Trading Capacity for Performance in Disk Arrays

Xiang Yu

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

January 2004

To Mom, Dad and Min

Abstract

During the last two decades, while disk capacity, disk bandwidth and CPU performance have been improving at a rate of about 60% per year following Moore's law, disk access latency has only been improving at a rate of about 10% per year. The performance gap between disk access latency and the rest of the computer system has been increasing exponentially. Moreover, Redundant Array of Independent Disks (RAID) has become a standard approach to improving the bandwidth and capacity of disk subsystems after over a decade of research and development. As a result, disk access latency becomes a performance bottleneck of many I/O intensive applications. Although several techniques, such as striping, mirroring and data replication within a disk track, have been proposed to reduce disk access latency by sacrificing disk capacity, it is not clear how to systematically configure disk arrays to reduce disk access latency and improve disk throughput for a variety of workloads and disk characteristics.

This dissertation proposes a novel way of designing disk arrays that can flexibly and systematically reduce disk access latency while improving disk throughput. We call this new disk array configuration family "SR-Array" because it considers reducing both *seek* time and *rotational* delay in a balanced manner. The dissertation shows, via theoretical and experimental studies, that the SR-Array approach can indeed improve disk access latency significantly compared with existing disk array configuration techniques.

The dissertation makes several contributions. First, we have developed analytical models for disk array configurations, and we show how to use the models to guide disk array designs towards optimal configurations by considering both disk and workload characteristics. Second, we have proposed a robust disk head position prediction

mechanism without any hardware support and a new algorithm to reduce both seek time and rotational delay for the SR-Array disk configurations. Third, we have implemented a prototype disk array system together with an accurate simulator in a layered approach that incorporates the configuration models. Finally, we have demonstrated that the SR-Array approach can indeed improve disk I/O performance significantly for several I/O intensive workloads over existing disk array configuration techniques.

Acknowledgments

I would like to thank my advisor, Prof. Randolph Wang, for his knowledge, youthful spirit, openmindedness and encouragement. This dissertation would not be made possible if without his guidance. I have benefited greatly from the dynamic discussions with him that opened up my thinking, gave me confidence and taught me how to approach research problems. He has inspired many ideas in this dissertation.

I am grateful to Prof. Kai Li and Prof. Larry Peterson for serving as readers in my dissertation committee and for their valuable comments and suggestions on thesis drafts. I am fortunate to have worked with and learned from many faculty members in the Computer Science Department. Prof. Douglas Clark and Prof. Edward Felten guided my academic study and research through my general exams. I appreciate that they have agreed to serve on my dissertation committee. Prof. Kai Li, Prof. Jaswinder P. Singh and Dr. James Philbin have supported my work on the VMMC platform that provided me the necessary skills and confidence of engineering system prototypes. This experience helped me substantially in implementing the experimental framework in my dissertation research. I would also thank our graduate coordinator, Melissa Lawson, and CS staff for all their help during my study.

I would like to thank various funding agencies for their visions and efforts to support my graduate study and research. This dissertation is supported in part by the Scalable I/O project under the DARPA grant DABT63-94-C-0049 and by the National Science Foundation under grant CDA-9624099 and CCR-9984790.

System research requires a lot of teamwork. I would like to thank my fellow students Angelos Bilas, Han Chen, Stefanos Damianakis, Benjamin Gum, Liviu Iftode, Minwen Ji, Dongming Jiang, Sanjeev Kumar, Cheng Liao, Zhiyan Liu, Robert Shillner, Limin Wang, Chi Zhang, Yuanyuan Zhou and all others for their help and for all

I have learned from them. Special thanks to Yuqun Chen for our initial adventure of tracking the rotational movements of disk heads together and for many productive discussions that contributed into the research direction of this dissertation.

No words are enough to describe my gratitude to my parents for their love, nurturing, encouragement and sacrifice. Their own careers in scientific exploration always inspire me to pursue excellence. I am deeply thankful to my wife, Min Wu, whose love and support have made our life in Princeton so much enjoyable.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

During the past two decades, we have witnessed explosive growth of disk areal density and capacity at an annual rate of about 60% [15]. In the same period, CPU performance follows Moore's Law and also maintains the same speed of growth. On the other hand, disk latency has been improving at about only 10% per year [15]. As a result, the performance gap between disk access latency and the rest of the computer system has been increasing exponentially and makes application performance increasingly unbalanced. Moreover, Redundant Array of Independent Disks (RAID) has become a standard approach to improving the bandwidth and capacity of disk subsystems after over a decade of research and development. Applications that leverage rapid growing disk bandwidth and capacity are increasingly limited by slowly improving disk access latency. Although cost per byte and capacity per drive remain the predominant concern of a large sector of the market, a substantial performance-sensitive (in particular, latency-sensitive) market exists. Database vendors today have already recognized the importance of building a balanced secondary storage system. For example, in order to achieve high performance on the TPC-C [46] benchmark, vendors configure disk arrays based on the number of disks involved instead of the

total aggregated capacity of all the drives.

There are many performance-enhancing techniques in the disk array literature. These include striping[30], mirroring[4], replication of data within a track to improve rotational delay[32], and a combination of these, such as the RAID-10 configuration [5, 23, 45], which combines mirroring and striping so that each unit of the striped data is also mirrored. All of these techniques share the common theme of improving performance by using more disks and sacrificing total usable storage capacity. Among all the factors, the following can significantly affect the overall performance of a disk array system.

- characteristics of the disks, such as disk arm movement speed, platter rotation speed, I/O access overhead and disk form factor;
- disk array configurations;
- scheduling policies for disk I/O requests;
- workloads; and
- information redundancy requirements that affect reliability.

While many aspects of the above factors impact the overall performance of a disk array, this study is motivated by the following questions in particular.

- Given a fixed number of disks, what combination of the existing configuration techniques should we choose to use to get optimal performance?
- Are there better alternatives than the existing ones?
- How do the workload characteristics affect the choice of these configurations?
- What are the theoretical models for deriving optimal configurations?

- How does the best configuration depend on the quantity and the basic performance characteristics of the disks used in the disk arrays?

- Given a fixed budget, is it more cost effective to increase memory cache size or to increase the number of disks in the array?

This dissertation proposes a novel way of designing disk arrays that can flexibly and systematically reduce seek time and rotational delay. We have demonstrated, via modeling and experiments with a real implementation, that this approach can improve disk I/O performance significantly over existing disk array configuration techniques. We call the resulting disk array configuration family "SR-Array" in that it reduces two key parts of disk access latency, *seek* time and *rotational* delay, in a balanced manner based on both disk and workload characteristics. Our study focuses on the kind of workloads that occur frequently in office file systems and database transaction systems. We study latency and throughput of disk arrays while considering above performance related factors. Parts of the study have been published [54].

The key contributions of this dissertation are:

- a flexible strategy (SR-Array) for configuring disk arrays and its performance models;

- a robust disk head position prediction mechanism without any hardware support, and a new algorithm for reducing both seek time and rotational delay in disk arrays;

- a prototype disk array system together with an accurate simulator in a layered approach that incorporates various configuration models and position-sensitive scheduling algorithms;

- evaluations of the SR-Array approach that show significant improvements of disk I/O performance in several I/O intensive workloads over existing disk array configuration techniques.

More specifically, we have presented the SR-Array configuration family that flexibly combines striping with rotational replication to reduce both seek and rotational delay. The power of this configuration lies in that it can be flexibly adjusted in a balanced manner that takes a variety of parameters into consideration. We present a series of analytical models that show how to configure the array by considering both disk and workload characteristics.

To evaluate the effectiveness of this approach, we have designed and implemented, in a layered approach, a prototype disk array with an accurate simulator that incorporates the SR-Array and other existing configurations models. In the process, we have developed a method for predicting the disk head location on off-the-shelf SCSI hard drives without special hardware support. This mechanism is a crucial ingredient in the success of the SR-Array configurations because it enables not only our algorithm to reduce both seek time and rotational delay but also the implementation of position-sensitive scheduling algorithms, such as Shortest Access Time First (SATF) [26, 43], across all the disks in a disk array. Because these algorithms involve inter-disk replicas, it would have been difficult, without the head-tracking mechanism, to choose replicas intelligently even if the drives themselves perform sophisticated internal scheduling.

Our experimental results demonstrate that the SR-Array provides an effective way of trading capacity for improved performance over existing disk array configuration techniques in several I/O intensive workloads. For example, under one file system workload, a properly configured six-disk SR-Array delivers 1.23 to 1.42 times

lower latency than that achieved on highly optimized striping and mirroring systems. The same SR-Array achieves 1.3 to 2.6 times better sustainable throughput while maintaining a 15 ms response time on this workload.

This dissertation is organized into several parts. Chapter 2 focuses on mathematical analysis of using disk arrays to improve disk system latency and throughput and introduces a new disk array configuration family: SR-Array. Chapter 3 describes an experimental framework that is designed to drive our empirical study, which includes verifying our theoretical modeling results and getting more insights on other workloads and disk array configurations. Chapter 4 uses the above framework to study various disk array configurations under different workloads. Chapter 5 summarizes this dissertation and Chapter 6 presents a few future research directions.

Chapter 2

Modeling Disk Arrays

This chapter focuses on analytical analysis of using disk arrays to improve disk system latency and throughput. We first make some assumptions of a mathematical model that approximates the behaviors of real disk drives. Based on this mathematical model of a single disk drive, we then analyze different strategies of using multiple disk drives to form a disk array to reduce the latency and improve the throughput.

We start this discussion by reviewing disk modeling basics and analyzing a single disk latency model in Section 2.1. After going over the existing techniques of using multiple disks to improve latency in Section 2.2, we introduce and analyze in Section 2.3 a new disk array configuration strategy called the SR-Array disk array configuration family, which can more effectively reduce latency. In Section 2.4, we discuss modeling throughput of the SR-Array configuration family. We conclude in Section 2.5.

