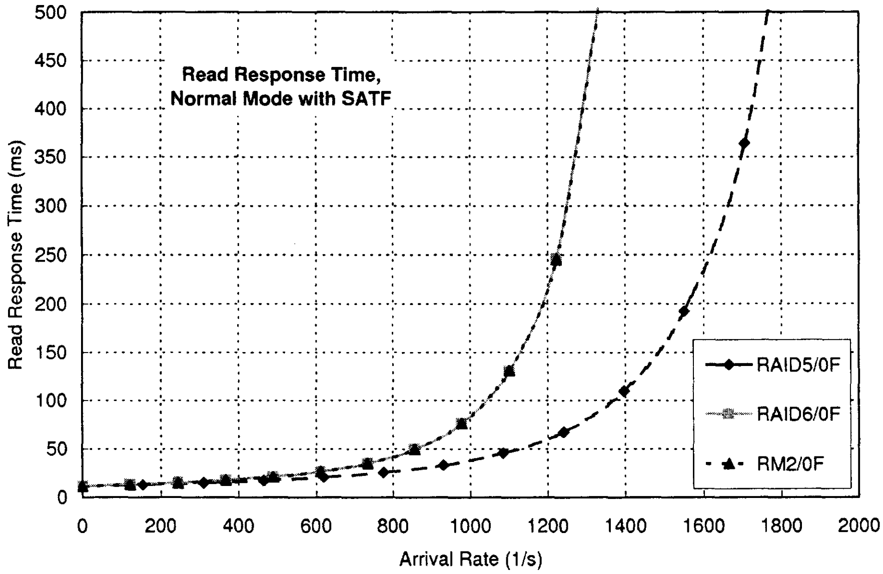


Figure 3.5 Mean read response time in normal mode, with $N = 19$,$f_r = 75\%$, and SATF scheduling policy on each disk. Stripe unit size is 128KB.

It can be observed that SATF can greatly increase the maximum throughput as well as decrease the response time. By comparing Figure 3.1 and Figure 3.5, it is observed that the throughput at a response time threshold of 300ms for all schemes have an improvement of at least 50% with SATF policy. When there are one disk failure, the comparison between Figure 3.2 and Figure 3.6 shows an improvement over 70%. When there are two disk failures, both RAID6 and RM2 benefit from SATF an improvement over 80% in throughput.

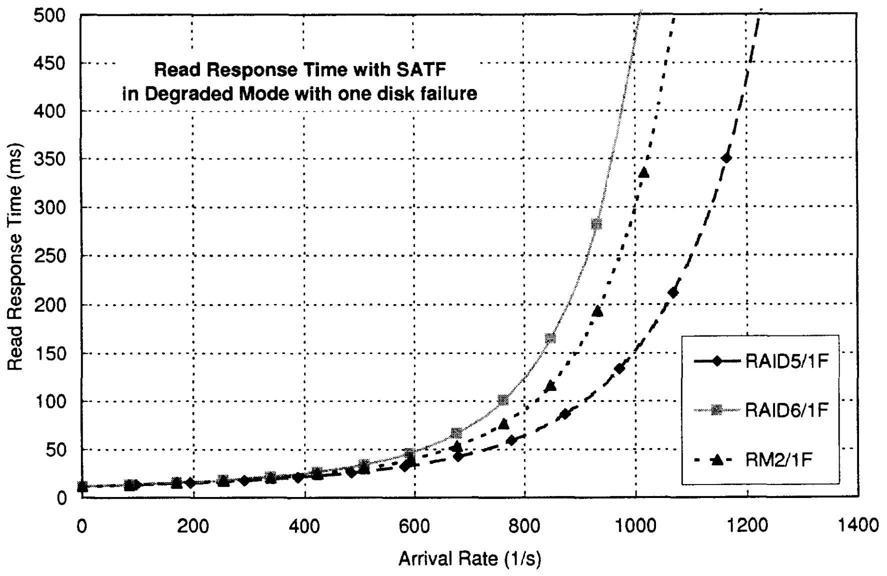


Figure 3.6 Mean read response time with one disk failure, $N = 19, f_r = 75\%$, and SATF scheduling policy on each disk. Stripe unit size is 128KB.

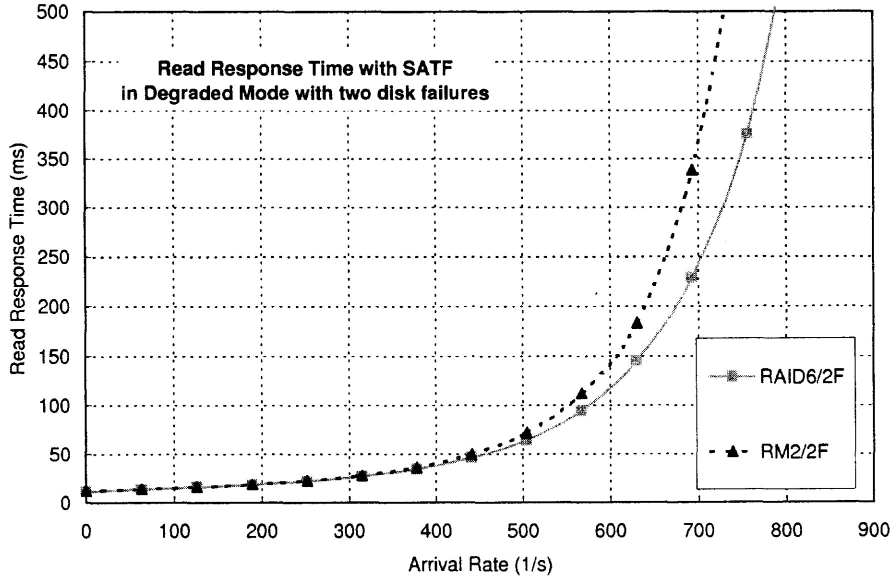


Figure 3.7 Mean read response time with two disk failures, $N = 19, f_r = 75\%$, and SATF scheduling policy on each disk. Stripe unit size is 128KB.

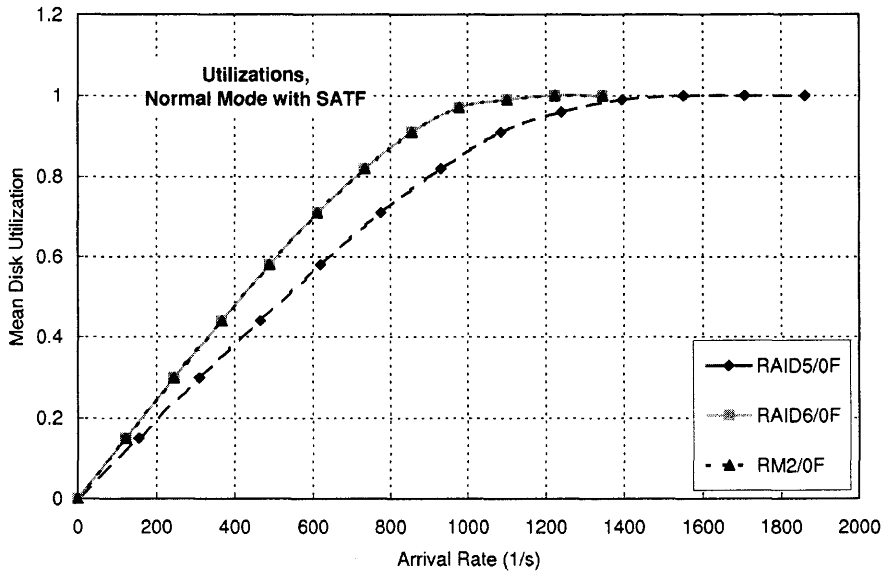


Figure 3.8 Mean disk utilization in normal mode, with $N = 19, f_r = 75\%$, and SATF scheduling policy on each disk. Stripe unit size is 128KB.

In a disk array that uses FCFS local scheduling policy, the response time tends to reach infinity at near saturate arrival rates. In contrast, in a disk array that uses SATF scheduling, the response time shows moderate increase even at very high arrival rates. In other words, the saturate arrival rate for SATF is well greater than the saturate arrival rate for FCFS. For example, as shown in Figure 3.5, the system is not saturated even when the system response time is already higher than 300 ms. Further simulation study has shown the saturate arrival rate with SATF scheduling is about double the saturate arrival rate with FCFS. The reason behind this is that SATF always select from the queue the request with lowest access time. When the arrival rate increase, the queue length increase as well. Therefore, the SATF have a larger set of candidate requests and thus can probably find a request with less access time than before. In effect, a longer queue helps reduce the mean service time for executing requests and therefore increases the throughput.

In Figure 3.8, the utilizations versus arrival rates are show for RAID5, RAID6 and RM2 running in normal mode. Comparing with Figure 3.4, the major difference is that the curves with SATF scheduling are not straight lines as with FCFS scheduling. This indicates the mean service time for requests decreases with higher arrival rates. In particular, the systems continue to yield increasing throughput after the disks are fully utilized. This means those extra throughput after the disks are 100% utilized should solely owe to the reduction in mean service time.

CHAPTER 4

ARCHITECTURE FOR THE HETEROGENEOUS DISK ARRAY

Based on the analysis and motivation in Section 1.4, the author proposes the Heterogeneous Disk Array (HDA) architecture with the following features:

- Allowing different disk models.
- Allowing multiple RAID levels to coexist in a single physical array.
- Utilizing available disk storage capacity to the maximum extent.
- Utilizing available disk access bandwidth to the maximum extent.
- Constant monitoring on the system performance to improve data allocation decisions and to make automatic load balancing possible.

The heterogeneity lies in both disk drive models and organizations, i.e., RAID levels. The heterogeneity in the device means that the disk array consists of disk drives of various capacity and access bandwidth. While in the array organization, different redundant data protection levels (RAID5, RAID6, RAID1/0) can be used simultaneously in the same disk array. Even for two datasets that use same RAID level (RAID5 or RAID6), their parity group sizes can be different.

In this chapter, the architecture of the proposed Heterogeneous Disk Array is described. The function and design of essential components are discussed. Especially, the data structures for metadata and the flow charts for processing requests are shown.

4.1 Heterogeneous Disk Array Architecture

The architecture for the heterogeneous disk array (HDA) system is shown in Figure 4.1. The system consists of an array controller and the underlying hard disks of various models. As the central component of the heterogeneous storage system, the array controller consists of the following parts:

Scheme Selector: chooses proper RAID level and parity group size for different applications;

Splitter: breaks a large allocation request to smaller requests;

Distributor: selects appropriate hard drive to store the data, also called **allocator** since it handles allocation requests;

System Directory: provides logical-to-physical address mapping and data-parity block mapping;

System Tuner: tunes the system by moving data blocks to achieve near-optimal balanced utilization of the hard disks. This is an optional module of the system;

Performance Monitor: monitors the utilized bandwidth and capacity on all disks as inputs to the distributor.

4.1.1 Request Types

There are three kinds of incoming requests for the heterogeneous disk array: *allocation requests*, *update requests*, and *read requests*. An allocation request is to create a new object, i.e., to assign some free space from the disk array to a new file. An update request, as implied by the name, is to update on the disk the data object (a file) that has been previously allocated. An allocation request may or may not be followed

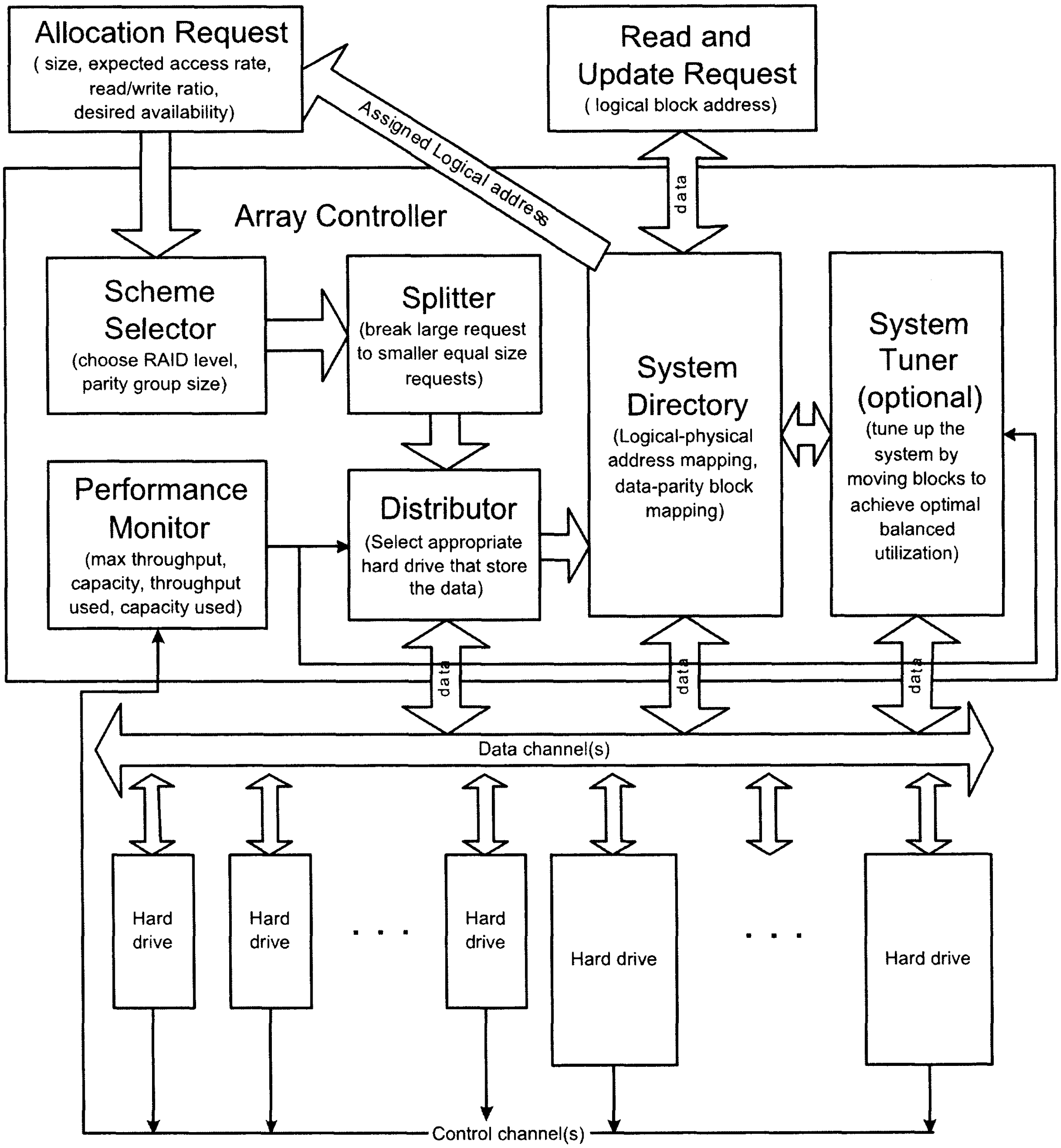

Allocation Request
(size, expected access rate, read/write ratio, desired availability)
Read and Update Request
(logical block address)
Assigned Logical address
data
Array Controller
Scheme Selector
(choose RAID level, parity group size)
Splitter
(break large request to smaller equal size requests)
System Directory
(Logical-physical address mapping, data-parity block mapping)
System Tuner (optional)
(tune up the system by moving blocks to achieve optimal balanced utilization)
Performance Monitor
(max throughput, capacity, throughput used, capacity used)
Distributor
(Select appropriate hard drive that store the data)
data
data
data
Data channel(s)
Hard drive
Hard drive
. . .
Hard drive
Hard drive
. . .
Hard drive
Control channel(s)

Figure 4.1 System architecture for heterogeneous disk array.

immediately by the writing of the file, which is an one time activity. A read request has its usual meaning – reading data from disk.

The read and update requests are simpler than allocation requests. The only parameter required for read and update requests is the logical address and the size of the read/update. The logical addresses of requests are translated into physical addresses by the *system directory.* The controller then checks those physical addresses and fetches or updates the data. For an update request, the *system directory* also checks whether there are any parity block(s) that need to be updated. If so, the parity block(s) is also updated.

For allocation requests, besides the size, the following parameters are required as well:

- Desired availability rating (i.e. reliability).
- Expected access rate in accesses per second.
- Expected read/write ratio for future accesses.

The first parameter, desired availability rating, will be used in deciding the RAID level for the data. There are various ways to specify the availability. For example, it can be specified quantitatively using MTTDL (Mean Time To Data Loss) or qualitatively, e.g., "very high availability required". Another simple approach is to *tag* the allocation request with a RAID level, i.e. let the user specify the RAID level. The next two parameters are related to the accessing pattern of the data object being allocated. These two parameters will be used in the allocation of the data blocks. More details will be given in Section 4.2 and Chapter 5.

4.1.2 Scheme Selector

When an allocation request comes to the array controller, it is first send to the *Scheme Selector*. If the availability of the request is specified in terms of MTTDL, the *Scheme Selector* module will select a suitable RAID level (i.e. RAID 0/1/5/6) based on its desired availability rating. This decision is made through the help of a reliability model. If the request is tagged with a RAID level, the scheme selector does nothing and passes it onto the next module.

The problem of reliability modeling is define as follows:

Reliability modeling Given a set of n disk drives with their Mean Time To Failure (MTTF): $\{m_1, m_2, \ldots, m_n\}$, and a target Mean Time To Data Loss (MTTDL), determine the level x, $x \in \{0, 1, 5, 6\}$, for RAID, and determine the number of disks $1 \leq g \leq n$ such that a RAIDx array consisting of any subset of g disks has MTTDL $\geq$ target MTTDL.■

Patterson et al. provided a simple expression for the MTTDL of a disk array [48] that can tolerate one failure:

$$MTTDL = MTTF_{RAID5} = \frac{(MTTF_{disk})^2}{N(G-1)MTTR_{disk}}$$

where N is the total number of disks in the array, G is the number of disks in a RAID group (i.e. a set of disks over which a parity is computed), $MTTF_{disk}$ is the mean time to failure of a component disk, $MTTR_{disk}$ is the mean time to repair of a component disk, which is in fact the rebuild time of a disk array. This model assumes that disk failure rates are identical, independent, and exponentially distributed random variables. In arrays that maintain one or more on-line spare disks, the repair time can be very short, usually less than an hour, and so that the MTTDL can be very long and exceeds the normal projected disk deployment intervals (5 years).

The reliability models for other redundant disk arrays have been studied in [26], [56] and [57].

4.1.3 Splitter

The purpose of the *splitter* is to break a big allocation request into smaller pieces so that each piece does exceed the size of a *relocation block* (see Section 4.2). These small pieces are called sub-allocation requests. Each sub-allocation request is handled separately or in a batch. Additional constraints may apply to this batch of allocation requests. For example, if the data are stored using the RAID5 scheme, it is desirable that the sub-allocation requests are sent to different disks.

4.1.4 Distributor

The *distributor* is the key component of the system. It takes a batch of equal-size blocks from the *splitter* and assign them to disks so that bandwidth and capacity utilizations for all disks are roughly equal and the constraints are met. The task of the *distributor* can be modeled as a vector scheduling problem with constraints. The details of this problem will be discussed in Chapter 5.

Since the distributor considers both bandwidth and capacity utilization, the throughput model described in Chapter 2 is used to estimate the utilization given the hard disk specifications and workload characteristics.

Because the allocation requests are handled by the distributor, the distributor is also called the *allocator*.

4.1.5 System Directory

While processing an allocation request, after deciding the placement of each allocation sub-request, the *distributor* outputs its decision to the *system directory.*

The *System Directory* component is a set of data structures and procedures that provide the following services:

1. Stores and retrieves the logical-to-physical address mapping information in an efficient way (in terms of both space and time).
2. Stores and retrieves the data-parity relations between blocks. In other words, given the logical address of a data block, it should return the logical address(es) of corresponding parity block(s), and vice versa.
3. Keeps track of the hotness of blocks for performance tuning.
4. Manages free space.

For the address mapping service, it is impossible to record every allocation request since there are too many requests and the mapping would be much too large. A multi-level indirection (like in virtual memory address mapping) is an attractive option.

In HP AutoRAID system [68], the disk storage is divided into small units called *Relocation Blocks* (RBs). The RBs are the basic units of migration in the system. A predetermined number of sequential RBs are combined into *Physical EXtents*(PEXes). Several PEXes (> 3) can be combined to make a *Physical Extent Group* (PEG). Then every PEG can be considered as a virtual disk array and be formatted to be RAID5, mirrored or remain unformatted. The system maintains a list of all RBs, each RB points to a PEG table to which it belongs. Every PEG has a PEG table, which holds list of RBs in PEG and list of PEXes used to store them. The logical structure of the address mapping is shown in Figure 4.2.

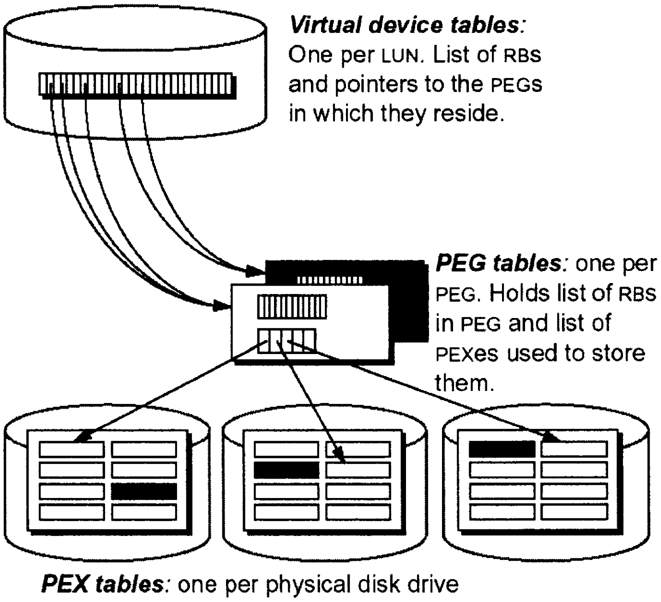

Virtual device tables: One per LUN. List of RBs and pointers to the PEGs in which they reside.
PEG tables: one per PEG. Holds list of RBs in PEG and list of PEXes used to store them.
PEX tables: one per physical disk drive

Figure 4.2 Structure of tables that map PEGs, PEXes and physical disk addresses in HP AutoRAID (extracted from [68]).

In AutoRAID, RBs are 64KB and PEXes are 1MB in size. Both are too small for current array configurations. In a 10TB array, 10M RB size will require roughly 20-30M memory for the address mapping, which fits well in the memory of an array controller and makes 10M a reasonable RB size.

It would be too much overhead to record the access rate of each block. Since the RB is the basic unit for data migration, it is natural to record the access rate of each RB.

Introducing RBs and PEGs into the system will also effect the way system handles allocation requests. Since RBs are the basic units of data migration, each RB should be formatted to a RAID level. The system directory should keep a list of partially-filled RBs, and try to fill these RBs first when handling allocation requests.

More details on the data structure and address mapping flowcharts are given in Section 4.2.

4.1.6 System Performance Tuner

The allocations are based on predicted access rates. Such prediction requires the extension of operation system functionality to record the mean access rate and pattern (i.e. read/write ratio) for data generated by a certain application. However, even with the help of operating system, it is very difficult, if not impossible, to give accurate prediction in a multiprogramming environment.

Furthermore, there are several constraints that should be satisfied when making allocations. For example, the data and parity blocks that are in the same stripe must be placed on different disks. These constrains make load balancing more difficult.

Therefore, it would be helpful to have a background process to tune the system when the system running at a low utilization. Such a process is represented as the *system tuner* component in Figure 4.1.

The *system performance tuner* takes disk utilization statistics (e.g. throughput and space utilization) and the access frequency for data as input, and balances the utilizations of all disk drives. Such balancing is performed by swapping data blocks between disk drives.

It has been shown in [69] that if there is an improvement involving n disks, there exists at least one improvement involving just two disks. In other words, the swap operation is all we need for the purpose of balancing. This should be interpreted as allowing the system to reduce the amount of neighborhood searching.

The *"disk cooling"* procedure described in [53, 54] is a greedy algorithm for load balancing. It tracks the *heat* associated with data blocks, computes the *temperature* of each disk, then relocate the hottest block from the hot disks, so that the number of blocks to be moved is minimized (See Section 1.5.1). The cooling process is triggered only when the temperature of hottest drive is higher than $1 + \delta$ the average temperature, where δ is a system parameter. The dynamic tracking of the heat of the blocks is implemented based on a moving average of the inter-arrival time of requests on the same block.

However, the disk cooling algorithm assumes homogeneous disk drives and does not consider the constraints introduced by redundancy schemes (e.g. data and parity cannot reside on the same disk).

In order to gather up-to-date information on the data access rate and device utilization, the *performance monitor* module constantly keep track of the device utilization while the *system directory* records the access rate for data blocks. This information is made available to the *performance tuner* to balance the load on disks.

Since data migrations are usually expensive operations, the system performance tuner runs in background and only when the system is idle or having light workload.

4.2 The Data Structure and Operations of System Directory

The system directory provides a transparent layer between logical addresses, which is visible to the user (OS), and physical addresses, which can be understood by devices. In traditional disk arrays, there are similar layers which are defined through algorithmic functions. For example, the device number for a logical address A in RAID5 with left symmetric layout can be calculated by $A/SU \bmod N$, where SU is stripe unit size and N is total number of disks.

In HDA, the layer is implemented by fully-associative mapping. This mapping offers great flexibility: the actual data blocks can be moved without affecting the user address space. In other words, the data migration is transparent to the user. This transparency is very important for automatic performance tuning, which is described in Section 4.1.6. However, there is a price to pay for the fully-associative mapping – the size of the mapping table. It is impractical to map every possible physical address to a logical address. In HDA, this mapping is toward *relocation blocks* (RBs). A relocation block is the smallest unit of data migration. Therefore, each relocation block has an entry in the system directory and has a corresponding logical address, which is its RB number.

Another important task for the *system directory* is to keep track of the actual access rate to data objects. The actual access rate will be the input information to the *system performance tuner* which is discussed in Section 4.1.6. Due to the volume of requests, it is impossible to record every access to every data block. Actually, since the relocation block is the smallest unit of data migration, only the aggregated access rate to a relocation block need to be recorded.

In this section, the detailed design of the system directory is described. The information stored in the system directory are crucial to the Heterogeneous Disk Array. It includes all the information about the format and data organization of the array and some statistical data. The aggregation of this information are named *"metadata"*, and the corresponding data structure are named *meta data structure* hereinafter.

This section is organized as follows: first the entities in the metadata are listed, then the frequent operations are analyzed. The address translation diagram, which is optimized according to the frequent operations, is then presented. This is followed by a list of the actual data structures used in HDA. The estimated meta information size and read/write operation flowcharts based on these data structure are discussed thereafter.

4.2.1 Addressable Entities in HDA

The addressable entities in the HDA are listed as follows:

1. **Device** A device is usually a physical disk drive, or more precisely, it is an array-controller-visible device. However, it can also be a disk array. Each device is identified by a unique *Logical Unit Number* (LUN). Each device has its own capacity and maximum throughput (bandwidth) and all blocks in the device are addressable by using a single integer value, which is referred to as *device_offset*.

2. **Relocation Block (RB)** A relocation block (RB) is a set of contiguous sectors in raw storage space. The size of an RB is a predetermined fixed number.

3. **Virtual Disk (VD)** A virtual disk (VD) is a predetermined number of contiguous RBs on a device. Here, "contiguous" means that they are physically one after another rather than their addresses are consecutive.

4. **Virtual Array (VA)** A virtual array (VA) consists of one or more VDs from different devices. A virtual array is formatted to use a certain RAID level.

5. **Data/Parity Block** Data blocks are accessible by user applications, while parity blocks store redundant information and are usually transparent to users. Both data and parity blocks are addressable.

These entities and their relationships are shown in Figure 4.3.

For each of the first four entities, there is an address associated with it. The addresses are consecutive integers starting from zero. The addresses for different entities are in different address space, i.e., there both RB number 0 and device number 0. The address spaces are not necessary to be linear, which means there can be holes in the address space. However, a linear address space for device, RB, VD and VA

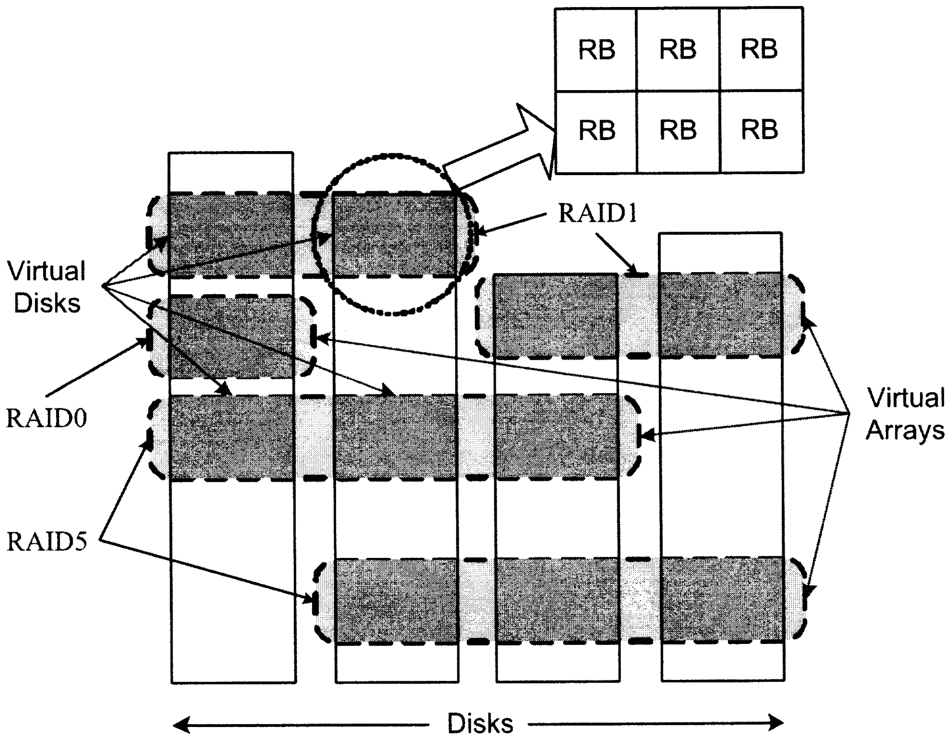


Figure 4.3 Entities in Heterogeneous Disk Array.

numbers can greatly simplify and speedup the table lookup procedure, as will be shown later.

For data/parity blocks, there are two addresses associated with each data block – physical address and HDA address. The HDA address is a logical address that is visible to users. The physical address is the address that can be understood by underlying devices. Both of them are compound addresses, as described below:

Physical Address A physical address is a *(LUN, device_offset)* pair, where LUN is the device# for a disk drive. Every physical address uniquely locates a block in the raw storage space of HDA. This block may stores user data or redundancy info. Therefore, it might not be addressable in user storage space.

HDA Address HDA addresses are visible to the file system. It is the counterpart of physical address in user storage space. An HDA address is an *(RB#, RB_offset)* pair, where RB_offset is the offset from the beginning of the RB.