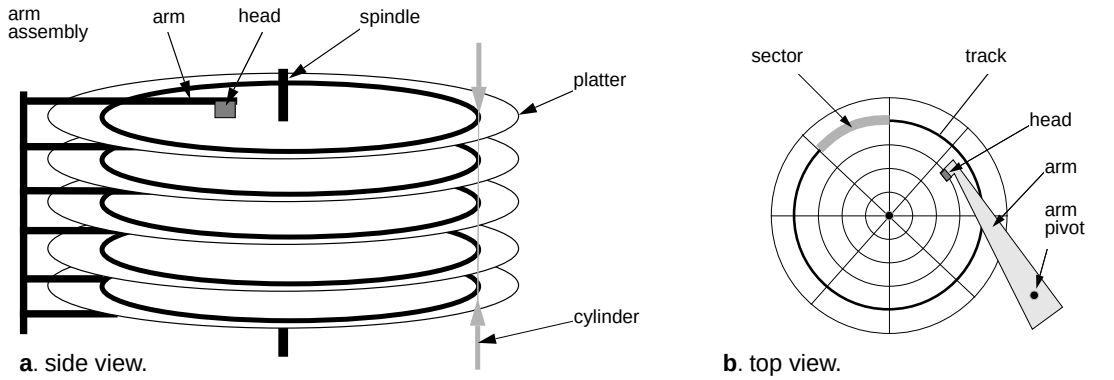


Figure 1: the mechanical components of a disk drive.

Figure 2.1: An illustration of the disk model presented by Ruemmler and Wilkes [36, 37].

2.1 Modeling Single Disk Latency

In the section, we first discuss the modeling assumptions and then discuss disk latency modeling of a single disk.

2.1.1 Modeling Basics and Assumptions

First, we examine how we model a single disk drive. We briefly review the major steps of disk operations. Ruemmler and Wilkes explain disk modeling in greater detail in their papers [36, 37].

Disk Structure

As illustrated in Figure 2.1 [36, 37], one commonly used simple model treats a disk as a a group of one or more constantly rotating donut shaped platters with one disk arm "flying" over each recording surface. The arms are bound to each other and can move from the inner edge of the donut to its outer edge. The arm movement and the rotating platter make it possible for the tip of each arm to reach every position on a

recording surface, where data bits are lined up in concentric circular tracks. A head at the tip of the arm can read or write those bits that pass underneath as the platters rotate.

Data Layout

All tracks that have the same radius but are on different recording surfaces form a cylinder, with each track in the cylinder served by a different read/write head. Therefore, the number of heads inside the disk drive is the same as the number of tracks per cylinder. Because all heads move together when the disk arm moves, these heads are ideally always on the tracks that belong to the same cylinder. Only one head is active to perform a read or write request at any instant. In other words, the disk physically serves at most one request at any instant.

Tracks are usually evenly spaced on recording surfaces. We use track density to represent the number of tracks that lie in a unit length of radius. Today's technological advances make it possible to pack more than 10,000 tracks in one inch. The tiny gaps between adjacent tracks make it increasingly hard to position all the heads accurately on the same cylinder simultaneously. A little disturbance of temperature or other factors can result in one head being positioned on one cylinder while another head could be positioned on a neighboring cylinder. This mal-alignment is one of the practical reasons that the disk drives limit the number of operating heads to one at any given instant of time.

On each track, bits are spaced evenly and linear density (recording density) is used to represent the number of bits in a unit length of the circle. A fixed number of bits (typically 4096 bits or 512 bytes) form a sector. A sector is the smallest addressable unit that a disk exposes to its user.

All cylinders, tracks (or heads), and sectors are numbered. A request uses its

cylinder number, head number and sector number to locate a particular sector. A disk drive can also be considered as a linear list of sectors for a user, who may not care about the internal data layout. Currently, Linear Block Address (LBA) is used to specify the sector that a user desires to access. The firmware on the disk drive translates the LBA to the proper cylinder, head and sector identifiers.

Not all bits on the platter are used to record user data. Some of the bits are used to store redundant information for error correction, mark boundary of accessible sectors, mark identification information of cylinders or make room for mechanical and timing variations (within allowed range) on disk rotation and disk arm and head movement. For a particular family of disk drives, these types of bits usually take a fixed portion of the total bits in a track. Hospodor and Hoagland describe typical data bits arrangements in their paper [21].

The amount of useful bits on a disk is represented by the advertised capacity after the disk is formatted. Occasionally, a portion of the recording surface can lose its recording capability over long term use. In order to keep this from affecting the overall capacity of the disk, a number of spare sectors, tracks, or cylinders are reserved to replace the failed ones.

In order to avoid interference between each other, each bit stored in the disk requires an exclusive area on the platter and maintains at least a fixed distance to the area of other bits. Given a fixed distance between two neighboring bits, an outer track is inherently capable of holding more bits than an inner track because the circumference of a track is proportional to its diameter. To simplify engineering, early generation of disk drives let the outer tracks "waste" some space so that they store the same number of bits as the innermost track. This way, every track has constant number of bits and it is easy for the disk controller to calculate where a particular LBA sector is located. We call this "constant track capacity" layout scheme.

To use the capacity of outer tracks more efficiently, disk drives produced in last ten years divide a disk platter into a number of exclusive zones of consecutive cylinders. All tracks within the same zone must have the same number of sectors. Zones at outer radius have more sectors per track than those at inner radius. By limiting the number of zones, disk controllers only need to look up a small table to locate a sector while keeping the number of wasted bits small. Each zone can also include a number of spare sectors, tracks or cylinders at the end so that the replacements of a small number of defective sectors can always be close to their original locations.

The ratio between the number of sectors on the outermost track to the number of sectors on the innermost track is usually less than 2 for 3.5 inch disk drives. For the Seagate Cheetah X15 family of disk drives, the ratio is around 1.3. The ratio is around 1.5 for both the Seagate Cheetah 18LP and 36LP families of disk drives used in our experimental platform and empirical studies.

Performance Parameters

A common approach to modeling the access time of disk data is to sum the seek time, rotational latency and data transfer time [32, 42]. We also use this model in our analytical study.

In order to read/write a specified sector, the disk firmware has to first translate the LBA of the sector into a tuple of cylinder, head and in-track sector numbers. The firmware looks up a table and moves the arm onto the target cylinder first. This arm movement between cylinders is called a seek, which usually has four phases: acceleration, coasting at constant speed, deceleration, and settling. For short seeks, the coasting phase is not needed. The first three phases allow the head to quickly but approximately reach the target cylinder, while the settling phase fine-tunes the head position.

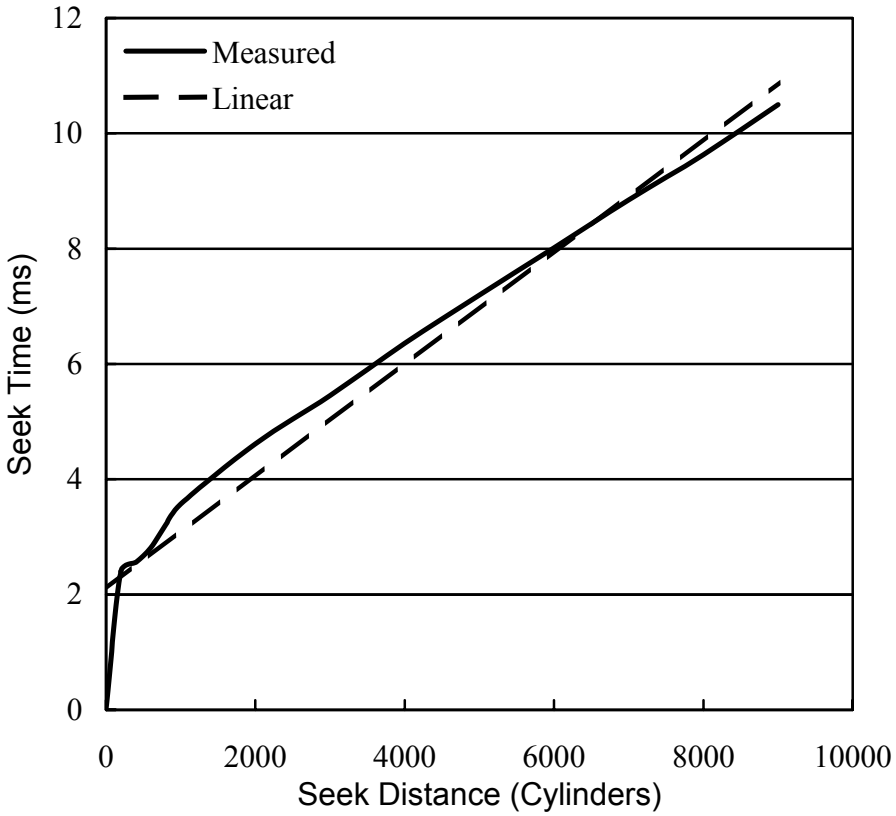


Figure 2.2: An example of a SCSI disk seek profile: seek time as a function of the seek distance. The graph includes a measured curve and a linear approximation of the measured curve.

Figure 2.2 is a typical seek profile of a disk. For large seek distances, the seek time is close to a linear function of the seek distance because the arm moves over a substantial distance at a nearly constant coasting speed. However, for short distances, the arm spends most of its time in mechanical acceleration and deceleration. If the disk arm moves randomly, modeling seek time as $T_s = a + bS$ [14] is a good approximation most of the time, where a and b are constants and S is the number of cylinders the arm travels. The seek time of the state of art commercial hard drives is about several milliseconds. Note that there are several other possible mathematical models for the seek time, which model non-linear time behavior of the small distance seeks more accurately [37, 17, 20]. As the acceleration speed of the disk arm grows

over the years, the non-linear seek time range becomes smaller and the arm travels well below 10 percent of the total number of cylinders during that time. Therefore, we use the simple linear model here to simplify our analysis without severely compromising our results.

After the head takes some time to settle on the target track, it starts to read the bits passing underneath and waits for the marker bits corresponding to the target sector. Once it identifies the marker bits, it starts to read or write the target sector, which includes some redundant bits for error correction. The waiting time is usually called the "rotational delay" and the read/write time is called the "data transfer time."

Because the disk platter rotates at a constant speed, the waiting time can be expressed as $T_r = cR$, where c is a constant and R is the angle that the platter rotates between the time when the head arrives on the track and when the head starts accessing the sector. The full rotation time of state of art commercial hard drives is about several milliseconds, which is comparable to a full seek delay.

To read/write a sector, the head needs to read/write from the first bit to the last bit of the sector. For a track that has all its N sectors distributed evenly, the read/write time of one sector is $1/N$ of the full rotation time. For today's disks with high linear density and fast rotation speed, the time for the disk head to fly over a few sectors and transfer the information is about tens of microseconds. Although a sector in the innermost zone requires more time for the head to fly over than a sector in an outer zone does, its read/write time is still insignificant compared to the seek and rotation times.

There is also time spent in the operating system, the I/O controller, and the disk controller. For small accesses over a few sectors, we model these times as a fixed overhead.

In summary, for small accesses studied in this thesis, we ignore the data transfer time and model the time cost to include the following:

1. seek time that is proportional to the seek distance;
2. rotational latency, which is proportional to rotational angle;
3. constant overhead time that an access incurs that is independent of the seek distance and rotational angle. We refer to this time as T_o in later analysis.

Now, we examine the performance data collected from a hard drive used throughout this thesis and present the model we build for the Seagate ST39133LWV SCSI hard drive. Later in the thesis, we compare the results from real disks with the results derived by our model to show that our model is indeed a good approximation for computing latency and throughput of some workloads.

Our first experiment measures the minimum time between reading two sectors on two different cylinders. This measurement tries to minimize the rotational delay by only measuring the shortest possible time between reading one sector on one cylinder and another sector on a different cylinder.

Figure 2.3 gives the minimum read time (given on the Y-axis) between sectors at different cylinder distances (given on the X-axis). As we can see, the access time can be modeled as a linear function with an overhead of $2.7ms$ and maximum scalable seek time of $9.2ms$ over 9772 cylinders.

We also measure the full rotational time by repeatedly reading one sector and calculating the average time between two consecutive readings. The result we obtain is 10026 RPM(Rotations Per Minute), which is close to the manufacture advertised 10000 RPM for the disk drive.

Knowing that every track on the disk has a minimum of 230 sectors, we can bound the maximum time for the head to scan one sector by $60s/10000/230 < 30us$, which

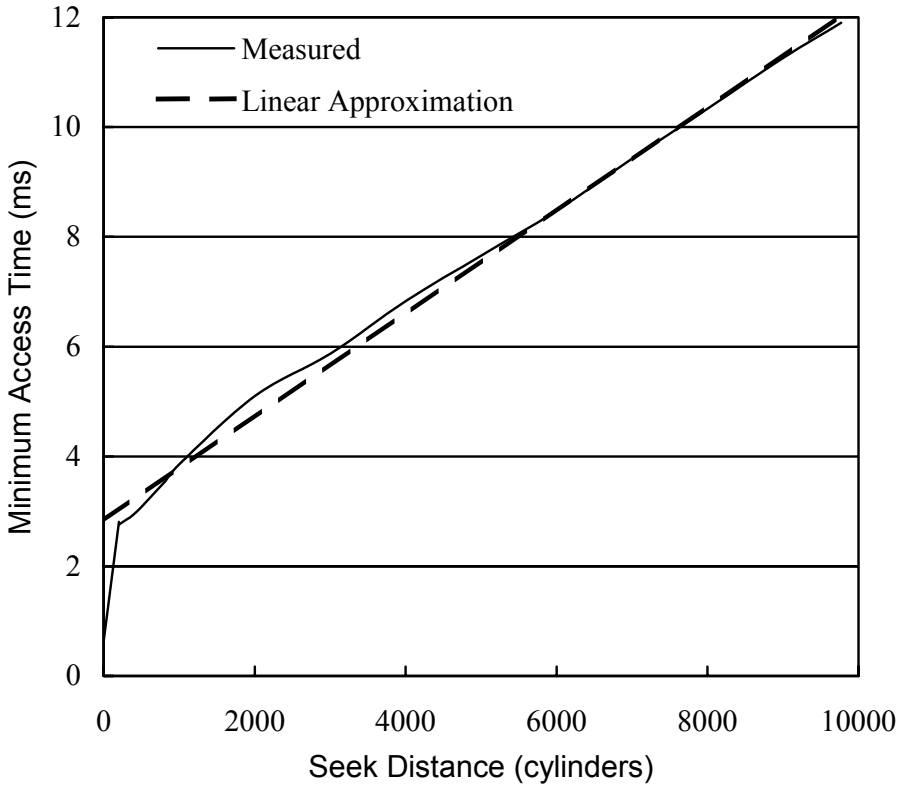


Figure 2.3: Minimum access time as a function of seek distance measured on a Seagate ST39133LWV disk drive. The minimum access time is the shortest possible time between reading one sector on one cylinder and reading another sector on a different cylinder at a particular seek distance.

is indeed insignificant comparing to the seek, rotation time, and the fixed overhead if only a small number of sectors are accessed in a given request.

2.1.2 Expected Latency of Small Random Access

We examine the performance of random accesses. Various researchers[10, 44, 27, 20] have proved that the average seek distance of random accesses on a constant track capacity drive is $N_S/3$, where N_S is the total number of cylinders of the disk. Therefore, if the constant overhead T_o (defined near the end of Section 2.1.1) incurred by every access is excluded, the average random seek time is $S/3$, where S is the the

full seek time excluding T_o.

There are different ways of calculating the above average time. Presented below as an example is a method using integral calculus. We also employ similar analysis later in this chapter. If we assume constant track capacity, the arm is evenly distributed on different cylinders on the platters. Let v_s be the coasting speed of the arm and R and r are the radius of the outermost and the innermost cylinders respectively. The average seek time S_1 is the average of the seek time from a cylinder at radius position x to another cylinder at radius position y, which is $\frac{|y-x|}{v_s}$, over all possible combinations of x and y.

$$\begin{aligned} S_1 &= \frac{\int_{x=r}^{R} \left(\int_{y=r}^{R} \frac{|y-x|}{v_s} dy \right) dx}{\int_{x=r}^{R} \left(\int_{y=r}^{R} dy \right) dx} \\ &= \frac{1}{3} \frac{R-r}{v_s} \\ &= \frac{S}{3} \end{aligned} \tag{2.1}$$

Once the disk arm moves to the target track, it needs to wait for an average of half a rotation to reach the target sector. So the average rotational delay is $R_1 = R/2$, where R is the full rotation time. This is a well known result that Houtekamer [11] and many others have derived it. After the rotational delay, the head spends a fixed amount of time per sector to access the data. As discussed above, when accessing small number of sectors, the disk spends most of its time on seek and rotation instead of transferring useful data.

Overall, The average delay (also called access time) of a random access T_1 can be

modeled as

$$T_1 = \frac{S}{3} + \frac{R}{2} + T_o \tag{2.2}$$

2.1.3 Effect of Zoning on Small Random Access Time

Almost all the current hard drives use zoning to achieve a track capacity close to the theoretical maximum capacity $2\pi x D_l$, where x is the radius of the track and D_l is the number of bits per unit length on a track, i.e. bit density. Therefore, even for random accesses, the distribution of the disk head positions is no longer uniform across all cylinders. Instead, the heads are more likely to be positioned on the outer cylinders, which have more sectors on them. Now, we recalculate the average seek time based on the assumption that (a) R and r are the radius of the outermost and the innermost cylinders respectively; and (b) each cylinder has a capacity proportional to $2\pi x D_l$. We also use t to represent the ratio between R and r.

$$\begin{aligned} v_s &= \frac{R-r}{S} \\ &= \frac{t-1}{S} r \end{aligned} \tag{2.3}$$

We introduce weights of $2\pi x D_l$ and $2\pi y D_l$ into the average calculated without the consideration of zoning in equation (2.1):

$$S_1' = \frac{\int_{x=r}^{R} \left(\int_{y=r}^{R} (2\pi x D_l)(2\pi y D_l) \frac{|y-x|}{v_s} dy \right) dx}{\int_{x=r}^{R} \left(\int_{y=r}^{R} (2\pi x D_l)(2\pi y D_l)\, dy \right) dx}$$

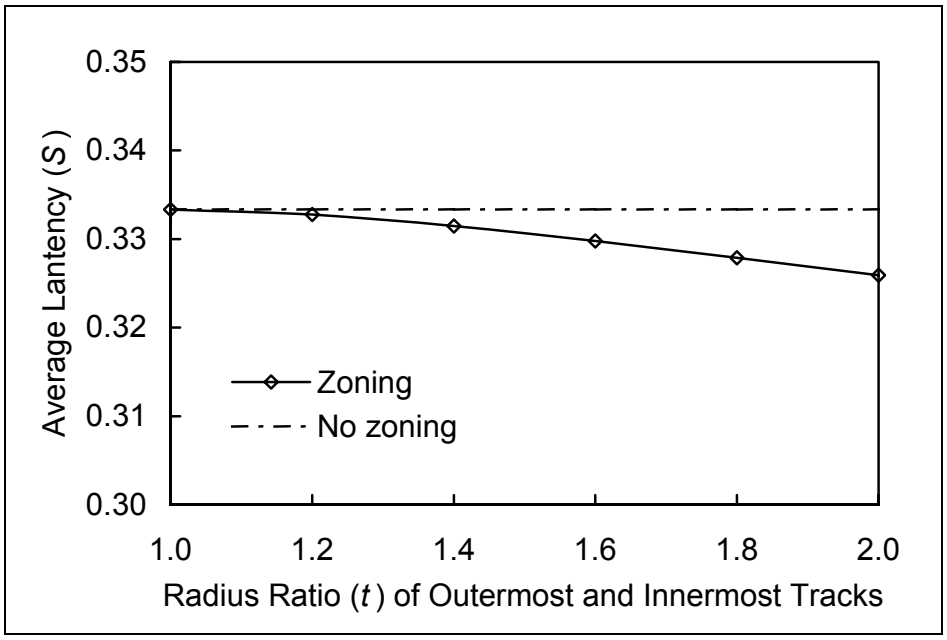


Figure 2.4: Average seek time, in terms of the fraction of the full seek time, as a function of the ratio between the outermost track radius and the innermost track radius for two kinds of bit layout schemes: one uses constant track capacity and does not use zoning; the other uses zoning and records close to the maximum number of sectors allowed under a given linear bit density.

$$
\begin{aligned}
&= \frac{\int_{x=r}^{R}\left(\int_{y=r}^{R} xy\frac{|y-x|}{v_s}dy\right)dx}{\int_{x=r}^{R}\left(\int_{y=r}^{R} xydy\right)dx} \\
&= \frac{\frac{4}{15}\left(R^5-r^5\right)-\frac{4}{3}R^2r^2\left(R-r\right)}{v_s\left(R^2-r^2\right)^2} \\
&= \frac{\frac{4}{15}\left(t^5-1\right)-\frac{4}{3}t^2\left(t-1\right)}{\left(t^2-1\right)^2\left(t-1\right)}S
\end{aligned}
\tag{2.4}
$$

Figure 2.4 illustrates the difference between S_1' and S_1 with the ratio t between 1.0 and 2.0. For a 3.5 inch disk drive, t is usually between 1.5 and 2.0. For a 2.5 inch disk drive, t is about 1.25. The data shows that the difference is smaller than 3 percent in both cases. This means that using equations (2.1) and (2.2) to model

disks with zoning not only simplifies the analysis but is also a good approximation.