4.2.2 Frequent Operations

The most frequent operation for a HDA is read operation. The next frequent operations are writing and allocation. Therefore, the most frequent address mapping is the translation from an HDA address to a physical address. To make this translation fast and efficient, a direct mapping from RB# to physical address should be maintained in the HDA controller.

There are various operations (e.g. read, write, allocation, rebuild) in a disk array. Each operation requires some kind of address translation. Those address translations are listed below.

1. **HDA address to physical address.** The read/write are the most frequent operations. Both operations arrive with the target HDA address as its parameter. The HDA address should be translated into corresponding physical address very efficiently. An HDA address consists of an RB# and a offset. Therefore, a fast and efficient mapping from RB# to physical address should be maintained in the controller.

2. **Scheme lookup.** Given an HDA address, the array controller should be able to determine what RAID scheme was applied on this block. This translation is used while processing write requests or rebuild requests in case of disk failure.

3. **Buddy lookup.** For write requests, if the target block is protected by one or more redundant blocks, those check blocks also need to be updated. Therefore, the array controller should be able to locate the corresponding check block(s)

given a data block. Certainly, buddy lookup should be preceded by a scheme lookup to determine the number of check blocks.

4. **Physical address to VA information.** When one of the disks fails, a rebuild process starts to recover all protected data blocks on the failed disk. Therefore, the controller should be able to tell which virtual array does a physical block belong to, and determine the addresses for its buddy blocks.

5. **Create new VA from free space.** This happens when the system need more space for a given RAID scheme. Please note that an HDA creates VAs as needed rather than pre-partitions raw disk space into different VAs.

6. **Enumerate all VAs for a given RAID scheme.** The array controller should be able to enumerate all the VAs for a given RAID scheme. This happens when the system manager queries array configuration or a backup software needs to copy all of the data for a given RAID scheme.

7. **Relocate RBs.** The physical location of an RB can be moved by load balancing process. The mapping between RB# and physical address need to be changed accordingly.

4.2.3 Address Translation Diagram

Based on the previous discussion, the address mapping diagram is defined in Figure 4.4. This diagram is optimized to provide fast address translations for frequent operations while having enough flexibility and requiring moderate storage. The redundancy in the metadata is reduced to the minimum in order to reduce the cost for maintaining the integrity for metadata.

The addresses of entities are translated by following the arrows in the address translation diagram. The ratios above links means whether it is a one-to-one or many-

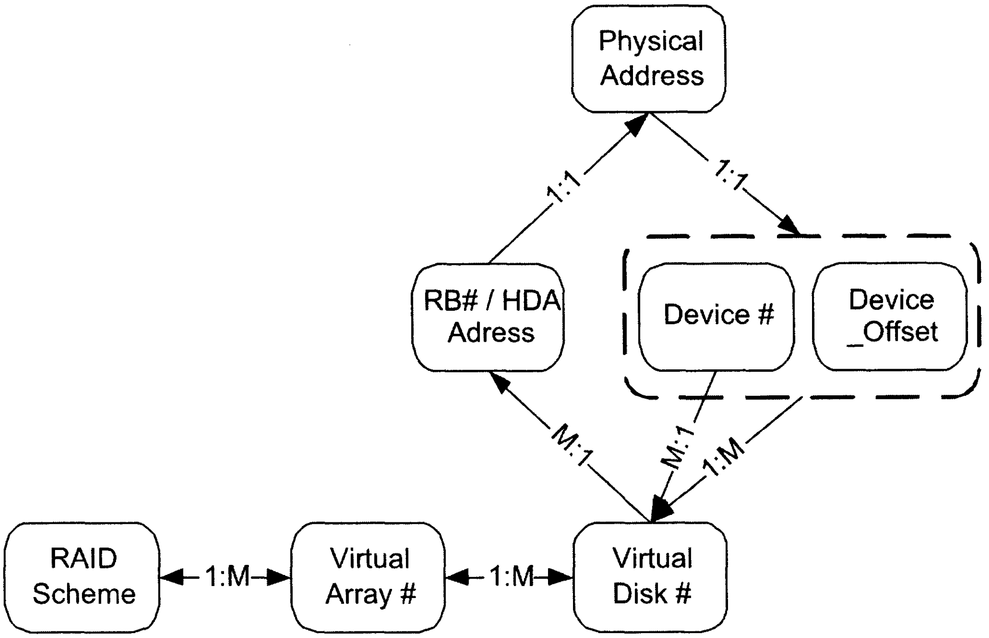


Figure 4.4 Address mapping diagram in HDA. The address entities can be translated following the arrows in the diagram. The ratios on the links means whether it is a many-to-one, one-to-many, or one-to-one relationship.

to-one or one-to-many relationship. For example, by following the one-to-many link between **Device#** and **Virtual Disk#**, we can enumerate all the virtual disks that resides on a given physical device.

The links are directed, which means translations can only be done following the arrow. For example, and **RB#** can be translated directly into a **Physical Address**, but no direct translation exist from **RB#** to **Virtual Disk#**. However, such translation can be done indirectly by following the links: **RB#** $\xrightarrow{1:1}$ **Physical Address** $\xrightarrow{1:1}$ (**Device#**, **Device_offset**) pair $\xrightarrow{M:1}$ **Virtual Disk#**. Some translations are bi-directional, which are equivalent to two separate links with opposite directions. The mappings can be done only in one direction because the mapping is indexed to allow efficient lookup.

4.2.4 The Data Structure for Meta Information

A simplified illustration for the tables maintained in the HDA are shown in Figure 4.5. There are four kinds of tables. Their data structures and relationships are as follows:

4.2.4.1 Table of RBs. There is one global table of RBs in the HDA. It stores the mapping from RB# to physical address and the heat index from the RB. The data structure for a row in the table is shown below:

RB#	Device#	Device_offset	Heat index

This table is indexed by RB#, which is indicated by underlining. The Device# and Device_offset make a physical address pair. The Heat index records the access rate for the blocks in this RB.

If the RB# is sequential, which means it starts from zero and grows without having holes, the field RB# is implied by table subscript can therefore be omitted to save space.

4.2.4.2 Virtual Disk Table. Virtual disks (VDs) are fixed sized consecutive disk spaces on physical devices. Each VD contains a fixed number (RB_PER_VD) of RBs. The RB#'s in a VD may not be consecutive as a result of background workload balancing. Each VD can be a member of a certain Virtual Array, or marked as unformatted space.

There is a VD table for each physical device in the HDA. A record in the VD table consists of the following fields.

VD# = (Device#, index in device)	VA#	RB$_1$	RB$_2$	$\cdots$	RB$_{\mathtt{RB\_PER\_VD}}$

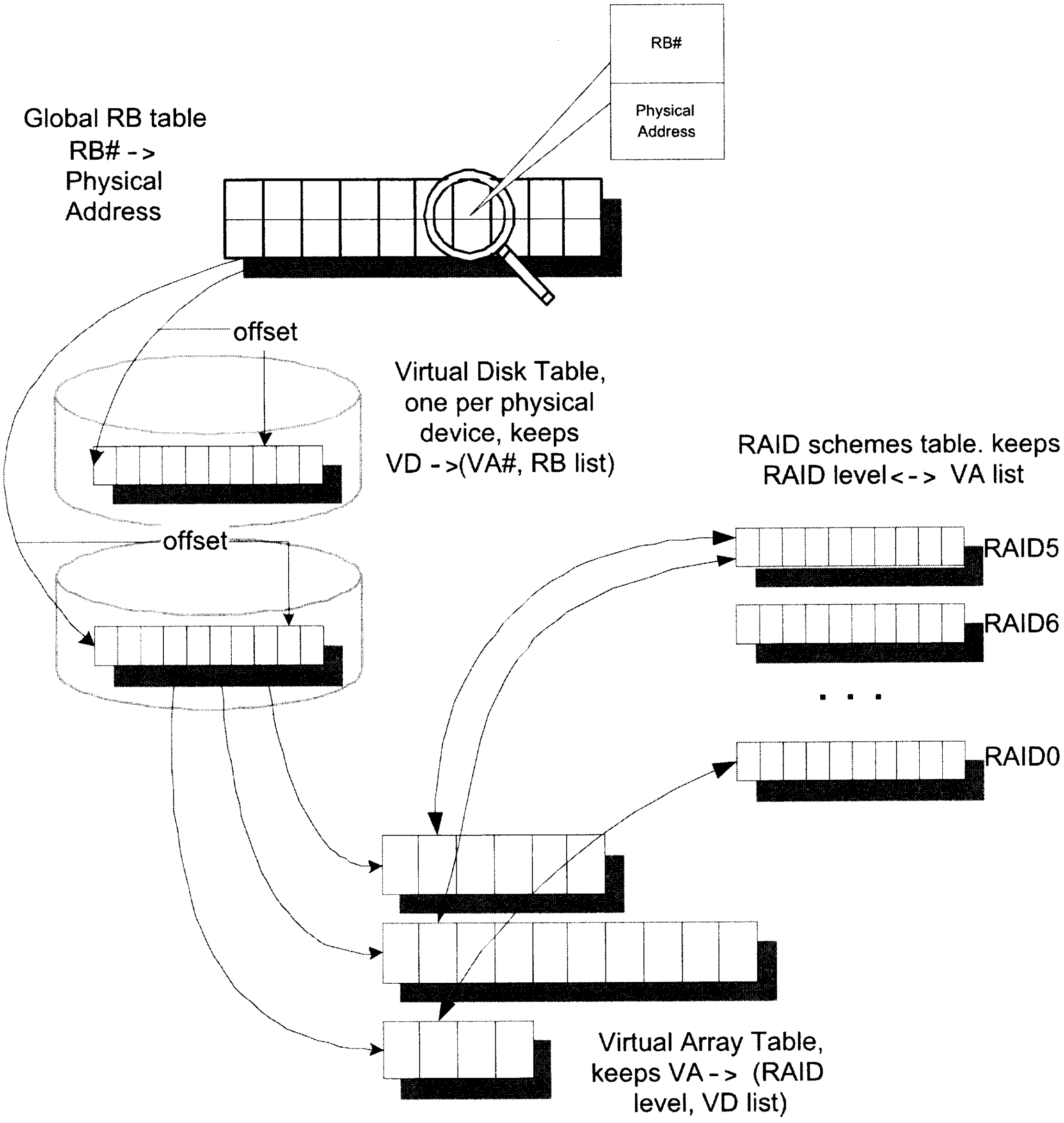

RB#
Physical Address
Global RB table RB# -> Physical Address
offset
Virtual Disk Table, one per physical device, keeps VD ->(VA#, RB list)
offset
RAID schemes table. keeps RAID level<-> VA list
RAID5
RAID6
. . .
RAID0
Virtual Array Table, keeps VA-> (RAID level, VD list)

Figure 4.5 Tables maintained in HDA.

VD# consist of Device# and an index of the VD in that device. Since the VDs have fixed size, a VD# can be directly mapped to a physical address simply by multiplying the VD size with the index.

The VD table is sorted on VD#. VA# is the id for the VA to which VA the VD belongs. RB1, ..., $\text{RB}_{\text{RB\_PER\_VD}}$ are the list of RBs in this VD. Since RB_PER_VD is a predefined constant, the VD table entries are of fixed size as well. As before, VD# is implied by table subscript and can be omitted.

4.2.4.3 Virtual Array Table. The virtual array table is a global table. It keeps all information about the organization of the virtual arrays, including RAID scheme, component virtual disks, parity location etc. The data fields for a record in VA table is as follows:

VA#	RAID scheme	Num VDs (=k)	VD_1	VD_2	$\cdots$	VD_k

VD1, VD2, ..., VDk are the list of VDs that constitute this virtual array. The parity VD(s) are listed first. For example, in a RAID5 (or more precisely, RAID4) virtual array, VD1 is the virtual parity disk. In RAID6 (without parity rotating), VD1 and VD2 stores parity P and Q, respectively. Since each virtual array can have different number of virtual disks, the length of the records are variable. An index is made on VA# for fast locating the corresponding record.

4.2.4.4 RAID scheme Table. The global RAID scheme table stores all the VAs for a certain RAID level in the HDA. The fields for each record in the table are shown below:

RAID scheme	Num VAs	VA_1	VA_2	VA_3	$\cdots$

The records are variable length. An index is made on RAID scheme.

4.2.5 Estimated Meta Data Size

In this subsection, the field width for the data structures discussed in previous section are decided. Then the meta data size is estimated.

The first decision to made is the field width for RB#. Initially an RB size of 10 MB is assumed, which is a relatively small chunk that can be moved easily. A field width of 4 bytes can support up to a raw storage of 40 PB, which is adequate since it can support a disk array consisting of 200,000 disk drives with each disk having 200GB. The width for other data fields can be defined similarly and are summarized in Table 4.1.

Table 4.1 Width for Data Fields in HDA

Data field	width (bytes)	support up to
RB#	4	array capacity 40 PB
Device#	2	65536 devices
Device_offset	4	single device of 2048 GB
Heat index	6	enough precision for a float number
VD# = (Device# + index)	(2+2) = 4	single device of 640 × RB_PER_VD GB
VA#	4	4G virtual arrays
RAID scheme	1	255 RAID schemes

For a mid-size disk array of 100TB raw capacity, the sizes for each table and also the total meta info size are listed in Table 4.2. RB_PER_VD = 8 is assumed in the calculation. It is shown that the meta information requires about 180MB memory space, which is reasonable for a 100TB array and can be kept in memory for efficient access.

Table 4.2 Memory Requirement for Tables in HDA

Entry in Table:	size per entry (bytes)	est. num of entries	table size
the RB table	$2 + 4 + 6 = 12$	100T/10M = 10M	120MB
the VD table	$4 + 4 \times 8 = 36$	10M/8	45MB
the VA table	$4 + 4 \times$ (avg. VD per VA)	1.25M/(VD per VA)	$<$ 10MB
RAID schemes	$4 + 4 \times$ (VA in scheme)	# of RAID schemes	$<$ 5MB
Total size of meta info for a 100TB array with RB_PER_VD = 8			180 MB

4.2.6 Read and Write Operation Procedure

The flowchart for the address translation in read and write operations are shown below. To simplify the discussion, only the operations in normal mode is discussed and the write operation assumes RAID5. Operations in degraded mode and/or in other RAID schemes are similar.

4.2.6.1 Flow Chart for Read Operation. Read is the most frequent operation in HDA. For a read operation, the HDA address need to be translated into a physical address. The steps required for such translation are shown in Figure 4.6.

The read operation has two parameters – target HDA address and request size. The HDA address consists of two parts: RB number and offset within the RB. The RB number is looked up in the RB table to find out its physical address, which is a (device number, device offset) pair. This device offset gives the physical address for the starting point of the RB. When added by the offset within RB, the physical address for the target block is found. The request is passed down to the device without any change. To sum up, the translation from HDA address to physical address requires only one table lookup and one addition. Note that since RB number is a sequential integer starting from zero, a lookup in the RB table is equivalent to

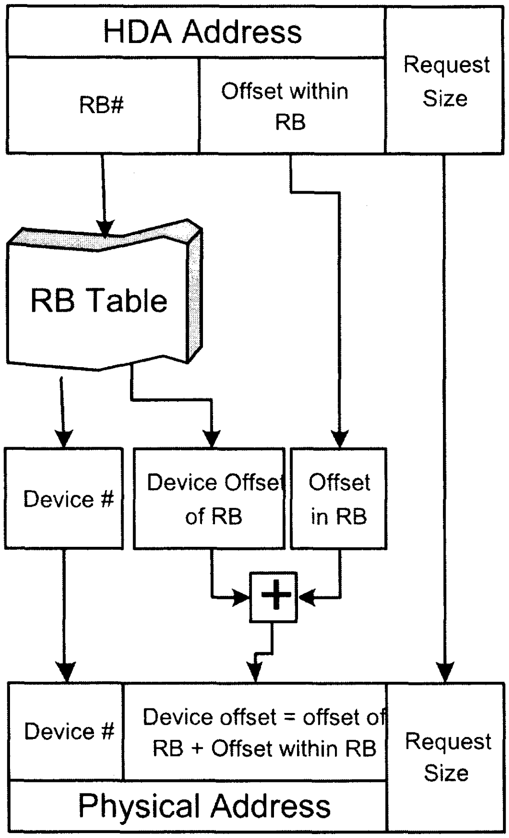


Figure 4.6 Flow chart of address translation for a read operation.

fetching an element from a large array, which requires only one multiplication (for element address) and one memory access.