2.1.4 Small Access with Locality

We have considered a uniform distribution of requested sectors in the last two sections. However, this uniform distribution assumption is not always true and many workloads exhibit significant locality.

For example, in the UNIX Fast File System [31], all cylinders are divided into several different cylinder groups, each of which includes several consecutive cylinders. Obviously, seeks within the same cylinder group are often shorter than seeks across cylinder groups. FFS assumes that files under the same directory are likely to be accessed together and attempts to place them under the same cylinder group to reduce the seek time. For a typical office workload, this layout scheme significantly reduces the average seek distance for the disk arm. We call this phenomenon "seek locality".

In order to quantify seek locality, we introduce a "seek locality index" L to be the ratio between the average seek distance of a uniformly distributed random access workload and the observed average seek distance in the workload. This way, the average of the linear part of seek time is reduced to

$$S_{1L} = \frac{S}{3L} \tag{2.5}$$

Compared to reducing seek time through layout techniques, reduction of rotational delay is harder. There are several reasons behind the difficulty.

First, the disk drive interface currently only reveals a linear list of sectors to the operating system and, therefore, special effort is needed to discover the layout of sectors. Furthermore, zoning makes the effort even more difficult because different

tracks may have different number of sectors and sectors n and $n+1$ may no longer be close to each other.

Second, the rotational delay of a request depends not only on the location of the sector on the disk but also on the timing of the request. Because the disk rotates constantly, the relative position of the head and the target sector keeps changing. A small delay may cause the head to just miss the target sector after it settles on the target track and, as a penalty, it has to wait another full rotation for the target sector. Therefore, even if the operating system places the files close to each other rotationally, the delay between their accesses is not guaranteed to be minimal.

The above considerations lead us to believe that the impact of software decisions on rotational locality is less profound for non-sequential I/Os[1]. Results from simulation presented in Chapter 4 also confirms that this is a good approximation. In the rest of the analysis, we shall approximate the random access delay T_{1L} as

$$T_{1L} = \frac{S}{3L} + \frac{R}{2} + T_o \tag{2.6}$$

2.1.5 Disk Latency, Capacity and Cost Trend

Over the past two decades, we have witnessed the explosive growth of disk areal density, which is at an annual rate of about 60% [15]. The reduction of the size of the disk head sensor using a giant-magnetoresistive(GMR) technology increases the track density while the reduction of the platter film thickness results in greater linear density. Fueled by the improvement of both track and linear density, the growth has been close to 100% in past 2-3 years. The rapid growth of linear density also makes segments of a few sectors or a few kilo-bytes of data occupy a smaller fraction

[1]For sequential I/Os not focused by this study, the data transfer time, instead of access delay, is dominant.

of a track than ever before and therefore results in much less bit transfer time for these small data accesses, which can be typical in database applications. As the disk mechanical speed has been improving at about only 10% per year [15], the latency of small I/Os increasingly dominates. As a result, disks are becoming increasingly unbalanced in terms of the relationship between capacity and latency.

Although cost per byte and capacity per drive remain the predominant concerns, a substantial performance-sensitive (and, in particular, latency-sensitive) market exists. Database vendors today have already recognized the importance of building a balanced secondary storage system and use several latency reduction techniques to improve the overall performance of their systems.

2.2 Improving Latency by Using Disk Array

In the last section, we have discussed how to model the latency for a single disk. In this section, we discuss various techniques to improve the latency by using a disk array composed of multiple disks.

2.2.1 Motivation and Background

Because the average latency is a function of maximum seek time, disk full rotation time, and the overhead, disk manufacturers can reduce one or several of them to improve latency. They can make disks rotate faster, disk arms move faster, or disk access overheads smaller, but these improvements are subject to technology and cost limitation.

When these limits are reached, it is hard to continue improving latency by using only one disk. By placing multiple disk arms evenly on a cylinder and choosing a head closest to the target sector, we may be able to reduce the rotational relay.

We can also restrict the movement of each arm to a subset of disk cylinders; the approach reduces the range of arm movement and, therefore, the average seek time. Calderbank et al. presented a performance model for a two headed disk [3]. However, the current market focus on improving the cost/capacity ratio appears to preclude the commercial development of solutions involving multiple arm assemblies per disk despite the presence of applications that demand lower latency. A more practical alternative is to use a disk array that can improve both latency and throughput.

There are several known techniques that use multiple disks to reduce seek time, rotation time or both. For example, striping reduces seek time [30] and mirroring reduces both seek time and rotational delay [4]. These two techniques are widely used in industry. Ng et al. use rotational replication to reduce rotational delay [32]. However, this technique has not been implemented so far. One contribution of this thesis is to provide a prototype implementation and analysis of this rotation time reduction technique on current off-the-shelf commercial disk drives. We delay the discussion of the implementation details to Chapter 3.

2.2.2 Striping Analysis

Figure 2.5 shows an example of striping, where one full disk of data is evenly spread over n disks. This seek reduction technique effectively "transforms" disks into disks with the same rotational delay but $1/n$ the maximum seek distance of the original disk. Therefore, the average access time for one random request in equation (2.2) in our disk model is reduced to $T_{stripe}(n)$.

$$T_{stripe}(n) = \frac{S}{3n} + \frac{R}{2} + T_o \tag{2.7}$$

The cost of this latency reduction is that only $1/n$ of overall disk space is used.

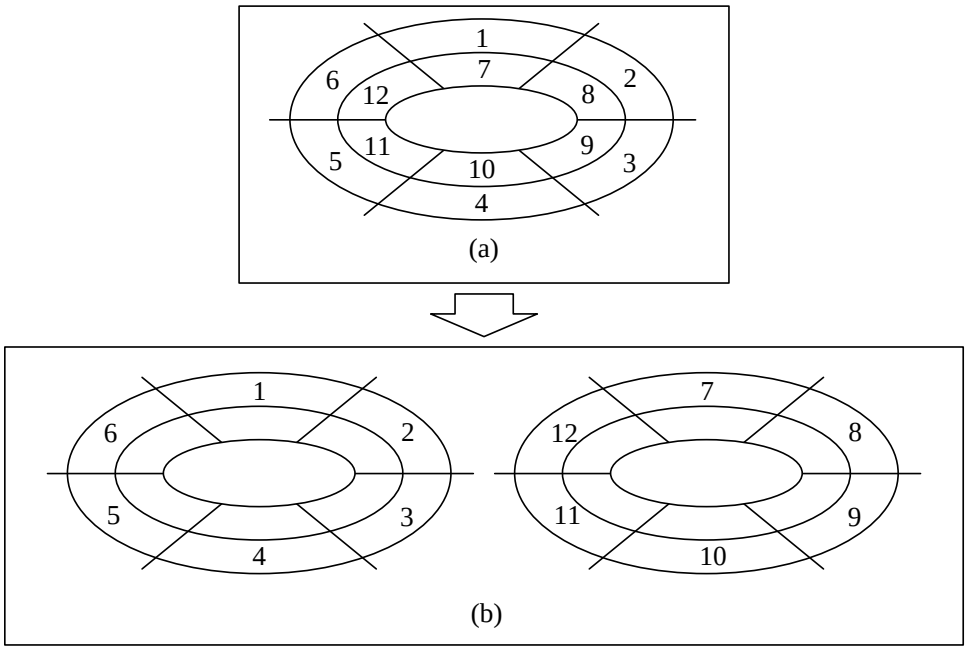


Figure 2.5: An illustration of 2-way striping: (a) the original disk with 12 sectors; (b) the data is striped to two disks, each of which holds 6 sectors.

2.2.3 Rotational Replication Analysis

Figure 2.6 shows an example of rotational replication, where each sector on a disk has a total of n replicas on the same track. This way, each disk can only hold $1/n$ of the total original data. Therefore, like the striping technique, we use n disks to hold one disk of data. However, unlike the striping technique, we now have multiple replicas of the same data, and only need to access the closest replica in a read. For writes, on the other hand, all the replicas have to be written before the benefit of future read latency reduction of the same data can be fully exploited as above.

Evenly Positioning Rotational Replicas

One way to place multiple replicas on the same track is to separate them evenly as shown in Figure 2.6(b). This way, the rotational delay the disk head has to

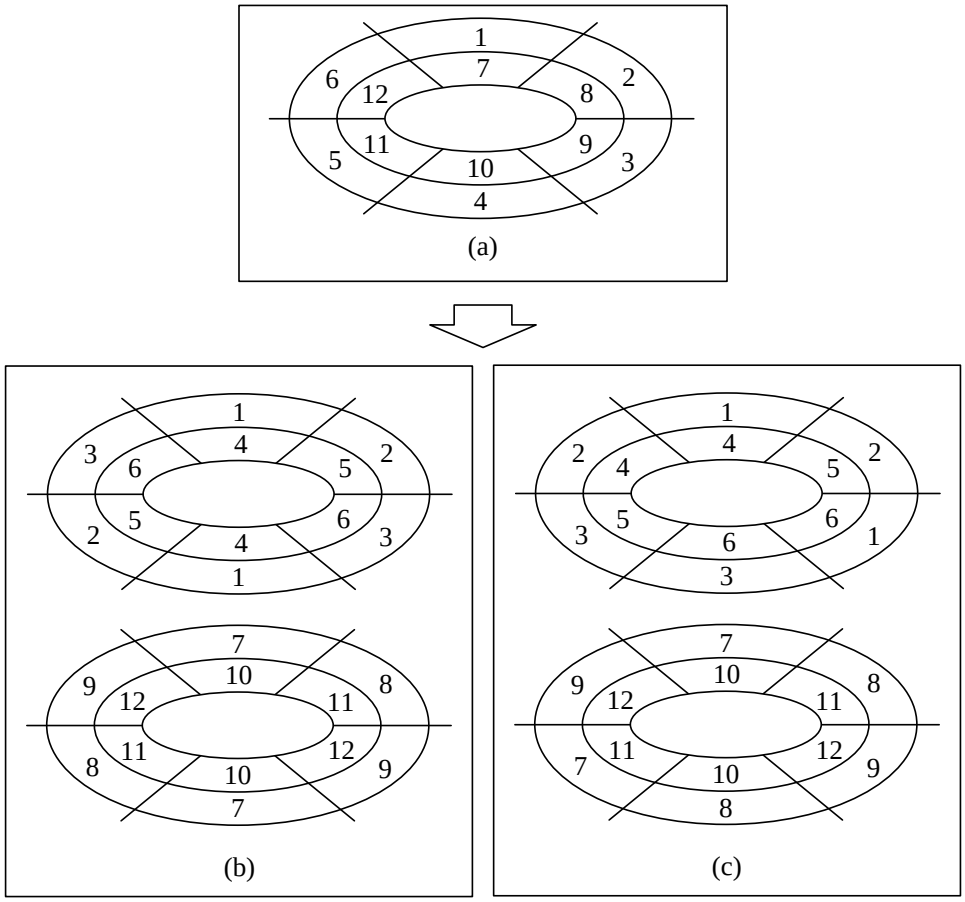


Figure 2.6: An illustration of 2-way rotational replication: (a) the original disk with 12 sectors; (b) the data is spread to two disks, each of which holds 6 sectors and rotationally has 2 copies of each sector spread evenly in the same track; (c) the data is spread to two disks, each holds 6 sectors and rotationally has 2 copies of each sector located randomly in the same track.

incur for a random request falls into a uniform distribution from 0 to $1/n$ of full rotation. Therefore, the average rotational delay is reduced to $R/2n$, $1/n$ of the original average, and the average latency of one random request in equation (2.2) is reduced to $T_{rrep\_read}(n)$.

$$T_{rrep\_read}(n) = \frac{S}{3} + \frac{R}{2n} + T_o \tag{2.8}$$

Rotational replication makes the disks system appear as if it rotates n times faster.

In order to always have n replicas to choose from for a read, we need to update all replicas upon each write request. This approach may worsen write performance. In the case of replicas spread evenly in the track, the disk head has to rotate a fraction $(n-1)\cdot\frac{1}{n}$ of the full rotation to write $n-1$ more replicas after it reaches the first one. Hence, we model the write time as $T_{rrep\_write}(n)$.

$$T_{rrep\_write}(n) = \frac{S}{3} + \frac{(2n-1)R}{2n} + T_o \tag{2.9}$$

Randomly Positioning Rotational Replicas

Another way is to put multiple replicas randomly in the same track as shown in Figure 2.6(c). The average rotation time $TR_{read}(n)$ is the average of the minimum of n independent random variables between 0 and R. It is calculated as the sum of all possible minimum values x multiplied by the probability of realizing that minimum value $n\left(\frac{R-x}{R}\right)^{n-1}$.

$$\begin{aligned} TR_{read}(n) &= \int_0^R xn\left(\frac{R-x}{R}\right)^{n-1} dx \\ &= \frac{n}{R^{n-1}}\left(\frac{R^n}{n} - \frac{R^n}{n+1}\right) \\ &= \frac{R}{n+1} \end{aligned} \tag{2.10}$$

On the other hand, the average rotation time $TR_{write}(n)$ for writes is the average of the maximum of n independent random variables between 0 and R. This means that the average of $R - R_{write}(n)$ is the average of the minimum of n independent random variables between $R-0$ and $R-R$, which turns out to be the same situation of

calculating $TR_{read}(n)$.

$$\begin{aligned} TR_{wrtie}(n) &= R - TR_{read}(n) \\ &= \frac{nR}{n+1} \end{aligned} \tag{2.11}$$

The above results show that reads of evenly distributed replicas have lower average rotational delay than reads of randomly distributed replicas. This is reversed for writes, however.

An Invariant: The Total Time of One Read and One Write

From the above results, we notice that the sum of the average rotational delay of reads and that of writes is one full rotation time R for both even and random placement of replicas. Assuming the disk head is at a random position on the track when it settles, we can further prove that the sum of the expected rotational delay for one read and one write is always one full rotation time, no matter how many replicas are there and how the replicas are positioned in a track.

The basic idea is that for every starting head location x_r that satisfy $a_{prev} < x_r \leq a_{next}$, where a_{prev} is the position of the last replica the head just has passed and a_{next} is the position of the next replica a_{next}, we can construct another starting head location $x_w = a_{prev} + a_{next} - x_r$ that satisfies $a_{prev} \leq x_w < a_{next}$. This way, the read time from head location x_r is $a_{next} - x_r$ for reading the closest replica at a_{next}, while the write time from head location x_w is $R - (x_w - a_{prev})$ for writing all the replicas. Therefore, the sum of the read and the write times is $a_{next} - x_r + R - (x_w - a_{prev}) = R$. Based on this construction, each starting head position for reads matches a unique starting head position for writes. Therefore, the sum of the expected time for one read and the expected time for one write is R, which is independent of the number of replicas

and the location of replicas.

Implication of The Invariant

This invariant means that the time saved on one read because of the existence of the replicas is the extra time spent for one write, which has to go through all these replicas on the same track. For a particular workload that has more reads than writes, making more replicas can reduce overall latency. If reads and writes are equally frequent, the number of replicas(n) does not change the average overall latency. If writes are more frequent than reads, the approach with no replication is always the best, if replicas must be written in the foreground. When disks have idle time, some replicas can also be written during idle periods in order to improve write performance. This strategy is discussed in later sections.

Another implication of this invariant is that there is always a strategy better than random placement of the replicas. When reads outnumber writes, the evenly spaced replication strategy is better. When writes outnumber reads, the approach of having no replicas is better because of better write time.

In Chapter 3, we discuss the implementation of evenly spaced rotational replicas. Because accessing every replica has a small overhead in the implementation, increasing the number of replicas after a certain degree does not help further improve the average latency even when reads outnumber writes.

2.2.4 Mirroring Analysis

Figure 2.7 shows an example of mirroring, where one full disk of data is replicated on n disks.

Mirroring has some similar aspects as rotational replication. First, mirroring uses

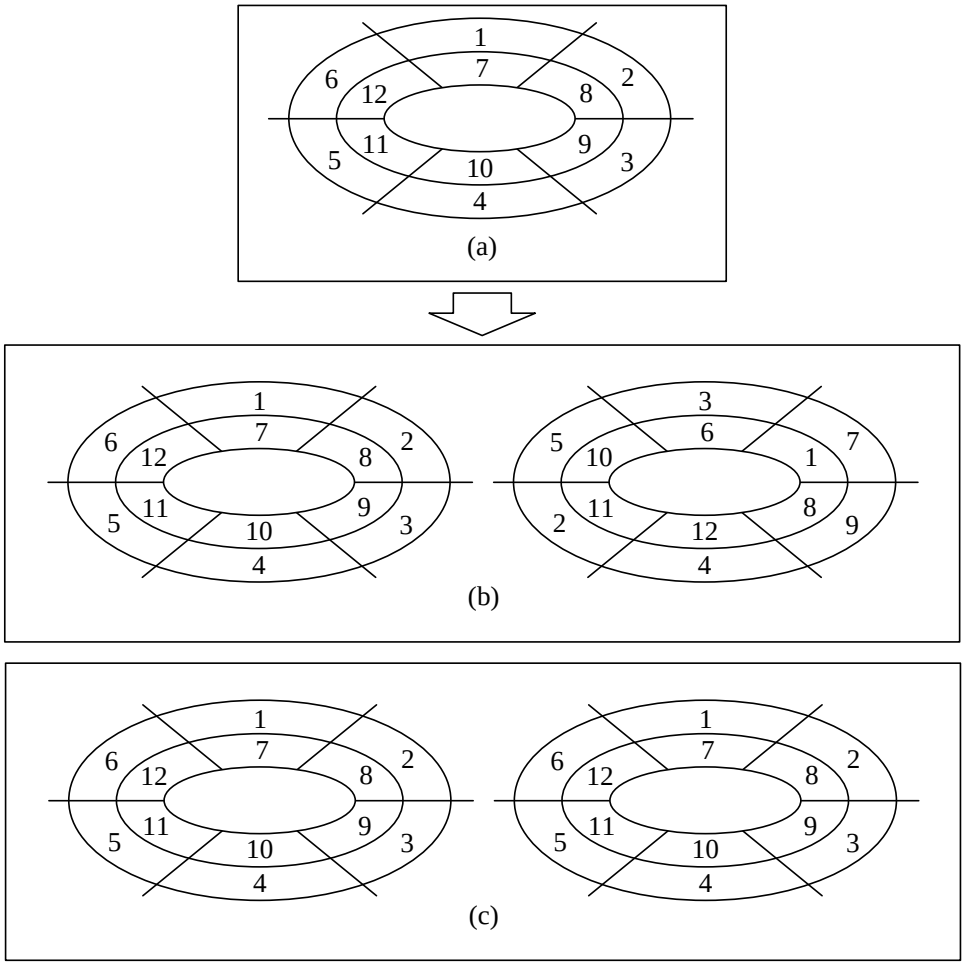


Figure 2.7: An illustration of 2-way mirroring: (a) the original disk with 12 sectors; (b) the data is copied to a second disk, which also holds the same 12 sectors but in a random layout; (c) the data is copied to a second disk, which also holds the same 12 sectors but in the same layout.

multiple replicas to reduce the read access time at the cost of only using $1/n$ of total disk space. Second, mirroring also needs to write all replicas in order to effectively reduce the read time.

However, mirroring differs from rotational replication in several important ways. First, mirroring places replicas of the same data on different disk drives. Because disks tend to fail independently, there is no data loss in a mirrored system unless all n disks fail, whereas a system that employs rotational replication alone loses $1/n$ of total data

per failed disk drive. However, this data reliability does not come for free because mirroring needs to use all the disk heads for one write, whereas rotational replication only needs one. Later, we show that mirroring uses disk heads less efficiently than rotational replication does. In terms of disk modeling analysis, mirroring is more complicated than both striping and rotational replication because of the fact that data resides on different disks, whose heads may not be moving together.

There are several configurations of mirroring with minor differences in data layout and read/write strategies. We discuss some of them in the following sub-sections.

Random Mirroring without Rotational Position Tracking

Random mirroring assumes that all the disk arms and the replicas for a sector in different drives are at independent random positions as illustrated in Figure 2.7(b).