4.2.6.2 Flow Chart for Write Operation. Figure 4.7 illustrates the address translation procedure to process a write request. RAID5 scheme is assume in this flow chart. The input HDA logical address will be mapped into two physical address: one for the data block, the other for the parity block. For other scheme such as RAID6, similar procedures can be define to give the physical address for both parity blocks.

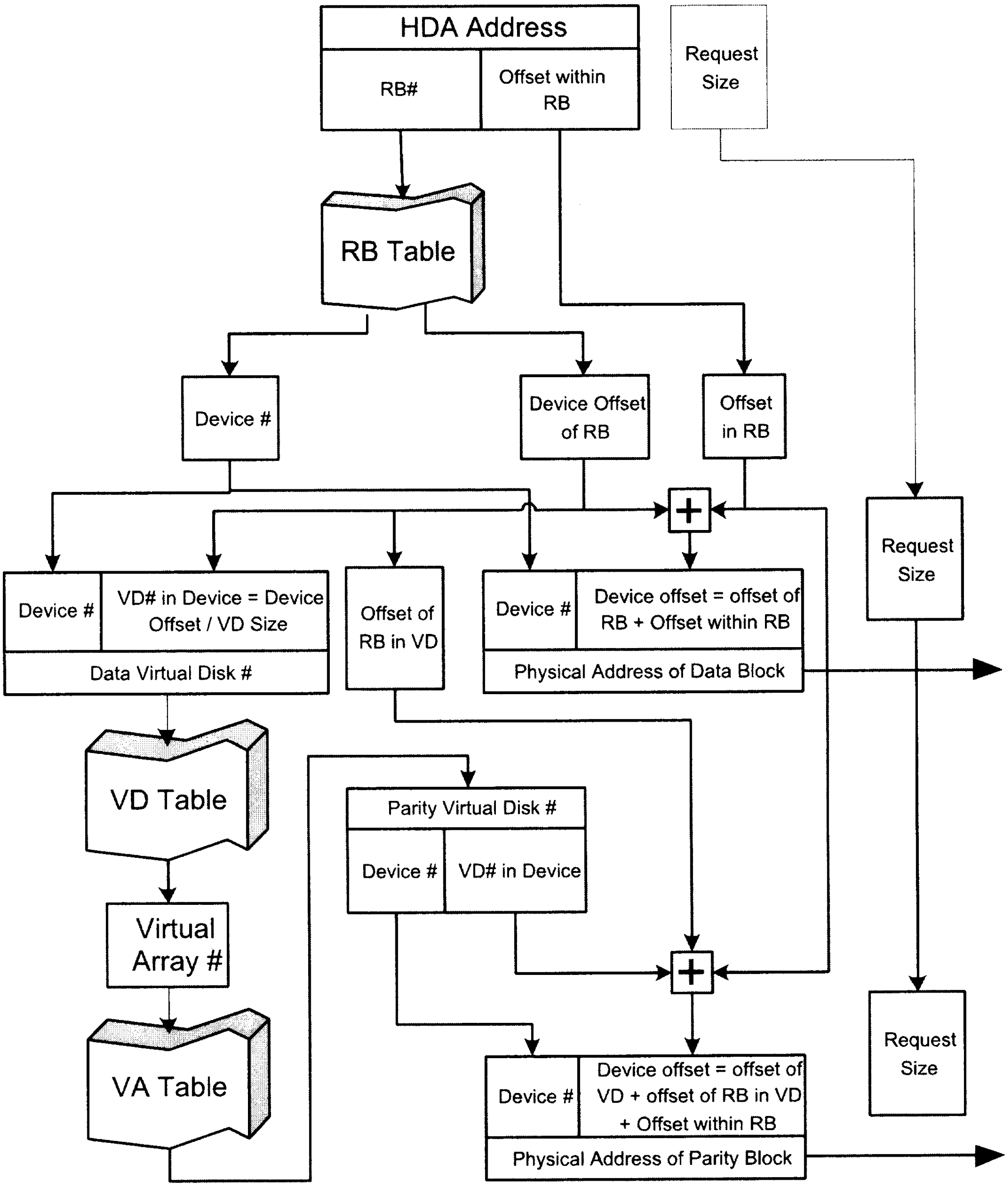


Figure 4.7 Flow chart of address translation for a write operation. RAID5 is assumed. The input is the HDA address of the block to be written. The outputs are the physical address of the data block and corresponding parity block. The request size remain unchanged. Large requests that spans multiple RBs are divided into smaller pieces (not shown) so that each piece is within one RB.

The physical address for the data block is generated in the same way as in processing a read request. The RB table is looked up to get the physical address for the starting point of the RB. The device offset of RB is broken into two parts: a VD index within device and RB index within VD. This is possible because the VD contains a fixed number of RBs and RBs have fixed size. Therefore, the VD index within device equals the device offset divided by VD size, while the RB index within VD is the remainder. Coupled with the device number, the VD index in device becomes a VD number on which the data block resides. By looking up in the VD table, the VA# for the VA of which the VD is a component can then be obtained. By looking up the VA table, the VD# for the parity virtual disk can be obtained. The physical address for a given VD# can easily be obtained by multiplying the VD index by VD size. The final physical address of the parity block then consists of the device number copied from parity VD# and a device offset which is the sum of offset of VD in device, offset of RB in VD and offset within RB.

CHAPTER 5

ALLOCATIONS IN HETEROGENEOUS DISK ARRAY

The allocator is the key component of HDA. It handles all allocation requests and manages free space. It decides the device from which the space is allocated for the allocation request. If there is not enough free space in the virtual array, a new virtual array of the desired scheme is created. In this case, the allocator also need to determine the subset of devices from which the new virtual array will be created.

As has been discussed in Section 1.4, one challenge in heterogeneous disk array is how to utilize both capacity and bandwidth to the maximum extent. In this chapter, the problem is formalized and one possible solution is described. This solution is then verified in a simplified environment to show its effectiveness and robustness. In Section 5.4, more discussions on the constraints on allocation are given. Finally, the actual allocation algorithms used in the performance study of HDA (see Chapter 6) are described.

5.1 Problem Analysis and Formalization

Both disks and allocation requests are modeled as 2-dimensional vectors. For an allocation request, the first dimension is the access rate, and the second dimension is size. For a disk drive, the two dimensions are maximum throughput (accesses per second) and capacity. As shown in Figure 5.1, disk allocation can be modeled by adding request vectors and making sure that their sum is less than the disk vector in any coordinate.

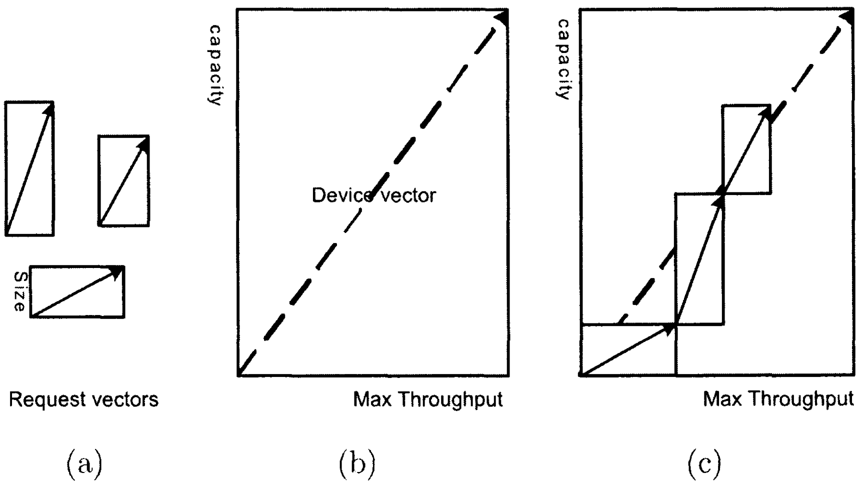


Figure 5.1 Allocation is modeled as vector sum. (a) is sample request vectors, (b) is a sample device vector, (c) shows the disk allocation is modeled as vector sum

The problem of balancing the utilization in terms of both throughput and capacity is defined as follows:

Problem Definition In a set D of n disks, the ith disk is represented by a vector $d_i = (X_i, C_i)$, where X_i denotes the maximum throughput and C_i denotes the capacity of that disk. Given a set J of allocation requests, each request is represented by a 2-dimensional vector $p_j = (x_j, c_j)$, where x_j is its anticipated access rate, c_j is the size of data. A valid solution is a partition of J into n subsets $J_1, \ldots, J_n$ such that the sum of both dimensions in a subset does not exceed the corresponding limit, i.e., satisfying both $\sum_{j \in J_i} x_j \leq X_i$ and $\sum_{j \in J_i} c_j \leq C_i$ for all $1 \leq i \leq n$.

Let

$$U_i^x = \frac{\sum_{j \in J_i} x_j}{X_i}$$

denotes the utilization (of throughput) of device i, and

$$U_i^c = \frac{\sum_{j \in J_i} c_j}{C_i}$$

denotes the utilization of capacity of device i. Two possible objective functions are:

1. minimize

$$F_1 = \max_{1 \leq i \leq n} \{U_i^x, \alpha U_i^c\} \tag{5.1}$$

or

2. minimize

$$F_2 = \mathrm{Var}\{U_i^x\} + \alpha \, \mathrm{Var}\{U_i^c\}, \tag{5.2}$$

where $\alpha \geq 0$ is a user defined emphasis factor, $\mathrm{Var}\{x_i\}$ is the variance over a set of number $\{ \, x_i \mid 1 \leq i \leq n\}$, i.e.:

$$\mathrm{Var}\{x_i\} = \frac{\sum_{1 \leq i \leq n} x_i^2}{n} - \left(\frac{\sum_{1 \leq i \leq n} x_i}{n} \right)^2$$

■

The value of α is usually chosen between 0 and 1 to emphasize more on the throughput, because balanced throughputs are more important than balanced disk capacities, and more often system bottleneck is throughput rather than capacity.

Although many storage system design and optimization tools choose to balance the response time [2, 5, 69, 53], the optimization target in HDA is throughput because of the following considerations:

1. Most applications do not impose stringent limits on I/O response time. Typical disk service time for a read request is about ten milliseconds for modern disks. This implies that the mean response time is reasonable, as long as the utilization of the disk is not very high, e.g., below 90%.

2. By giving higher priority to requests that are more urgent, requests that require lower response time can be satisfied.

3. Response times for requests which are processed by the same disk interfere with each other, because of their competition for disk access. Since our system is

dynamic and the RAID levels and data layouts are determined at *runtime*, a placement that can satisfy the response time requirement currently may not be able to satisfy the requirement later. New data may be placed on the same disk, introducing additional requests and therefore the response time will grow.

4. Although disk utilization is not strictly additive (depends on disk scheduling policy), it roughly satisfies the triangle inequality. For example, workloads A and B incurring a disk utilizations equal to 0.1 and 0.2 on separate disks are expected to result in a disk utilization less than or equal to 0.3, with an appropriate disk scheduling policy, such as SATF. This property enables the placement problem to be modeled as a bin packing problem.

5. If the response time is balanced over all disks, the faster disks will have higher utilization. Consider a RAID5 system processing only read requests, the load on all surviving disks is doubled when a single disk fails. If the faster disks have higher utilization, such doubling may lead to overloading, and the response time on that disk becomes infinite.

Possible solutions to this problem can be classified into *off-line* and *on-line* algorithms. In an *off-line* algorithm, the full knowledge of all items are given before the algorithm starts. In contrast, an *on-line* algorithm assigns every item p_i solely on the basis of the item's own information and system statistical data, without any information on subsequent items. The decisions of an on-line algorithm are irrevocable – assigned items can not be reassigned to other device or such re-assignment would incur high cost.

Although the problem is defined as an off-line problem, both on-line and off-line heuristic solution for the problem are required. The off-line algorithm will be used by the *system tuner*. The on-line algorithm will be used by the *allocator*.

It is easy to see that the problem is NP-hard [25], which means it is quite impossible to find an optimal solution in polynomial time. Therefore it is more practical to seek a low-cost heuristic algorithms to find acceptable solution in linear time.

This problem belongs to a generalized multi-dimensional variable sized vector scheduling problem [14]. There are numerous studies on related problems since 1970's, including variable sized bin packing, vector packing and vector scheduling. However, to the best of our knowledge, there is no existing online algorithm that solve the problem at hand satisfactorily.

5.2 A Solution Based on A Greedy Heuristic

In a typical storage system, each block takes a very small fraction of throughput and capacity. This can be mapped to a bin packing problem with small items, in which a better asymptotic ratio is often possible.

Greedy algorithms are natural candidates for finding approximate solutions to variable size vector scheduling problems. The best-fit heuristic is extended to the two-dimensional vector scheduling problem: an incoming request is assigned to the kth device so that the target function is minimized. The algorithm is given in Figure 5.2.

Since the first objective F_1 uses the max function, there may yield multiple choices of disk number with the same objective value. In this case, one of the choices is selected randomly. For the objective function F_2, such situation will rarely happen.

Algorithm BESTFIT_SCHEDULE(U_i^x, U_i^c, p)

Input: The utilization of throughput (U_i^x) and capacity (U_i^c) for device i, $1 \leq i \leq n$; the expected throughput (x) and capacity (c) consumption of the new item, represented as a vector $p = (x, c)$.

Output: The kth device, $1 \leq k \leq n$, to which the new item should be assigned to.

1. Denote the value of target functions with the new item assigned to the ith device as $f(i)$, $1 \leq i \leq n$.
2. Compute $f(i)$ for all $1 \leq i \leq n$, using a throughput model described in Chapter 2. If a device does not have enough space or bandwidth for the new item, which means utilization is higher than one for either capacity or throughput, remove that device from the set of possible answers.
3. Select k such that $f(k) \leq f(i), 1 \leq i \leq n$. If there are multiple choices, select one randomly.
4. Return k.

Figure 5.2 Best-fit schedule algorithm for allocator.

5.3 Verifying the Best-fit Allocation Algorithm

In this section, a synthetic workload is used to verify the effectiveness and robustness of the best-fit heuristic described in Section 5.2. Effectiveness means that the heuristic can provide a better solution than some other algorithms (described below). Robustness means that the algorithm can yield a reasonable solution with imprecise estimation of access rate.

5.3.1 Experiment Parameters

Three hard drives with different capacities and bandwidth are used in the experiment. The models and their specifications are shown in Table 5.1.

Table 5.1 Specifications of Hard Drives Used in the Preliminary Experiment

Manufacturer	Model	Capacity	Max Throughput (access per second)	RPM (1/min)	Year
IBM	IBM18ES	8.6 G	88.01	7200	1998
SEAGATE	Barracuda	2.0 G	74.53	7200	1997
SEAGATE	Cheetah4LP	4.2 G	91.63	10033	1996

The other allocation strategies used in the comparisons are:

Round robin To place the allocation request onto one of the hard drives in a round robin manner.

Random To place the allocation request onto a random hard drive

Proportional to throughput To place the allocation request onto a hard drive with probability proportional to its maximum throughput (i.e. bandwidth). In other words, the probability of putting a request on disk i is

$$p_i = \frac{x_i}{\sum_{1 \le j \le n} x_j}$$

where x_i is the bandwidth of disk i and n the total number of disks.

Proportional to capacity To place the allocation request onto a hard drive with probability proportional to its capacity. In other words, the probability of putting a request on disk i is

$$p_i = \frac{c_i}{\sum_{1 \le j \le n} c_j}$$

where c_i is the capacity of disk i and n the total number of disks.

It is worthwhile to note that **round robin** strategy is equivalent to traditional striping. **Proportional to throughput** and **proportional to capacity** are the two most common allocation strategies used in distributed storage. The AdaptRaid5 [19, 20] uses data layout equivalent to proportional to capacity. The disk merging technique described in [71, 72] is in fact a proportional to throughput algorithm.

5.3.2 Effectiveness

Since the problem is NP-complete [25], heuristic algorithms are more practical. Instead of embarking on a lengthy worst case analysis, it is of more interest to investigate its performance under typical workload. Therefore, a program is developed to investigate the effectiveness of the algorithm shown in Figure 5.2 with synthetic workload.

In the synthetic workload, the arrival process is Poisson. The requested allocation size follows exponential distribution with a cutoff threshold. Requests whose size greater than a threshold are ignored since such request are already divided into smaller pieces by splitter. The estimated access rate is also exponentially distributed with a cutoff threshold. Extremely hot data are excluded from disk access since they usually reside in the cache.

Table 5.2 gives the results of the experiment with accurate estimates of access rates. The parameters for the workload are: request size is exponentially distributed with mean 8KB, minimum 1KB and the cut off threshold is 128KB; access rate is exponentially distributed with mean 0.000128 accesses per second and the cut off threshold is 10.

The allocation process stops when any of the three disks is 99% utilized in either capacity or bandwidth. $U_x(i)$ and $U_c(i)$ are the utilizations of throughput and capacity respectively for disk i when the program stops. The *# of reqs* is the number of requests held by the disks at the end of the run. The three disks used in the experiments are listed in Table 5.1. The σ_x and σ_c are the standard deviation of the utilizations for the throughput and capacity over the three disks, respectively. The objective functions F_1 and F_2 are described in Section 5.1 on page 102.

Table 5.2 Experiment Result with Synthetic Workload and Accurate Estimation of Access Rate

Placement strategy	# of reqs	$U_x(1)$	$U_c(1)$	$U_x(2)$	$U_c(2)$	$U_x(3)$	$U_c(3)$	σ_x	σ_c
Best fit with objective F_1	1539041	0.79	0.75	0.79	0.99	0.79	0.99	0.00	0.14
Best fit with objective F_2	1793705	0.92	0.99	0.92	0.99	0.92	0.99	0.00	0.00
round-robin	734350	0.36	0.23	0.43	0.99	0.35	0.47	0.04	0.39
random	736583	0.36	0.24	0.43	0.99	0.35	0.47	0.04	0.39
proportional to max xput	840454	0.43	0.28	0.43	0.99	0.43	0.59	0.00	0.36
proportional to capacity	1157358	0.99	0.64	0.28	0.64	0.47	0.64	0.37	0.00

It can be observed that the best fit heuristic can hold more allocation requests and is therefore more cost-effective than other strategies. Round-robin and random strategies perform similarly. Both of them stop when the second disk is full since it has only 2.0GB, so that it becomes the bottleneck while the first disk drive is filled less than a quarter. Proportional to throughput and proportional to capacity perform slightly better, since they take into consideration one aspect of the disk features. Proportional to capacity performs better than throughput only because the aggregate throughput-capacity ratio (=16.94 per GB per second) is greater than the mean throughput-capacity requirement of the workload (=16.7 per GB per second). In other words, the capacity is the more limiting resource in this experiment.

The best fit heuristic with objective function F_2 (i.e. Equation 5.2) can allocate more than the double number of requests that can be allocated by the random or round-robin strategies. It can be observed that when the program stops, all three disks are filled 99% and the throughput utilization are also same. This means that all available capacity and bandwidth is fully utilized and resources are not wasted. The objective function F_1 (Equation 5.1) is inferior since it emphasizes the balancing of the utilization of disk bandwidth and pays less attention to the variance in capacity utilization.

In a word, it is shown by experiment that the best fit heuristic outperforms all other strategies by a large factor and can utilize all available resource, when the estimation of the access rate is accurate.

5.3.3 Robustness

In reality, the estimations of access rates and read-write ratios are usually inaccurate. Therefore, it is necessary to investigate whether the heuristic still works well with inaccurate estimates.

The inaccuracy appears in two ways: Firstly there can be a big *variance*, which means the actual mean access rate for a certain block is around the estimated mean with some fluctuation, while the overall actual access rate is same as estimated. Secondly the estimation can be *biased*, which means the estimated access rate always tends to be greater (or smaller) than the actual one, and this results in the overall over- (or under-) estimation of the access rate.

The program is modified to add both variance and bias to the estimated access rate. The extent of variance and bias is controlled by two variables: V and B, respectively. The inaccurate estimated access rate is generated by:

$$X_v = \text{random}() * 2V - V$$

$$\texttt{error\_percentage} = \begin{cases} X_v + B & X_v + B > -1 \\ -1 & X_v + B < -1 \end{cases}$$

$$\texttt{estimated\_access\_rate} = \texttt{actual\_access\_rate} \times (1 + \texttt{error\_percentage})$$

where random() returns a uniformly distributed random number in $[0, 1)$. X_v is a random variable which is uniformly distributed over $[-V, V)$. Consequently, the `error_percentage` is uniformly distributed over $[-V + B, V + B)$.

The result with $V = 1$ and $B = -0.5$ is shown in Table 5.3. The parameters of the workload are same as for Table 5.2 except that the estimated access rate are not accurate.

By comparing the results in Table 5.2 and Table 5.3, it is shown that the best fit heuristic performs almost equally well with inaccurate estimation as with accurate estimation. This is partially because each request takes only a very small portion of the resources of a disk. After a request is processed, its actual access rate is discovered and disk utilizations are updated. Therefore, the following requests will get the correct knowledge of previous requests and are routed to a proper disk.

Table 5.3 Experiment Result with Synthetic Workload and Inaccurate Estimation of Access Rate

Placement strategy	# of reqs	$U_x(1)$	$U_c(1)$	$U_x(2)$	$U_c(2)$	$U_x(3)$	$U_c(3)$	σ_x	σ_c
Best fit with objective F_1	1539043	0.79	0.75	0.79	0.99	0.79	0.99	0.00	0.14
Best fit with objective F_2	1793705	0.92	0.99	0.92	0.99	0.92	0.99	0.00	0.00
round robin	734350	0.36	0.23	0.43	0.99	0.35	0.47	0.04	0.39
random	734772	0.36	0.23	0.43	0.99	0.35	0.47	0.04	0.39
proportional to max xput	839267	0.43	0.28	0.43	0.99	0.43	0.58	0.00	0.36
proportional to capacity	1156598	0.99	0.64	0.28	0.64	0.47	0.64	0.37	0.00

5.3.4 Summary

In this section, the verification for the effectiveness and robustness of the best-fit heuristic is presented. It is shown that the best-fit heuristic work well when each allocation request takes very small portion of the system resource, which is generally the case in real system. Is is also shown that the second objective function $F_2 = \mathrm{Var}(U_i^x) + \alpha\,\mathrm{Var}(U_i^c)$ which checks the variance of the utilizations works better than the other objective function $F_1 = \max\limits_{1 \leq i \leq n} \{U_i^x, \alpha U_i^c\}$. Therefore, in the rest of the dissertation, only F_2 is considered.

5.4 Constraints on Allocation

The greedy algorithm discussed in the previous section can give satisfactory result with a synthetic workload (see Section5.3). However, there are several constrains that need to be considered while making the allocation. Some constrains are *soft constrains*, which means they are desirable but not a must. The others are *hard* *constrains*, which means that they must be complied to in order to get a correct allocation. The constrains are described as follows:

1. The data and corresponding parity stripe units in a RAIDx parity group must not be placed on the same device. This is a must-be-met constraint that ensures the data can be recovered when there is a device failure and is essential to the correctness of the scheme.

2. The blocks that are from the same allocation request should be placed on a single device up to the size of a stripe unit. This is a soft constraint. It encourages the space to be allocated in large continuous chunks and therefore reduces the number of fragments for an allocation. Since sequential read is much more efficient than random small reads, continuous data chunk provides higher transfer rate for large request sizes.

3. Preferably, the stripe units of a parity group should be placed on a set of devices of the same type. This is also a soft constraint. The purpose for this constraint is to reduce the service time and response time variance in a virtual array. Since the mean service time on different disk types for a given request may vary, a virtual array consisting of VDs from heterogeneous disk drives shows high variance in response time. Although this is not a big problem for applications that do not impose a response time limit, in some situations this high variance in response time may impair the performance. For example, at the time of one disk failure, a request to reconstruct a block is processed as a fork-join of a set of read requests on surviving disks. The high variance of response time on individual disks increases the overall response time of this fork-join request.

5.5 The Allocation Algorithms Used in Simulation

In this section, the procedures for free space management and allocation in HDA are described. These procedures are implemented by extending the DASim simulation toolkit [63] and the simulation results are given in Chapter 6.

A simplified configuration is considered, in which there is no deallocation of data blocks. In other words, the data are written on the drive and never erased. The absence of data deallocation greatly simplifies free space management, since no garbage collection is required.

HDA is a hierarchical organization, e.g. VA consists of VDs, each VD has multiple RBs, and an RB may hold multiple data blocks. Therefore, the free space management has multiple levels as well, i.e. management of free VDs and management of free space within VAs and VDs.

To start with, the conceptual flowchart to handle an allocation request is shown in Figure 5.3. Here, only RAID1 and RAID5 are shown, but other RAID levels can be incorporated similarly.

For each RAID level, there is an *active VA* associated with it. Allocation requests come tagged with a certain RAID level, then the corresponding *active VA* is checked. Space is allocated from the *active VA* for the allocation requests until space is exhausted. When this happens, a new VA is created from the free VDs on one or more devices (based on the RAID level), and the new VA become the active VA for that RAID level.

In Figure 5.3, there are three operations that need further explanation: (1) checking free space in VA (block a), (2) allocating from a VA (block b) and (3) creating new VA (block c). These operations are described below.

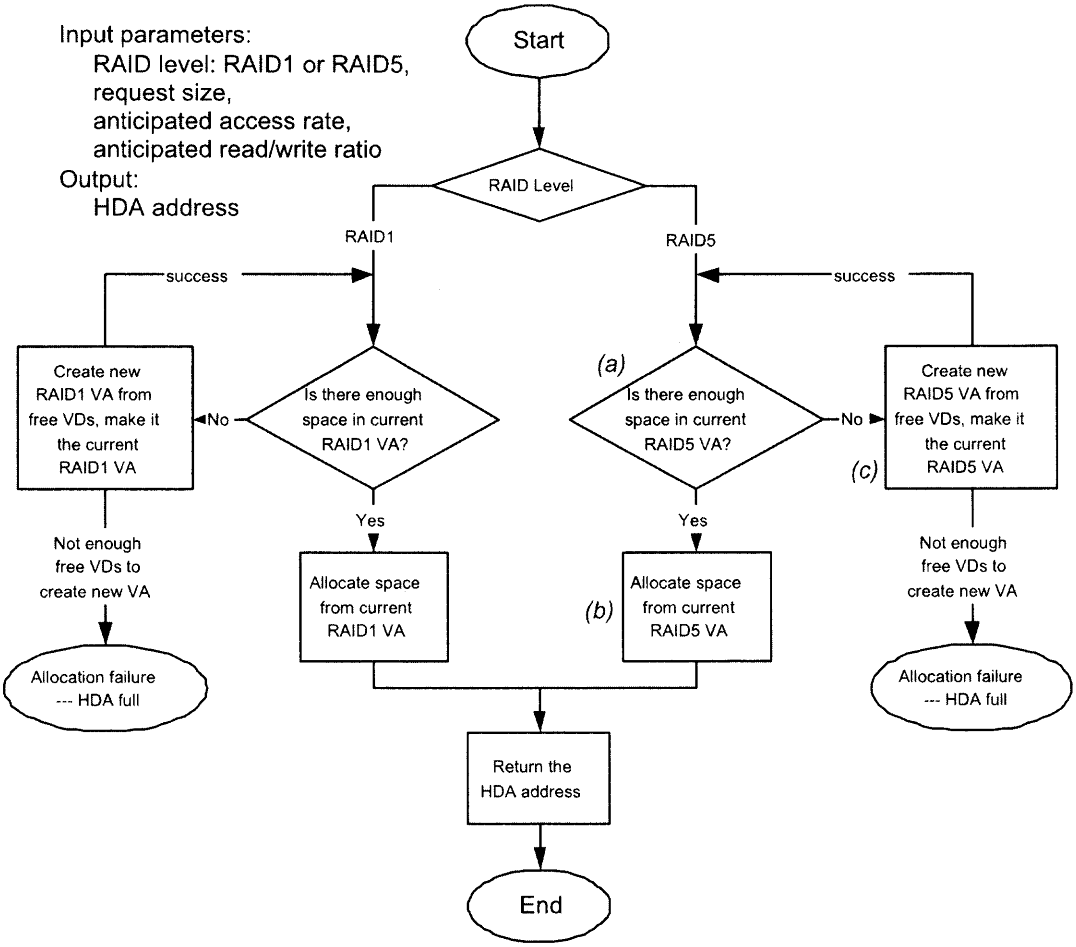

Input parameters:
RAID level: RAID1 or RAID5,
request size,
anticipated access rate,
anticipated read/write ratio
Output:
HDA address
Start
RAID Level
RAID1
RAID5
success
success
(a)
Create new RAID1 VA from free VDs, make it the current RAID1 VA
No
Is there enough space in current RAID1 VA?
Is there enough space in current RAID5 VA?
No
Create new RAID5 VA from free VDs, make it the current RAID5 VA
(c)
Yes
Yes
Not enough free VDs to create new VA
Allocate space from current RAID1 VA
(b)
Allocate space from current RAID5 VA
Not enough free VDs to create new VA
Allocation failure --- HDA full
Allocation failure --- HDA full
Return the HDA address
End

Figure 5.3 Flowchart to handle allocation requests for RAID1 and RAID5 virtual arrays.

5.5.1 Checking Free Space

Since the Allocator follows the Splitter in the HDA architecture (see Figure 4.1), no further split will occur. In other words, all requests should be put into a single RB. Consequently, the free space check should ensure that there is at least one data VD in the active VA that has enough free space, rather than the sum of free space on all data VDs is large enough.

5.5.2 Allocating from a VA

After checking that there is enough space for the allocation request, the allocation request is handled by the active VA of the corresponding RAID level (block *b* in Figure 5.3). The task is to decide to which VD this allocation request should be sent. The algorithm used is the greedy heuristic shown in Figure 5.2. The difference is that the capacity and bandwidth are for virtual disks rather than real ones. The capacity of each VD is just its size, while the bandwidth is calculated when the VD is incorporated into a new VA. The bandwidth of a VD is not a limit that physically exists, which means a VD can have bandwidth utilization greater than the bandwidth assigned to it, in which case the VD is actually using the bandwidth that is assigned to other VDs. In fact, the bandwidth for a VD is a goal rather than a limit. Therefore, we call this virtual bandwidth the *target bandwidth* of a VD. In general, the target bandwidth of a VD is a fraction of the remaining bandwidth of the disk. For example, if there are 10 free VDs on a disk, then each VD takes 1/10 of the unutilized disk bandwidth. However, this value is adjusted to speed up the balancing process. The details of how to compute the *target bandwidth* for a VD is shown in Section 5.5.3.