Bitton and Gray analyze both read and write latency for this case [4]. They assume that the rotational positions of the heads and the data sectors are not available to the algorithm for picking which disk to read. Therefore, the algorithm chooses to read the replica with the shortest seek time. For a write, the algorithm spends at most the longest seek time to reach the farthest replica. For both reads and writes, an average time of $R/2$ is spent on rotational delay after each seek.

Their results show that (1) the expected seek distance for reads is $\frac{S}{2n+1}$; and (2) the expected seek time for writes is approximately $S\,(1 - I_k)$, where $I_k = \prod_{k=1}^{n} \frac{2k}{2k+1}$. I_k drops from 0.53 for $n = 2$ to 0.27 for $n = 10$. When $n = 1$, the case degenerates to one copy and is consistent with our earlier random seek distance result. In summary, we have read latency T_{mirror_read} and write latency T_{mirror_write} as the following.

$$T_{mirror\_read}(n) \;=\; \frac{S}{2n+1} + \frac{R}{2} + T_o \tag{2.12}$$

$$T_{mirror\_write}(n) = S\left(1 - \prod_{k=1}^{n} \frac{2k}{2k+1}\right) + \frac{R}{2} + T_o \quad (2.13)$$

Random Mirroring with Rotational Position Tracking

With rotational position tracking, we can do better at picking the closest copy by considering the minimum of all n positioning times, which are sums of both seek time and rotational delay. When $S \gg R$, equation (2.12) and (2.13) are good approximations for average read and write latency.

On the other hand, when $R \gg S$, picking the replica with shortest rotational delay is close to the optimal strategy. Using this strategy, we can achieve an average rotational delay that is the average of the minimum of n random independent rotational delays, which is calculated in equation (2.10) as $\frac{R}{n+1}$. Writes are also modeled as the sum of the expected seek time and the average of the maximum of n random independent rotational delays.

$$T'_{mirror\_read}(n) = \frac{S}{3} + \frac{R}{n+1} + T_o \quad (2.14)$$

$$T'_{mirror\_write}(n) = \frac{S}{3} + \frac{nR}{n+1} + T_o \quad (2.15)$$

This result is the same as that achieved under random rotational replication.

Both results for the above strategy are upper bounds for the expected minimum of n different sums of seek time and rotational delay. Intuitively, when R is close to S, the replica with the minimum positioning time is likely to be neither the one with the shortest seek distance nor the one with the shortest rotational delay. Gum derives an approximation for read latency in his Ph.D. thesis [16] as the following.

$$T''_{mirror\_read}(n) = \sqrt{\frac{\pi RS}{4n}} + T_o \quad (2.16)$$

We can see that the overhead independent part of $T''_{mirror\_read}(n)$ improves proportionally to $\sqrt{n}$. Currently, we are not aware of any model for write latency that considers rotational position. We study both read and write latencies of mirroring again in our empirical study in a later chapter.

Synchronized Mirroring

Synchronized mirroring has all the disks always lay out the same sector of data at the same position as illustrated in Figure 2.7(c).

By deploying a circuit that connects all n disk drives in the mirroring set, we can guarantee that platters in all the drives have exactly the same rotational speed. By making all the disk arms always move together and having all the replicas spread out evenly on different drives, read latency can be reduced to T_{mirror_sync} because the heads can reach the replica on their own drive in evenly separated gaps. Its read latency is the same as that of rotational replication, its write latency is also similar to that of evenly spread rotational replication.

$$T_{mirror\_sync}(n) = \frac{S}{3} + \frac{R}{2n} + T_o \tag{2.17}$$

Dishon and Liu [9] have considered another form of synchronized D-way mirroring. They place the replicas at rotationally identical positions and have the heads of all disks reach the corresponding sectors at the same time. This way, the multiple replicas can be written at nearly the same time. This reduces the write latency so that it is the same as the read latency $T_{mirror\_sync}(n)$. However, this approach does not improve read latency because it allows no rotational delay reduction for reads. Both the average read and write latencies in this case are the same as those in the single disk case.

Note that having all the arms move together to the target cylinder is a requirement to attain the above expected latency. Otherwise, even if all the drives rotate at the same speed and the replicas are evenly spread relative to the track, the seek times and the landing rotational positions of all the heads on the target track become independent random variables. This scenario is similar to the random mirroring case discussed earlier. Therefore, there is no particular benefit for having all the drives rotate at the same speed without synchronizing arm movement on all disk drives. Nevertheless, it is a common practice to have the same data layout on all disks because it trivializes the tracking of all replicated sectors in the disk array.

Although synchronized mirroring has the same read/write latency as rotational replication on a single disk, it does have a limitation on the overall number of requests that the disk array can handle simultaneously. When there are many pending read requests, it can only serve one request at a time while random mirroring can have different heads serve different read requests simultaneously.

Mirroring Advantage and Limitation

In summary, with or without the restriction on disk arm movement, synchronized mirroring has no apparent benefit over random mirroring when the throughput of the entire mirrored system is more important than expected latency of one request. Therefore, we will only use the strategy of random mirroring with the same sector layout order on all disk drives when evaluating mirroring in this thesis.

Both striping and rotational replication only reduce one term of the latency equation to $1/n$: they can improve either the seek time or the rotational delay, but not both. When n is sufficiently large, these two techniques face a diminished return of latency reduction of only one term. When rotational positioning is considered in picking a disk drive to serve a request, random mirroring performs well because it

can achieve a better result by considering the sum of seek time and rotational delay together. Moreover, random mirroring also provides better reliability.

Mirroring's low latency and reliability does not come for free. Compared with the other two techniques, mirroring does worse on writes when there are many pending writes. Both striping and rotational replication can have different disk drives execute different write requests simultaneously because each piece of data only exists on one drive. This is not the case with mirroring.

In the area of disk modeling, we are not aware of any study that gives good and simple closed-form results for write latency on mirroring. The lack of simple models makes it difficult for a disk array builder to find good configurations without resorting to empirical results.

2.2.5 Summary

There are other latency reduction techniques [47, 55] besides the ones discussed above. The common theme is to trade the fast improving capacity for improved latency, which improves slowly. By using a disk array with multiple disks to hold one disk worth of data, we can achieve performance improvements. We can reduce the maximum seek time to improve both read and write latencies or make more copies to increase the choices a system has in satisfying read requests. In general, with more replicas, the read performance improves but the write performance degrades.

We also find that each techniques discussed above has its own strength and could be best suited for certain workloads. In later sections, we also discuss the throughput model, which considers the reordering of multiple outstanding requests to improve performance in a disk array.

2.3 SR-Array Configuration Family

In the last section, we have discussed several individual latency reduction techniques and found that each technique has its own strengths and weaknesses. In this section, we discuss ways to combine individual techniques and form a new disk array configuration family. We also discuss the modeling of several configurations.

2.3.1 Combining Striping and Mirroring

Striping and mirroring can be combined to provide both the reliability of mirroring and the seek reduction of striping resulting in overall read latency reduction while lowering the cost of writing because of the lower degree of mirroring. Figure 2.8 shows an example of a disk array with both striping and mirroring.

While striping has the conventional name of RAID-0 and 2-way mirroring of RAID-1, the combination of the two is also widely used and is called RAID-10.

Because mirroring is not well modeled, the combination is not well modeled either. We only derive an approximation of read latency here for $D = D_s \cdot D_m$, where D_s is the number of disks devoted to striping and D_m is the number of disks devoted to mirroring. As a corollary of equation (2.16), we get the average read latency $T_{SM}(D_s, D_m)$, when S/D_s and R are close.

$$\begin{aligned} T_{SM}(D_s, D_m) &= \sqrt{\frac{\pi R \frac{S}{D_s}}{4D_m}} + T_o \\ &= \sqrt{\frac{\pi RS}{4D}} + T_o \end{aligned} \tag{2.18}$$

The result shows that the overhead-independent part of the latency can roughly improve proportionally with the square root of the number of disks D. For each D,

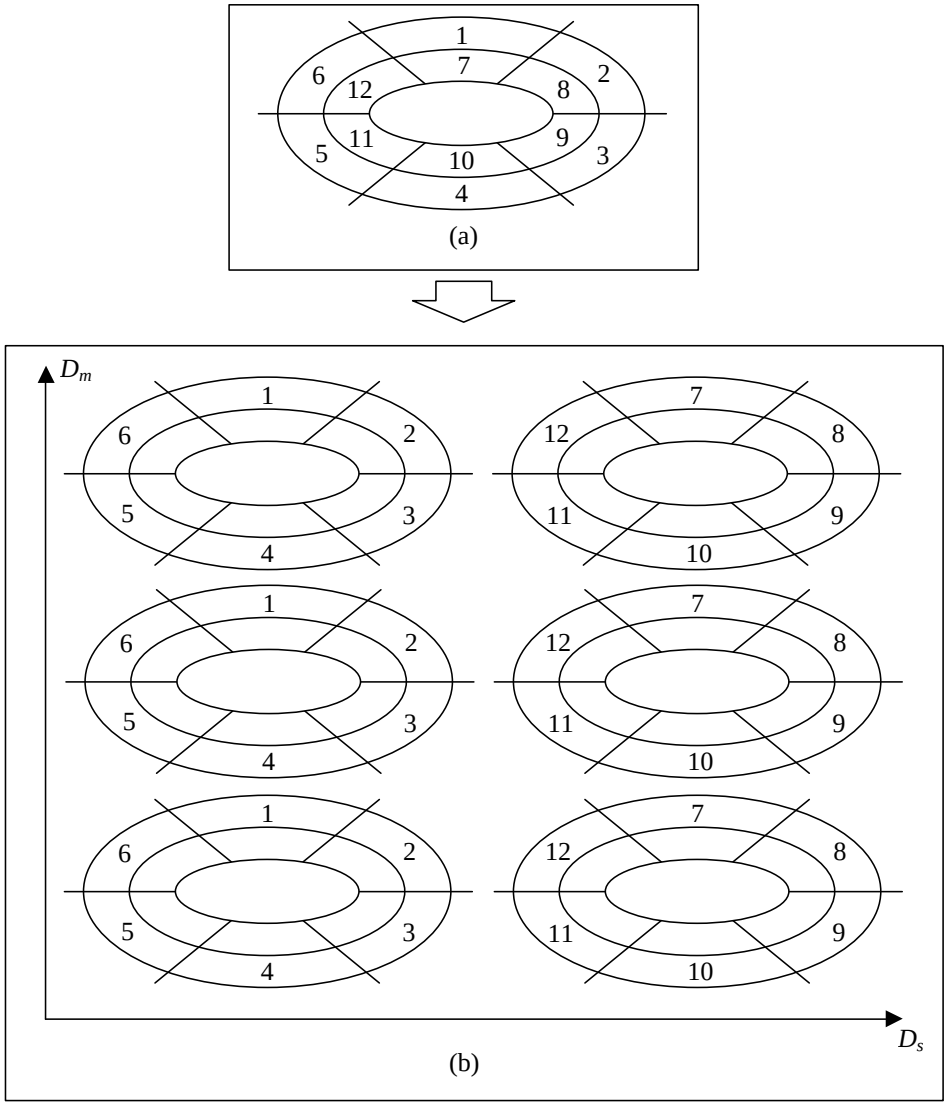


Figure 2.8: An illustration of a $D_s \times D_m$-way striped-mirroring disk array: (a) the original disk with 12 sectors; (b) a striped-mirroring 6-disk array with 2-way striping($D_s = 2$) and the resulting two striped disks replicated three times($D_m = 3$).

there can be several combinations of D_s and D_m value whose product is D. This equation gives no insight on latency differences among different D_s and D_m values, however. We depend on experimental results to study the performance differences among different combinations.