The procedure for allocating from a VA is shown in Algorithm 1. In the algorithm, the functions listed in Table 5.4 are used.

5.5.3 Creating new VA

For a certain RAID level, when the *active VA* run out of space, a new VA is created from free VDs. Each VD comes from different physical devices. Since the width of a VA (i.e., the number of VDs in a VA) is usually less than the total number of devices, a selection has to be made. In the current implementation, the decision is made based on the capacity usage. In other words, we choose VDs from the devices that have a higher percentage of free space.

Table 5.4 Functions Used in Allocation Algorithms

Function	Param	Return Value
Dev(v)	VD# v	corresponding device number for v
Ux(d)	Device# d	current utilization of bandwidth for device d
Uc(d)	Device# d	current utilization of capacity for device d
NumFreeVD(d)	Device# d	the number of free VDs on device d
GetFirstFreeVD(d)	Device# d	the VD# of the first free VD on device d
TotalBW(d)	Device# d	the total bandwidth for device d
TargetBandWidth(v)	VD# v	the target bandwidth for VD# v
GetVDs(va)	VA# va	an array of VDs in the VA
GetDataVDs(va)	VA# va	an array of data VDs in the VA
GetCheckVDs(va)	VA# va	an array of check VDs in the VA
Ux_VD(v)	VD# v	current utilization of bandwidth for VD# v
Uc_VD(v)	VD# v	current utilization of capacity for VD# v
ComputeVar(a)	Array a	compute the variance over the array

For RAID levels that have redundancy, another decision to be made is on which VD(s) should be used to store the parity blocks. In the current implementation, the parity VD for a RAID5 VA is chosen randomly. An alternative way of choosing the parity VD is to select the VD from the device that has the lowest bandwidth utilization. The argument for this approach is that the parity VD may have a higher arrival rate for requests than data VDs. However, this is not always true, since it depends on the mean read/write ratio of disk accesses, as well as the width of the VA.

For each VD in the new VA, its *target bandwidth* is computed. The target bandwidth is used in the best-fit heuristic to guide the allocations requests into a proper VD, as shown in Algorithm 1. The calculation of target bandwidth takes into consideration the current bandwidth utilization of the device. Therefore, it carries the statistical information of actual data access rate, which is observed by the system monitor. In general, the target bandwidth of a VD is the remaining bandwidth divided by the number of free VDs on the device. Then this target bandwidth is adjusted with a factor to speedup the balancing process. When the device is busier

Algorithm 1 Allocating from active VA

```
procedure AllocFromVA(VA# va, Request req)                          ▷ va is the active VA#
    Array dataVDs = GetDataVDs(va)                                   ▷ Get the array of data VDs
    n = dataVDs.length                                               ▷ The number of data VDs
    Array BWArray[n]                                                 ▷ Bandwidth utilization for dataVDs
    Array CapArray[n]                                                ▷ Capacity utilization for dataVDs
    for i=1 to n do                                                  ▷ Fill the two arrays
        BWArray[i] = Ux_VD(dataVDs[i])
        CapArray[i] = Uc_VD(dataVDs[i])
    end for
    CurrentMin = $\infty$                                            ▷ Variable to record the current min value
    CurrentVDi = -1                                                  ▷ Subscript in dataVDs of the current choice
    for i=1 to n do                                                  ▷ For each data VD
        SaveBW = BWArray[i]                                          ▷ Save old value
        SaveCap = CapArray[i]
        CapArray[i] = CapArray[i] + req.size / SIZE_OF_VD            ▷ Update utils
        BWArray[i] = BWArray[i] + req.bw / TargetBandWidth(dataVDs[i])
        value = ComputeVar(BWArray) + $\alpha \times$ ComputeVar(CapArray)
        if value < CurrentMin then                                   ▷ Find the minimum
            CurrentVDi = i
            CurrentMin = value
        end if
        BWArray[i] = SaveBW                                          ▷ Restore old value
        CapArray[i] = SaveCap
    end for
    ResultVD = dataVDs[CurrentVDi]                                   ▷ The result VD
    ······                                                           ▷ Bookkeeping – update the meta info
end procedure
```

than the average, the factor is smaller than one. When the device is less busy, the factor is greater than one.

The procedure to create a new VA is shown in Algorithm 2 and the procedure to calculate the target bandwidth for VDs is shown in Algorithm 3. The meaning of the functions used in the algorithms are listed in Table 5.4.

Algorithm 2 Compose a new VA from free VDs.

```
procedure ComposeVA(RAIDLevel R)                    ▷ VDS is the set of all VDs in VA
    MinHeap S = ∅                                   ▷ S is initialized as an empty heap
    for d=0 to NumofDevices-1 do                    ▷ For all devices
        if NumFreeVD(d) > 0 then                    ▷ If the device has free VD
            Pair(Uc(d), d) → S                      ▷ Add the pair into MinHeap S
                                  ▷ The heap is sorted on the first element of the pair
        end if
    end for
    Set VDS = ∅                                     ▷ The set to keep the VDs used in the new VA
    n = VA width for RAIDLevel R                    ▷ The width of virtual array
    if  n < |S| then                                ▷ Not enough free VDs
        return failure                              ▷ Fail to create a new VA
    end if
                       ▷ Select the first n device with lowest utilization of capacity,
                       ▷ and add the first free VD from the device to the result set.
    for  i=0 to n  do
        GetFirstFreeVD(S.popfirst()) → VDS
    end for
    computeTargetBW(VDS)        ▷ Compute the target bandwidth, see Algorithm 3
    choose a VD randomly from VDS to be the parity VD
    ......                                          ▷ Bookkeeping – update the meta info
end procedure
```

Algorithm 3 Compute the target bandwidth for VDs in a new VA

```
procedure computeTargetBW( set⟨VD#⟩ VDS) ▷ VDS is the set of all VD in VA
    avgUtil = $\sum_{i=1}^{n} Ux(Dev(VD_i))/n$      ▷ Compute mean utilization
    for i=1 to k do                                 ▷ For each VD in VDS
        dev = Dev(VD_i)                             ▷ Get its device number
        diff = Ux(dev) − avgUtil
        p = diff × 200
        if (diff > 0) then                          ▷ If the VD is on a busy disk
            factor = $0.9^p$
        else
            factor = $1.1^p$
        end if
        TargetBandWidth(VD_i) = $\frac{(1-Ux(dev))\times TotalBW(dev)}{NumFreeVD(dev)}$ × factor
    end for
end procedure
```

CHAPTER 6

PERFORMANCE OF HETEROGENEOUS DISK ARRAY

The Heterogeneous Disk Array architecture described in Chapter 4 is prototyped based on the DASim simulator [63]. This chapter reports the simulation assumptions and results.

The chapter is organized as follows: Firstly the configurations of the simulation is described. Then the results based on accurate estimates of access rates is presented. This is followed by the simulation results based on inaccurate estimates of access rates.

6.1 Configurations

A heterogeneous disk array with six disks is considered. The six disks have four models, with capacity ranging from 2 GB to 9 GB. The disk models and their specifications are given in Table 6.1.

Table 6.1 Specifications of Disks Used in HDA Simulation

Disk#	Model	Capacity	Bandwidth (access/second)	Bandwidth-Capacity Ratio	RPM	Sectors Per Track
0	IBM18ES	8.6G	88.01	9.779	7200	247-390
1	IBM18ES	8.6G	88.01	9.779	7200	247-390
2	Atlas10k	8.5G	116.89	13.13	10000	229-334
3	Barracuda	2.0G	74.53	35.49	7200	119-186
4	Barracuda	2.0G	74.53	35.49	7200	119-186
5	Cheetah4LP	4.2G	91.63	20.36	10000	131-195

The arrival process is Poisson for both allocation and access (i.e. read/update) requests. The request size follows exponential distribution. The mean request size is 100 sectors or 50KB, with a cutoff threshold at 4096 sectors and a minimum at one sector. The access rate is also exponentially distributed, with mean access rate 8×10^{-7} accesses per second. The minimum access rate is 0, the maximum is 10 accesses per second. Two RAID levels (RAID1 and RAID5) coexist in the HDA. Each allocation request is tagged as either RAID1 or RAID5, with 30% of them tagged as RAID1. Multiples runs of simulations is executed with various read/write ratio for data blocks, but only read:write = 3:1 is reported here.

The arrival rate for the allocation requests remains constant throughout the simulation. In other words, a constant flow of allocation requests is assumed. The arrival rate for accessing the data blocks depends on how many data objects have been allocated on the disk. The simulator keeps track of all the data objects that has been allocated and generates read/write requests according to the actual access rate of each data object after it is allocated. As time elapses, more allocation requests are processed and more space is allocated. Therefore, the arrival rate for data objects increases with time, so do the utilizations and response times for disk accesses. The simulation stops when either bandwidth or capacity utilization of any disk exceeds 95%.

The size of a relocation block is 10 megabytes, each virtual disk has 2 RBs. A RAID1 virtual array consists of two virtual disks. A RAID5 virtual array consists of five virtual disks, one of which stores the parity.

6.2 Simulation Results with Accurate Estimation of Access Rates

With accurate estimation of access rate and read write ratio, the simulation results are shown in Figure 6.1, 6.2, 6.3 and 6.4. Since the target for the HDA is to balance the utilizations on both capacity and bandwidth for all disks, we plot the two utilizations of each disk over the entire simulation time frame to show whether they are close to each other. The read response time and arrival rate on individual disks are shown as well.

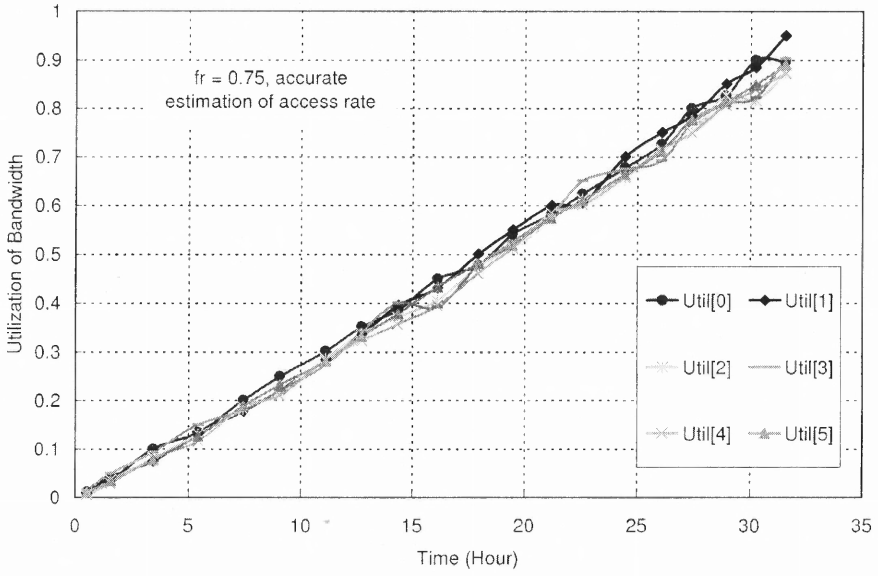


Figure 6.1 Utilization of bandwidth, with accurate estimations of access rate and read write ratio.

It can be observed from Figure 6.1 that the bandwidth utilization on all disks are very close to each other during the entire allocation process. There are some fluctuations, but the difference of the utilization between any two disks are less than 5%. In Figure 6.2, it is shown that the utilization of capacity on individual disks are closely matched at all time too. Therefore, both capacity and bandwidth are utilized in a balanced way and the system resources are fully exploited.

In Figure 6.3, the read response time for individual disk is shown. The response time for the 4th and 5th disks (index number 3rd and 4th) are higher than the rest

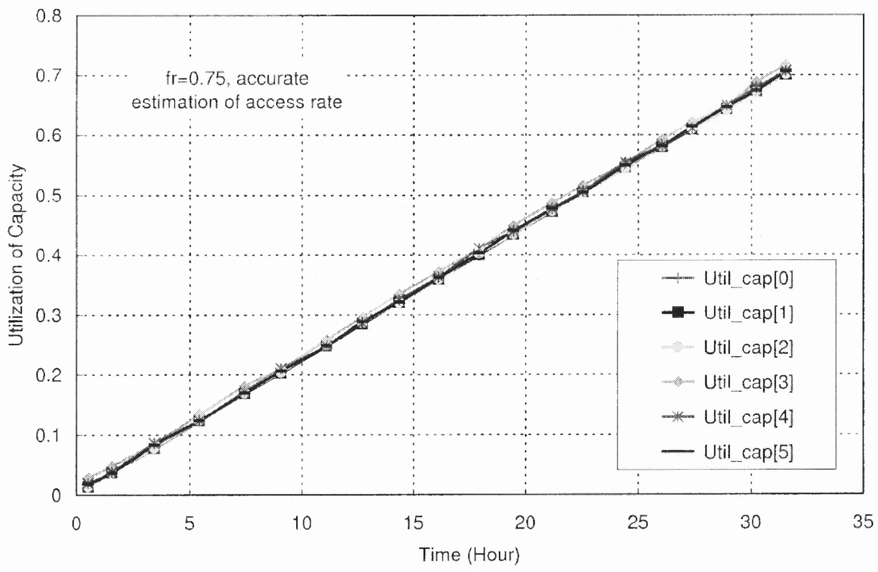


Figure 6.2 Utilization of capacity, with accurate estimations of access rate and read write ratio.

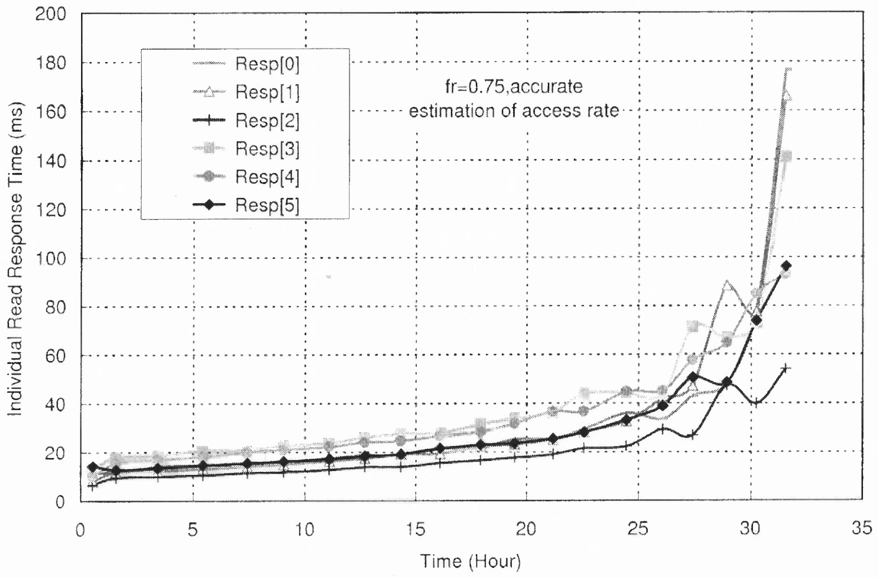


Figure 6.3 Read response time on each disk, with accurate estimations of access rate and read write ratio.

of disks because they are slower and have a higher mean service time. The curves follow the same trend and response time remains steady in a wide range of time. This means the system works in a stable manner.

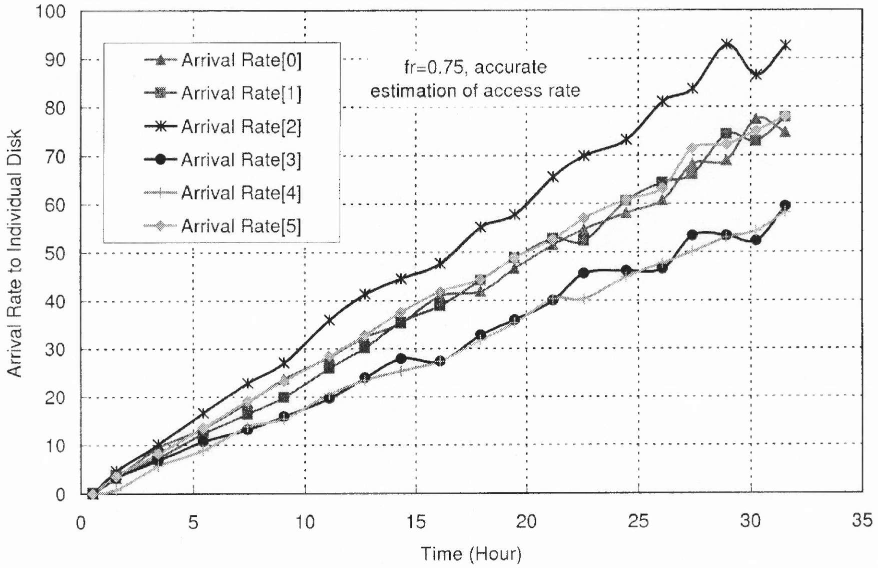


Figure 6.4 Arrival rate on each disk, with accurate estimations of access rate and read write ratio.

In Figure 6.4, the mean arrival rates for data access requests (i.e. read and update requests) are shown for each individual disk. All disks gains a linearly increasing arrival rate over the time. Among them, the 3rd disk (index number 2nd) has the highest slope, which means it takes a larger portion of all arriving requests, while the 4th and 5th (index number 3rd and 4th) disks take the lowest arrival rates. This is because the 3rd disk has the highest bandwidth and the 4th and 5th disks have the lowest bandwidths. The arrival rate grows almost linearly, and the ratio of arrival rate between any two disks remains the same, which equals the ratio of their bandwidths.

In short, the simulation shows that the HDA architecture can balance the utilization of both bandwidth and capacity over all disks, and therefore exploits the system resources to the greatest extent possible. The read response time on each disk may differ, since the disks have different access times but close utilizations. However, the difference is rather small. The arrival rate of read and update requests on each disk is proportional to the bandwidth of the disk.

6.3 Simulation Results with Inaccurate Estimation of Access Rates

In a real system, accurate estimation of access rates and read write ratios is an impossible task. Therefore, whether HDA can perform satisfactorily needs to be investigated. In this section, the simulation result of HDA architecture with inaccurate estimation of access rate is reported. The configurations remain the same as in Section 6.2, except that the estimated access rates and read write ratio vary from the actual values.

Similar to Section 5.3.3, the estimated access rate for an allocation request is a calculated by multiplying an error percentage on the actual access rate (see page 110). For the estimated read write ratio, a similar approach is applied. The value of estimated read write ratio is checked to make sure it is between zero and one.

In the simulation, a variance of 30% and bias of 10% (i.e., $V = 0.3$ and $B = 0.1$) is added to the estimated access rate. A variance of 30% and bias of 10% ($V = 0.3$ and $B = -0.1$) is added to the estimated read write ratio. The simulation result with inaccurate estimations are show in Figure 6.5, 6.6, 6.7 and 6.8.

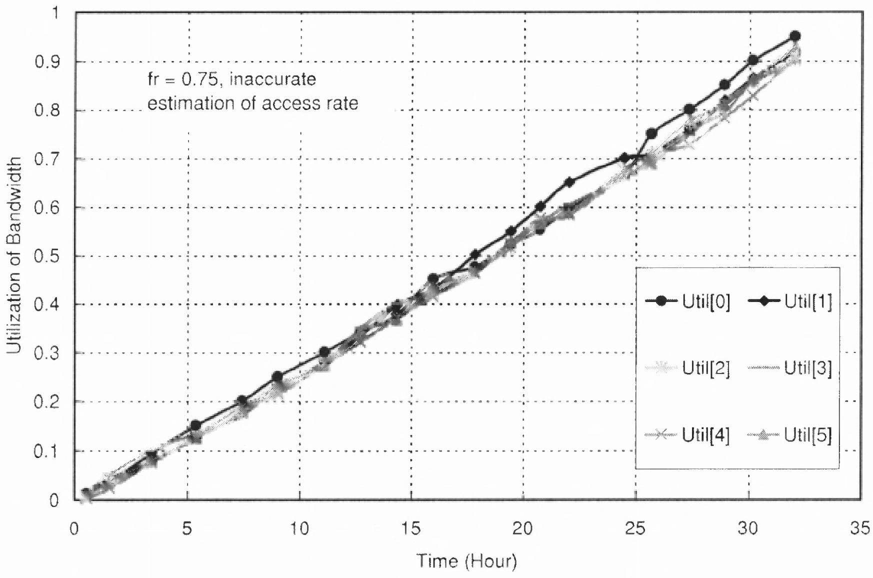


Figure 6.5 Utilization of bandwidth, with inaccurate estimations on access rate and read write ratio.

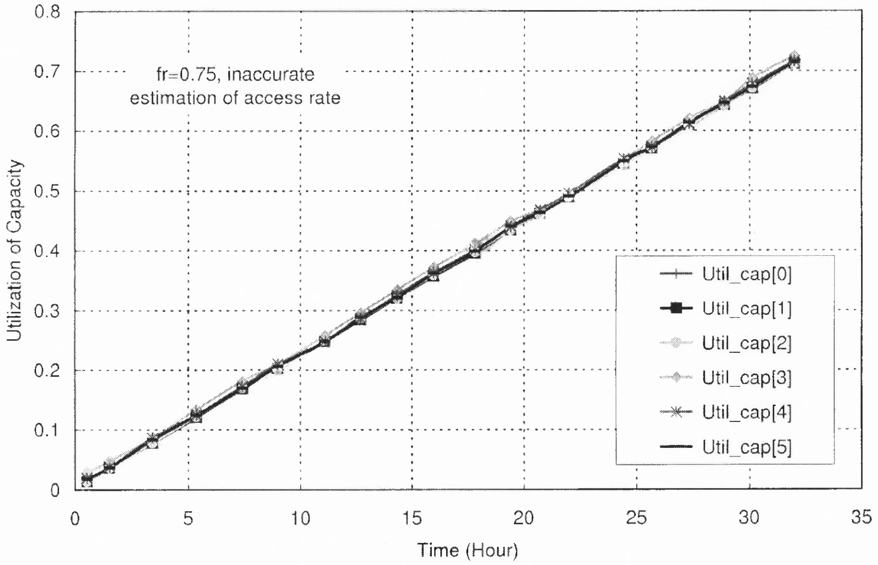


Figure 6.6 Utilization of capacity, with inaccurate estimations on access rate and read write ratio.

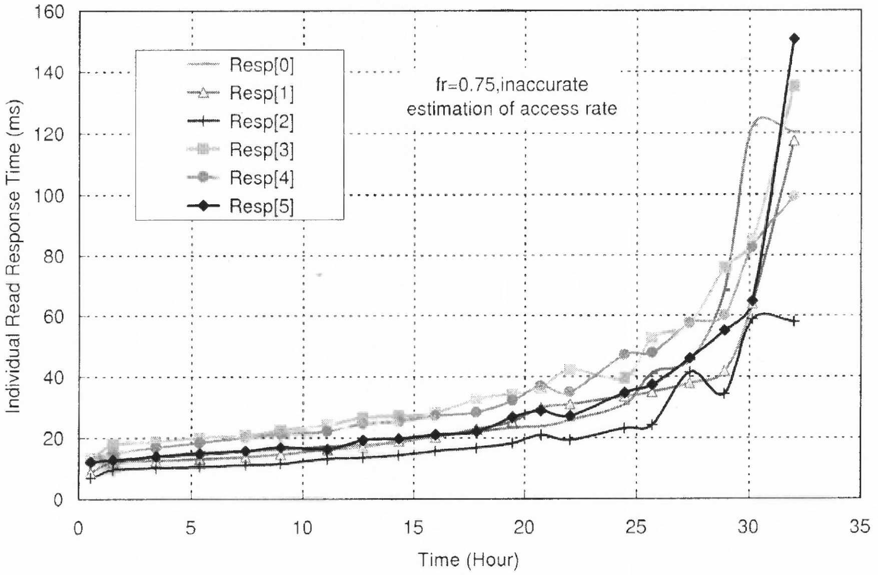


Figure 6.7 Read response time on each disk, with inaccurate estimations on access rate and read write ratio.

It is shown in Figure 6.5 and 6.6 that both capacity and bandwidth utilizations at all disks are close to each other. By comparing Figure 6.5 with Figure 6.1, it can be observed that the two figures are very similar. With inaccurate estimations, there is a little more fluctuations from the average, which is a result of the inaccuracy.

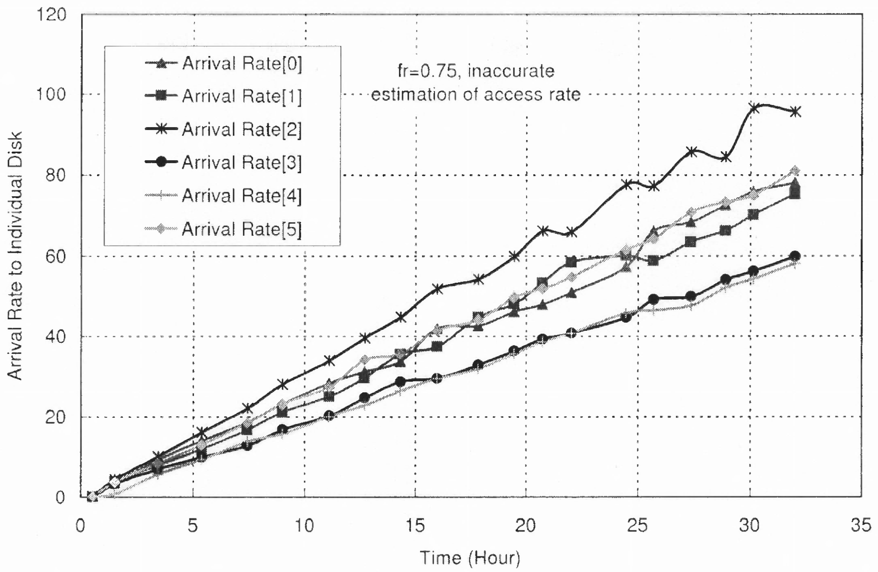


Figure 6.8 Arrival rate on each disk, with inaccurate estimations on access rate and read write ratio.

The response time and arrival rates for individual disks are shown in Figure 6.7 and Figure 6.8, respectively. Both of them are very close to the corresponding figure with accurate estimations in Section 6.2.

In a word, the simulation shows that the HDA system can tolerate inaccuracy in the estimation of access rate and read write ratio come with the allocation requests. Therefore, the HDA architecture works satisfactorily with inaccurate estimation of access rate and read write ratio.

CHAPTER 7

CONCLUSIONS

In this dissertation, device independent cost models for various RAID levels are defined. Based on these cost models and given detailed specification of disk characteristics, the maximum throughput achievable for various RAID levels can be obtained. A queuing analysis based on M/G/1 queuing model is also described. The response time are obtained for RAID0, RAID5, RAID6, EVENODD, and RM2 using this queuing analysis as well as by simulation. The simulation results serve as validation of the queuing analysis, and also to evaluate the performance of SATF scheduling policy.

In the second half of the dissertation, the Heterogeneous Disk Array (HDA) architecture is described. The motivations for this new architecture is based on several trends in storage technology. This heterogeneous disk array features: (i) Allowing different disk models and making efficient usage of the extra capacity and bandwidth made available by new disk drives, hence heterogeneous devices. (ii) Allowing multiple RAID schemes to coexist on a single physical device, hence heterogeneous configurations. (iii) The disk loads are roughly balanced in terms of both bandwidth and capacity utilizations.

The data structures for the HDA are defined and the algorithm for the allocation is described. A simulation based on these data structures and allocation algorithm is made to investigate its performance. Simulation results shows that it is possible to balance the utilization of bandwidth and capacity at the same time and therefore provides efficient usage of the available resources in a heterogeneous disk environment. The reliability requirements are also met by using a proper RAID scheme. The response time on each disk are stable and acceptable for a wide range of utilizations.

APPENDIX A

QUEUING FORMULAS

A.1 M/G/1 Queuing Formulas

Most of the equations in this section can be found in [61]. Let b_i be the ith moment of service time, λ be the arrival rate, $\rho = \lambda b_1$ be the utilization factor, the mean waiting time

$$\overline{W} = \frac{\lambda b_2}{2(1-\rho)} = \frac{\lambda(b_1^2 + \sigma_B^2)}{2(1-\rho)} = \frac{\lambda b_1^2(1 + c_B^2)}{2(1-\rho)}$$

where $c_B^2 = var[B]/b_1^2 = (b_2 - b_1^2)/b_1^2 = b_2/b_1^2 - 1$, is the coefficient of variation squared.

The second moment of waiting time is:

$$\overline{W^2} = \frac{\lambda b_3}{3(1-\rho)} + \frac{(\lambda b_2)^2}{2(1-\rho)^2}$$

The moments of response time

$$\overline{R^i} = E[(W+T)^i]$$

Given the waiting time and service time are independent, the first two moments are as follows:

$$\overline{R} = b_1 + \overline{W}$$

$$\overline{R^2} = b_2 + \overline{W^2} + 2\overline{W}b_1$$

A.2 Non-Preemptive Priority Queuing

The Head-of-the-line (HOL) priority queueing system can be described with the following parameters:

Jobs has P priority levels (with Phighest, 1 lowest),

P queues for each priority level,

Jobs at each level are served in FCFS order,

Jobs in class p have an arrival rate λ_p,

mean and second moment of service time is b_p and $b_p^{(2)}$,

The total arrival rate is $\Lambda = \sum_{p=1}^{P} \lambda_p$,

Fraction of arrivals in class p is $f_p = \lambda_p / \Lambda$, $1 \leq p \leq P$,

The mean overall service time is $b = \sum_{p=1}^{P} f_p b_p$,

The utilization factor for class p is $\rho_p = \lambda_p b_p$,

The overall utilization factor is $\rho = \sum_{p=1}^{P} \rho_p = \lambda \overline{x}$,

The mean waiting time and response time for class p requests is denoted by W_p and R_p, respectively. $R_p = b_p + W_p$.