2.3.2 SR-Array: Combining Striping and Rotational Replication

Because striping and rotational replication each reduces one term of the latency equation, we can combine both of them to reduce both terms. Figure 2.9 shows an example of this combination, which we call an SR-Array. Let us consider using a total of D disk drives to hold a single disk's worth of data. We decide to dedicate D_s disks to the dimension of striping to reduce the seek time to $S/3D_s$ and D_r disks to the dimension of rotational replication to reduce the rotational delay to $R/2D_r$. When $D_r = 1$, an SR-Array degenerates to simple D-way striping. When $D_s = 1$, it degenerates to simple D-way rotational replication. In Figure 2.9, $D_s = 2$ and $D_r = 3$.

The advantage of an SR-Array lies in not only reducing both seek time and rotational delay but also the fact that balanced reduction can be achieved in both terms. When S is less than R, more effort should be devoted to reducing R and vice versa. Another advantage is that both of the techniques are amenable to relatively simple analytical models and we can derive a good model to help answer the following question: given a fixed budget of D disks, what degree of striping and what degree of rotational replication should we use to achieve optimal average latency for a given workload? Figures 2.10(b) and (c) demonstrate this "aspect ratio" question. We first answer the question for read latency and then consider a workload with a mixture of reads and writes.

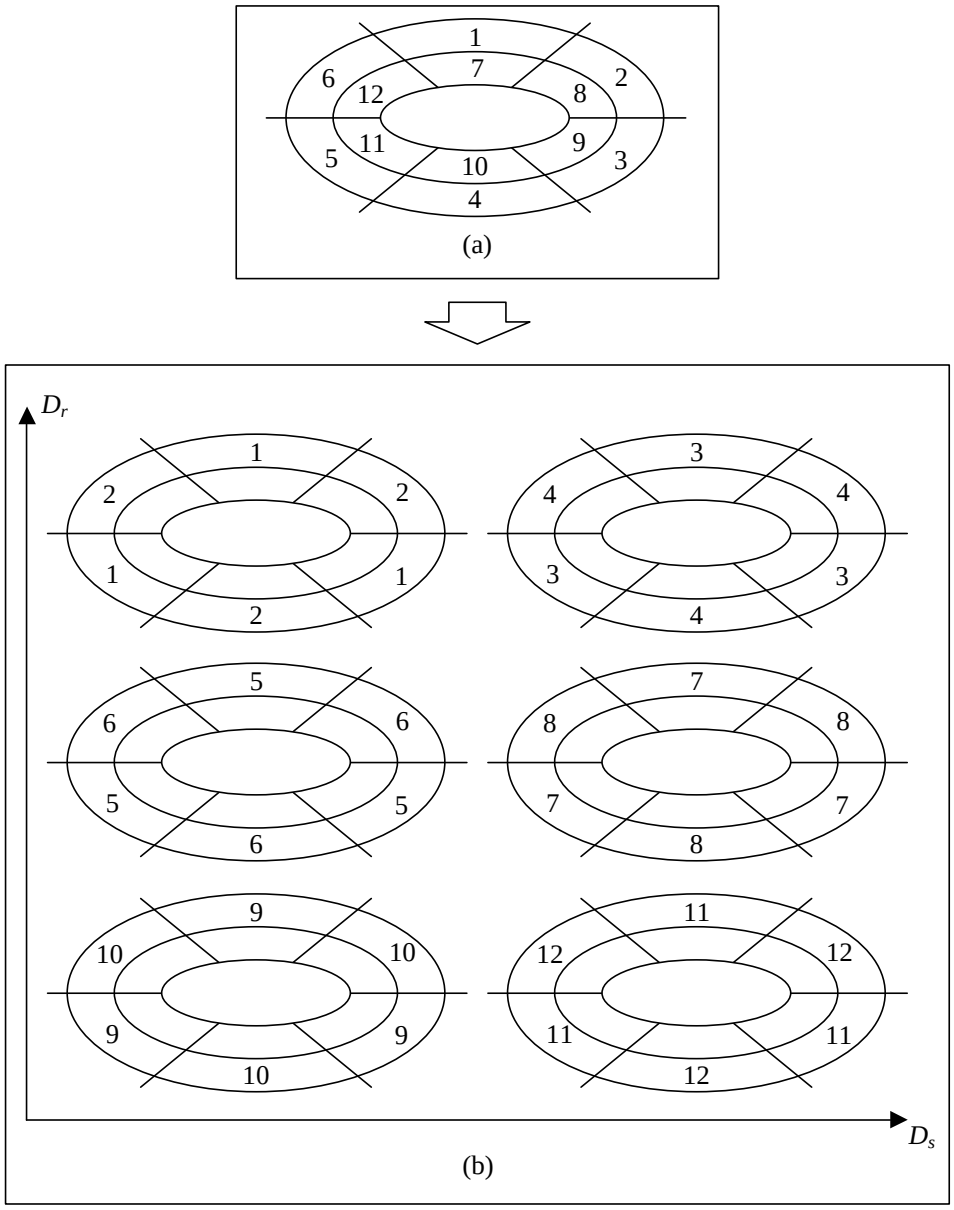


Figure 2.9: An illustration of a $D_s \times D_r$-way SR-Array: (a) the original disk with 12 sectors; (b) a 6-disk SR-Array with 2-way striping($D_s = 2$) and each sector in the resulting two-way striped disks is rotationally replicated three times($D_r = 3$).

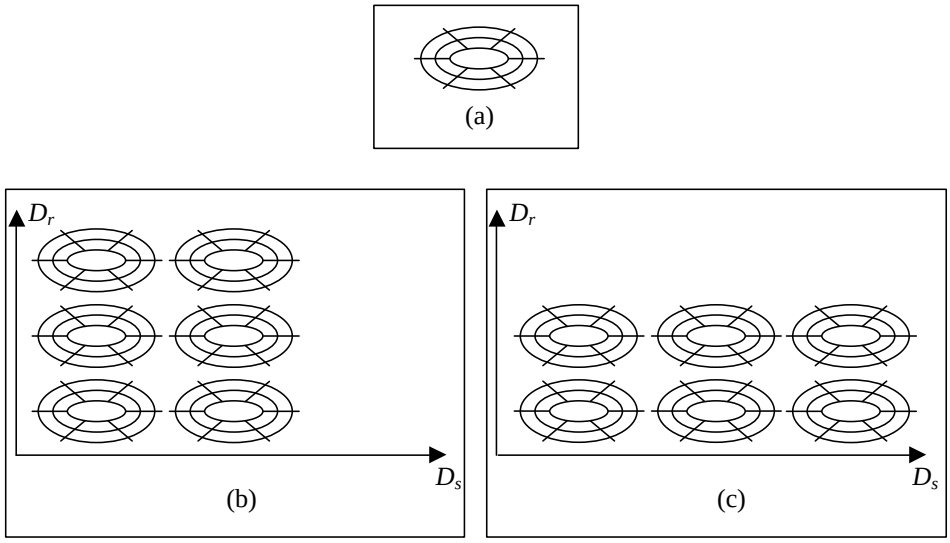


Figure 2.10: An illustration of the SR-Array aspect ratio question: (a) the original disk; (b) a 2×3 SR-Array with 6 disk drives; (c) a 3×2 SR-Array also with 6 disk drives.

Read Latency on SR-Array

The average random read latency $T_R(D_s, D_r)$ is the sum of the reduced seek time, the reduced rotational delay and the fixed overhead.

$$T_R(D_s, D_r) = \frac{S}{3D_s} + \frac{R}{2D_r} + T_o \tag{2.19}$$

We know that $a + b \leq 2\sqrt{ab}$ for any $a \geq 0$ and $b \geq 0$. Given the constraint of $D_s D_r = D$, we can prove that the following configuration reduces the seek time to T_{best_seek} and the rotational delay to T_{best_rot} and produces the best overall latency T_{best}.

$$\begin{cases} D_s = \sqrt{\frac{2S}{3R}D} \\ D_r = \sqrt{\frac{3R}{2S}D} \end{cases} \tag{2.20}$$

$$T_{best\_seek} = \sqrt{\frac{SR}{6D}} \tag{2.21}$$

$$T_{best\_rot} = \sqrt{\frac{SR}{6D}} \tag{2.22}$$

$$T_{best} = \sqrt{\frac{2SR}{3D}} + T_o \tag{2.23}$$

This result shows that in order to achieve the best reduction of overall latency, D_s and D_r have to be chosen in such a way that the reduced average seek time and the reduced average rotational delay are equal.

According to the above results, disks with slower rotational speed (larger R) demand a higher degree of rotational replication. In terms of the SR-Array illustration of Figure 2.9(b), this argues for a tall thin grid like Figure 2.10(b). On the other hand, disks with poorer seek characteristics (larger S) demand a larger striping factor. In terms of Figure 2.9(b), this argues for a short fat grid like Figure 2.10(c). To capture seek locality, we replace S with S/L, where L is the seek locality index used in equation (2.5). More seek locality leads to a larger L and also argues for larger optimal D_r and a tall thin grid.