The overall waiting time and response time are

$$W = \sum_{p=1}^{P} f_p W_p$$

$$R = \sum_{p=1}^{P} f_p R_p = b + W$$

let

$$\sigma_p = \sum_{i=p}^{P} \rho_i, \qquad W_0 = \frac{1}{2} \sum_{p=1}^{P} \lambda_p b_p^{(2)}$$

Then the waiting time for a requests with priority p is:

$$W_p = \frac{W_0}{(1-\sigma_p)(1-\sigma_{p+1})}, \quad 1 \le p \le P.$$

second moment of waiting time for priority class p is:

Let

$$W_0^{(2)} = \frac{1}{3}\sum_{k=1}^{P} \lambda_k b_k^{(3)}$$

$$\begin{aligned} W_p^{(2)} &= \frac{\sum_{k=1}^{P} \lambda_k b_k^{(3)}}{3(1-\sigma_p)(1-\sigma_{p+1})^2} + \frac{\left(\sum_{k=p}^{P} \lambda_k b_k^{(2)}\right)\left(\sum_{k=1}^{P} \lambda_k b_k^{(2)}\right)}{2(1-\sigma_p)^2(1-\sigma_{p+1})^2} + \\ &\quad \frac{\left(\sum_{k=p+1}^{P} \lambda_k b_k^{(2)}\right)\left(\sum_{k=1}^{P} \lambda_k b_k^{(2)}\right)}{2(1-\sigma_p)(1-\sigma_{p+1})^3} \\ &= \frac{W_0^{(2)}}{(1-\sigma_p)(1-\sigma_{p+1})^2} + \frac{\left(\sum_{k=p}^{P} \lambda_k b_k^{(2)}\right) W_0}{(1-\sigma_p)^2(1-\sigma_{p+1})^2} + \frac{\left(\sum_{k=p+1}^{P} \lambda_k b_k^{(2)}\right) W_0}{(1-\sigma_p)(1-\sigma_{p+1})^3} \end{aligned}$$

A.3 Fork Join Approximation

A.3.1 Two-way Fork-Join Approximation

In a K way fork-join queuing system, the expected value of the maximum of K response times (R_K^{max}) is an upper bound for $R_K^{F/J}$. When regular requests contribute heavily to the utilization of individual disks, the components of fork-join response time are almost independent. In this case, R_K^{max} in addition to begin an upper bound to $R_K^{F/J}$, is also a good approximation to it.

$$R_K^{max} = \int_0^\infty \left[1 - \prod_{i=1}^{K} R_i(t)\right] dt = \int_0^\infty \left[1 - R^K(t)\right] dt$$

One method to simplify the computation of the mean response time for fork-join requests is to match the first two moments of the response time distribution to an Erlang or hyperexponential distribution, depending on whether $c_R < 1$ or $c_R > 1$.

The coefficient of variation of response time can be obtained using the expressions for the moments of response time. If we concerned with SRW requests, which constitute a fork-join request,

$$c_R^2 = \frac{Var[R_{SRW}]}{R_{SRW}^2} = \frac{c_x^2 + \frac{\rho s_x}{3(1-\rho)} + \frac{\rho(1+c_x^2)}{4(1-\rho)^2}}{1 + \frac{\rho(1+c_x^2)}{1-\rho} + \frac{\rho(1+c_x^2)}{4(1-\rho)^2}}$$

where $c_x^2 = \overline{X_{SRW}^2}/X_{SRW}^2 - 1$ and $s_x = \overline{X_{SRW}^3}/X_{SRW}^3$. For $\rho = 0$(resp. $\rho \to 1$) $c_R = c_X$ (resp. $c_R \to 1$). For SRW requests, c_r tends to be smaller than one for $0 \leq f_r \leq 1$. Therefore, an Erlang distribution with a mean equal to the original distribution and $k = 1/c_R^2$ stages can be used. When k is not an integer, we can use linear interpolation: $R_k^{max} = (H-k)R_H^{max} + (k-J)R_J^{max}$, where $H = ceil(k)$ and $J = floor(k)$.

The Erlang distribution: k exponential stages with parameter μ.

$$f(t) = \frac{\mu(\mu x)^{k-1}}{(k-1)!} e^{-\mu t}, \quad k \geq 1$$

$$F(t) = 1 - e^{-\mu x} \sum_{j=0}^{k-1} \frac{(\mu x)^j}{j!}$$

with the mean $E[t] = \frac{k}{\mu}$, $E[t^2] = \frac{2k}{\mu^2}$, $\sigma_t^2 = var[t] = \frac{k}{\mu^2}$, $c_x^2 = \frac{1}{k} < 1$.

To make the mean equal to original distribution, let $\mu = k/R$.

$$R_K^{max} = \int_0^\infty \left[1 - \prod_{i=1}^{K} \left(1 - e^{-\mu_i t} \sum_{j=0}^{k_i - 1} \frac{(\mu_i t)^j}{j!} \right) \right] dt$$

Two queues with response time R_1 and R_2 and parameters μ_1, μ_2, k_1, k_2 and be approximated as:

$$R_2^{max} = R_1 + R_2 - \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \binom{m+n}{m} \frac{\mu_1^m \mu_2^n}{(\mu_1+\mu_2)^{m+n+1}}$$

A.3.2 Multi-way Fork-Join Approximation

Approximate response time with extreme value distribution (type II, i.e. maximum value), which is defined by:

$$P[X < x] = exp(-e^{-(x-a)/b})$$

with:

$$E[x] = a + \gamma b, \qquad Var[x] = (\pi b)^2/6$$

where $\gamma \approx 0.5772$ is the Euler-Mascheroni constant. Assuming all component operations have same distribution of response time, with first two moments R and $R^{(2)}$, then we can match the variance to the response time variance and get

$$b = \frac{\sqrt{6(R^{(2)} - R^2)}}{\pi}$$

The overall response time for a N way fork/join request is

$$R_N^{F/J} = (a + \gamma b) + b\ln(N) = R + b\ln(N)$$

REFERENCES

[1] N. Allen. Don't waste your storage dollars: what you need to know. Research note COM-13-1217, Gartner Group, March 2001.

[2] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems*, 19(4):483–518, 2001.

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

[4] K. S. Amiri. *Scalable and manageable storage systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, December 2000.

[5] E. Anderson, M. Kallahalla, S. Spence, R. Swaminathan, and Q. Wang. Ergastulum: Quickly finding near-optimal storage system designs. HP Laboratories SSP technical report HPL-SSP-2001-05, June 2002.

[6] E. Anderson, R. Swaminathan, A. Veitch, G. Alvarez, and J. Wilkes. Selecting RAID levels for disk arrays. In *Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, USA, January 2002.

[7] ANSI X3.131-1986, New York NY. *American National Standard for Information System - Small Computer System Interface(SCSI)*, 1986.

[8] M. Blaum. A Course on Error-Correcting Codes. Lecture Notes available from author, 1997.

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

[10] M. Blaum, J. Brady, J. Brunk, J. Menon, and A. Vardy. The EVENODD code and its generalizations. In H. Jin, T. Cortes, and R. Buyya, editors, *High Performance Mass Storage and Parallel I/O*, pages 187–205. John Wiley, 2002.

[11] E. Borowsky, R. Golding, A. Merchant, L. Schrier, E. Shriver, M. Spasojevic, and J. Wilkes. Using attribute-managed storage to achieve qos. In *Proc. of the 5th Int'l Conf. Workshop on Quality of Service*, Columbia University, New York, June 1997.

[12] J. P. Buzen and P. S. Goldberg. Guidelines for the use of infinite source queueing models in the analysis of computer system performance. In *Proceedings of AFIPS 1974 National Computer Conference*, volume 43, pages 371–374, 1974.

[13] J. Chandy and A. Reddy. Failure evaluation of disk array organizations. In *Proceedings of 13th International Conference on Distributed Computing Systems (ICDCS)*, pages 319–326, 1993.

[14] C. Chekuri and S. Khanna. On multi-dimensional packing problems. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, pages 185–194, 1999.

[15] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[16] S. Chen and D. Towsley. The design and evaluation of RAID5 and parity striping disk array architectures. *Journal of Parrallel and Distributed Computing*, 10(1/2):41–57, Feburary 1993.

[17] S. Chen and D. Towsley. A performance evaluation of RAID architectures. *IEEE Transactions on Computers*, 45(10):1116–1130, 1996.

[18] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. In *Proceeding of the Third USENIX Conference on File and Storage Technologies*, San Francisco, CA, March 2004.

[19] T. Cortes and J. Labarta. A case for heterogeneous disk arrays. In *Proceedings of the 1st IEEE International Conference on Cluster Computing (Cluster 2000)*, Saxony, Germany, November 2000.

[20] T. Cortes and J. Laborta. Extending heterogeneity to RAID level 5. In *Proceedings of the 2001 USENIX Annual Technical Conference*, pages 119–132, Boston, MA, June 2001.

[21] A. Dan and D. Sitaram. An online video placement policy based on bandwidth to space ratio (BSR). In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 376–385, San Jose, CA, May 1995. ACM Press.

[22] Validated disk parameters. `http://www.pdl.cmu.edu/DiskSim/diskspecs.html` (Retrieved on April 2004).

[23] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *ACM Computing Surveys (CSUR)*, 14(2):287–313, 1982.

[24] G. Fu and A. Thomasian. Fork join approximation and validation. Technical Report, Integrated System Lab, CS Department, NJIT, 2003.

[25] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA, 1979.

[26] G. A. Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage.* MIT Press, 1992.

[27] R. Golding, E. Shriver, T. Sullivan, and J. Wilkes. Attribute-managed storage. In *Workshop on Modeling and Specification of I/O*, 1995.

[28] E. Grochowski and R. Fontana Jr. Magnetic recording technology – advances through the year 2000 and beyond. In *Proceedings of 7th Biennial IEEE Nonvolatile Memory Technology Conference*, pages 8–12, Albuquerque, NM, USA, June 1998.

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

[30] L. Hellerstein, G. A. Gibson, R. M. Karp, and R. H. Katz. Coding techniques for handling failures in large disk arrays. *Algorithmica*, 12(2/3):182–208, 1994.

[31] J. L. Hennessy, D. A. Patterson, and D. Goldberg. *Computer Architecture: A Quantitative Approach.* Morgan-Kauffman Publishers, 3rd ed. edition, 2003.

[32] R. A. Hill. System for managing data storage based on vector-summed size-frequency vectors for data sets, devices, and residual storage on devices. US Patent 5345584, 1994.

[33] M. Holland, G. A. Gibson, and D. P. Siewiorek. Architectures and algorithms for on-line failure recovery in redundant disk arrays. *Journal of Distributed and Parallel Databases*, 2(3):295–335, 1994.

[34] M. C. Holland. *On-Line Data Reconstruction in Redundant Disk Arrays.* Technical report cmu-cs-94-164, Carnegie Mellon University, Pittsburgh, PA, April 1994.

[35] An analysis of RAID 5DP. HP White Paper, `http://www.hp.com`.

[36] A. Kuratti and W. Sanders. Performance analysis of the RAID5 disk array. In *Proceedings of International Computer Performance and Dependability Symposium*, pages 236–245, Erlangen, Germany, April 1995.

[37] E. Lee and R. Katz. Performance consequences of parity placement in disk array. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 190–199, 1991.

[38] W. Litwin and T. Schwarz. LH^*_{RS} : A high-availability scalable distributed data structure using reed solomon codes. In *Proceedings of the SIGMOD International Conference*, pages 237–248, Dallas, Tx, USA, 2000.

[39] Maxtor Enterprise SCSI Hard Drive Products: `http://www.maxtor.com/en/products/scsi` (Retrieved on April 2004).

[40] J. M. Menon. Performance of RAID5 disk arrays with read and write caching. *Journal of Distributed and Parallel Databases*, 11(3):261–293, July 1994.

[41] J. M. Menon. A performance comparison of RAID5 and log-structured arrays. In *Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing*, pages 167–178, Washington, D.C., Augest 1995.

[42] A. Merchant and P. Yu. Performance analysis of a dual striping strategy for replicated disk arrays. In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, pages 148–157, San Diego, January 1993.

[43] A. Merchant and P. Yu. Analytic modeling of clustered RAID with mapping based on nearly random permutation. *IEEE Transaction on Computers*, 45(3):367–373, March 1996.

[44] R. R. Muntz and J. C. S. Lui. Performance analysis of disk arrays under failure. In *Proceedings of the 16th VLDB Conference*, pages 162–173, Brisbane, Australia, August 1990.

[45] R. Nelson and A. Tantawi. Approximate analysis of fork-join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739–743, 1988.

[46] R. Paquet and M. Nicolett. The cost of storage management: A sanity check. Research note DF-14-6838, Gartner Group, November 2001.

[47] C.-I. Park. Efficient placement of parity and data to tolerate two disk failures in disk array systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(11):1177–1184, November 1995.

[48] D. A. Patterson, G. A. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of ACM SIGMOD 1988 International Conference on Management of Data*, pages 109–116, Chicago, IL, June 1988.

[49] W. Peterson and E. Weldon Jr. *Error-Correcting Codes.* MIT Press, 1972.

[50] K. K. Ramakrishnan, P. Biswas, and R. Karedla. Analysis of file I/O traces in commercial computing environments. In *Proceedings of ACM SIGMETRICS Conference*, pages 78–90, 1992.

[51] H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[52] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, March 1994.

[53] P. Scheuermann, G. Weikum, and P. Zabback. "Disk Cooling" in Parallel Disk Systems. *IEEE Data Engineering Bulletin*, 17(3):29–40, 1994.

[54] P. Scheuermann, G. Weikum, and P. Zabback. Data partitioning and load balancing in parallel disk systems. *VLDB Journal*, 7(1):48–66, 1998.

[55] J. Schindler and G. R. Ganger. Automated disk drive characterization. Technical Report CMU-CS-99-176, 1999.

[56] T. Schwarz. *Reliability and Performance of Disk Arrays.* PhD thesis, University of California, San Diego, 1994.

[57] T. Schwarz and W. Burkhard. Multi-dimensional disk array reliability. Technical report CS93-324., University of California, San Diego, October 1993.

[58] E. Shriver. A formalization of the attribute mapping problem. HP labs technical report HPL-SSP-95-10, HP Labs, July 1996.

[59] E. Shriver. *Performance Modeling for Realistic Storage Devices.* PhD thesis, New York University, New York, NY, 1997.

[60] Storage performance council. `http://www.storageperformance.org`.

[61] H. Takagi. *Queueing Analysis - A Foundation of Performance Evaluation.* North-Holland, 1991.

[62] A. Thomasian. Performance evaluation of RAID5 disk arrays. In *Tutorial, SIGMETRICS'98*, Madison , Wisconsin, June 1998.

[63] A. Thomasian, C. Han, G. Fu, and C. Liu. A performance evaluation tool for RAID disk arrays. Technical Report, Integrated System Lab, CS Department, NJIT, 2004.

[64] A. Thomasian and J. Menon. Performance analysis of RAID5 disk arrays with a vacationing server model for rebuild mode operation. In *Proceedings of the 10th Int'l Conf. on Data Engineering*, pages 111–119, 1994.

[65] A. Thomasian and J. Menon. RAID5 performance with distributed sparing. *IEEE Transactions on Parallel and Distributed Systems*, 8(6):640–657, 1997.

[66] K. Treiber and J. M. Menon. Simulation study of cached RAID5 designs. In *Proceedings of the 1st IEEE Symposium on High-Performance Computer Architecture*, pages 186–197, 1995. also IBM Research Report RJ 9823.

[67] D. Voigt. HP AutoRAID field performance. HP World 1998 Presentation #3354, Auguest 1998.

[68] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, pages 96–108, Copper Mountain, Colorado, USA, 1995.

[69] J. Wolf. The placement optimization program: a practical solution to the disk file assignment problem. In *Proceedings of the 1989 ACM SIGMETRICS International Conference*, pages 1–10, Berkeley, CA, May 1989.

[70] P. Zabback, J. Menon, and J. Riegel. The RAID configuration tool. In *Proceedings of the 3rd International Conference on High Performance Computing*, 1996. also IBM Research Report RJ 10055.

[71] R. Zimmermann and S. Ghandeharizadeh. Continuous display using heterogeneous disk-subsystems. In *Proceedings of the Fifth ACM Multimedia Conference*, pages 227–238, Seattle, WA, November 1997.

[72] R. Zimmermann and S. Ghandeharizadeh. HERA: Heterogeneous extension of RAID. In *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2000)*, Las Vegas, Nevada, June 2000.

[73] B. T. Zivkov and A. J. Smith. Disk cache design and performance as evaluated in large timesharing and database systems. In *Proceedings of 23rd Annual Conference of the Computer Measuremet Group (CMG)*, pages 639–658, Orlando, FL, December 1997.