The model indicates that the latency improvement on an SR-Array is proportional to the square root of the number of disks ($\sqrt{D}$). It is likely that the optimal D_s and D_r are not integer values. For such scenarios, we choose the closest integer values that could produce the best result in practice.

Read/Write Latency on SR-Array

Now we extend the latency model of an SR-Array to model the performance of both read and write operations. Naively, when we write a sector with multiple copies, we write them all consecutively in one rotation before we complete the write request. A potentially better technique is to complete writing the closest copy first and delay the

propagation of other copies to a future time when the disk is idle. The next chapter covers delayed write propagation in greater details.

When there is indeed idle time to write the replicas in the background, the latency of a write can be treated as that of a read. In the scenario of not being able to mask the cost of replica propagation in idle time, we must incur a write latency of $T_W(D_s, D_r)$.

$$T_W(D_s, D_r) = \frac{S}{3D_s} + R - \frac{R}{2D_r} + T_o \tag{2.24}$$

Let the number of reads be X_r, the number of writes that can be propagated in the background be X_{wb}, and the number of remaining writes that are propagated in the foreground be X_{wf}. We define the ratio p to be the fraction of requests that incur the latency of an average read request.

$$p = \frac{X_r + X_{wb}}{X_r + X_{wb} + X_{wf}} \tag{2.25}$$

Now, the average read/write latency can be modeled as $T(D_s, D_r)$.

$$\begin{aligned} T(D_s, D_r) &= pT_R + (1-p)T_W \\ &= \frac{S}{3D_s} + p\frac{R}{2D_r} + (1-p)(R - \frac{R}{2D_r}) + T_o \end{aligned} \tag{2.26}$$

The first term is the average seek incurred by any request. The second term is the average rotational delay consumed by I/O operations that do not result in foreground replica propagation (based on Equation (2.8)) with probability p; and the third term is the rotational delay consumed by writes whose replicas are propagated in the foreground due to lack of idle time (based on Equation (2.9)) with probability $1 - p$. We derive that the following configuration provides the best overall latency

T_{best}.

$$\begin{cases} D_s = \sqrt{\dfrac{2S}{3R(2p-1)}}D \\ D_r = \sqrt{\dfrac{3R(2p-1)}{2S}}D \end{cases} \tag{2.27}$$

$$T_{best} = \sqrt{\frac{2SR(2p-1)}{3D}} + (1-p)R + T_o \tag{2.28}$$

A low p ratio here calls for a short fat grid in Figure 2.9(b). A p ratio under 50% precludes rotational replication and pure striping provides the best configuration as discussed in Section 2.2.3. In the best case, when all write replicas can be propagated in the background (or when we have no writes at all), writes and reads become indistinguishable as far as this model is concerned, so $p = 1$ and the latency improvement is again proportional to $\sqrt{D}$.

2.3.3 Performance Improvement for Sequential Access

Although the disk array layout schemes of Figure 2.9(b) and the earlier Figure 2.6(b) improve the average latency of a random read request by using D_r rotational replicas to reduce the average rotational delay to $1/D_r$ of the original average, they also reduce the bandwidth available to a sequential request because the disk has to incur a track switch every $1/D_r$ of a full rotation instead of every full rotation as in the Figure 2.6(a) layout. As illustrated in Figure 2.3, a track switch is a little below $1ms$, which is about 1/6 of the full rotation time of the disk drives used in our study. Therefore, even a modest $D_r = 2$ in the Figure 2.6(b) layout can result in substantial performance penalty for sequential access.

In order to avoid this substantial performance penalty for sequential access, we introduce an alternative layout scheme for rotational replication, where we replicate

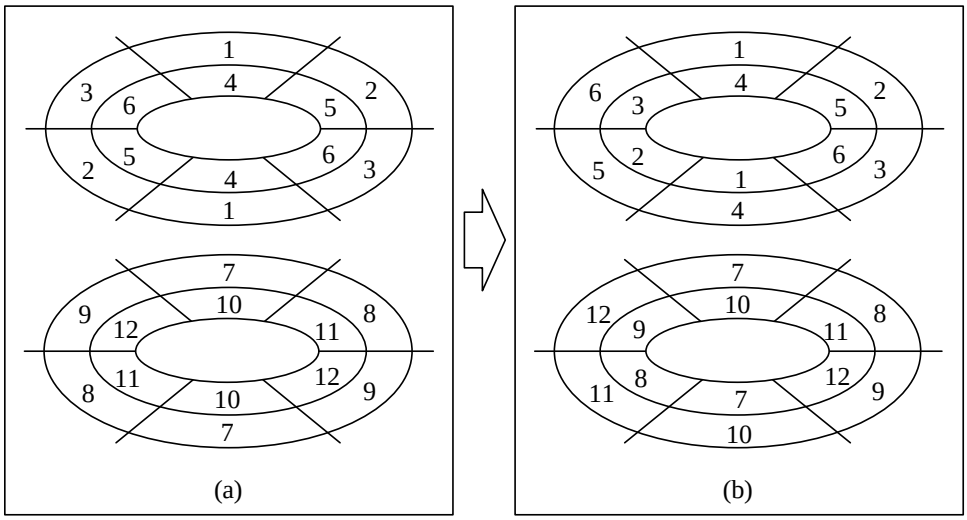


Figure 2.11: An illustration of 2-way rotational replication for sequential access for the 12-sector disk show in Figure 2.6(a): (a) the data is spread, as in Figure 2.6(b), to two disks, each of which holds 6 sectors and rotationally has 2 copies of each sector spread evenly in the same track; (b) the data is spread to two disks, each of which holds one track of 6 sectors and has 2 copies of the same track positioned next to each other, but with replicas of data sectors placed at evenly spaced rotational angles.

and rotate a full track into adjacent tracks as shown in Figure 2.11(b). We also include the original layout scheme of Figure 2.6(b) next to Figure 2.11(b) for comparison. We can also apply the same technique to our SR-Array layout illustrated in Figure 2.9(b). In this new layout scheme, the disk incurs a track switch every full rotation, the same overhead as that in the original disk without replication. When a track switch happens, instead of switching the head to an adjacent track in the Figure 2.11(a) case, the disk head skips the adjacent $D_r - 1$ adjacent replicated tracks and switches to a track that contains the next distinct set of data. With a modest D_r, this switch costs almost the same as one would when switching to an adjacent track.

Because the replicated tracks are placed adjacent to each other with an evenly spaced rotational angle, the seek time to these tracks are almost the same. The rotational delay for a random read request can also be the same if the disk drive can predicate and pick beforehand the correct track that has the replica with the

shortest delay. Our experimental platform described in the next chapter shows that we can indeed predict quite accurately even in the host system. For example, our predications are 99.8% accurate in the experiments of Section 3.5.2.

For replica writes of a small write request, we can switch to the adjacent replicated track as soon as we finish writing one replica in a track. As long as the track switching time is less than the minimum write time between the adjacent tracks, the new layout scheme can finish writing all the replicas in one full rotation without extra penalty time when compared with the layout scheme that puts all the replicas in the same track. In our experimental platform, we can switch track and write the next replica in time when $D_r \leq 6$.

In summary, when D_r is modest, the two layout schemes shown in Figure 2.11(a) and (b) result in the same seek latency and rotational delay for random read and write requests while the layout scheme in Figure 2.11(b) also enjoys the same performance for sequential requests as that of a traditional disk layout schema. Therefore, all our mathematical results on random requests in this chapter apply to both layout schemes. Our experimental platform uses the improved layout scheme for SR-Array.

2.3.4 Mirroring Extension of SR-Array

By combining striping and rotational replication, we arrive at an SR-Array configuration family that can reduce latency in a balanced manner. Because each piece of data only exists in one disk drive in an SR-Array, the reliability of the array is not as good as mirroring.

By adding mirroring to replicate each disk drive in an SR-Array D_m times, we can achieve better reliability. We call the result of this SR-Array extension as an "SR-Mirror". The SR-Mirror configuration family has 3 parameters: D_s, D_r and

D_m, where D_s implies that only $1/D_s$ of the space of the disk is used to hold any data (to reduce seek time), D_r is the number of replicas on the same disk, and D_m is the number of replicas on different disks. The total disks needed for an SR-Mirror configuration is $D = D_s D_r D_m$. A $D \times 1 \times 1$ system is D-way striping. A $1 \times 1 \times D$ system is a D-way mirror. A $D_s \times D_r \times 1$ system is an SR-Array. A $D_s \times 1 \times 2$ is the commonly used RAID-10 configuration.

From the read latency model for an SR-Array, we can see that an SR-Array behaves like a single disk with a leaner (the maximum seek distance reduced to $1/D_s$) and faster rotating disk (the full rotation time reduced to $1/D_r$). This way, we may approximate the performance of an SR-Mirror by replacing S with S/D_s and R with R/D_r in the mirroring models. As a corollary of equation (2.16), we obtain the average read latency $T_{SRM}(D_s, D_r, D_m)$, when S/D_s and R/D_r are close.

$$\begin{aligned} T_{SRM}(D_s, D_r, D_m) &= \sqrt{\pi \frac{\frac{R}{D_r}\frac{S}{Ds}}{4D_m}} + T_o \\ &= \sqrt{\frac{\pi RS}{4D}} + T_o \end{aligned} \tag{2.29}$$

While this equation does not shed light on how to choose the D_s, D_r, and D_m values, our intuition is to choose $D_m > 1$ only when a higher degree of reliability is desired. A D_m value that is larger than that required by reliability is undesirable because mirroring appears to deliver poorer write performance than rotational replication on a single disk (as discussed in Section 2.2.4). Unfortunately, we do not have a closed form model for the write latency of mirroring so that we cannot currently precisely quantify analytically the performance difference.

This thesis mainly studies the properties of the SR-Array both in mathematical models and in experiments. We will compare the SR-Array ($D_s \times D_r \times 1$) with

other configurations of the SR-Mirror, such as simple striping ($D_s \times 1 \times 1$), RAID-10 ($D_s \times 1 \times 2$) and striped-mirroring ($D_s \times 1 \times D_m$).

2.4 SR-Array Throughput Modeling

While the average latency of random requests is an important metric of a storage system, overall sustained throughput is a more useful metric to measure the system's ability of handling a heavy load of requests. In this section, we discuss some of the results we have derived for modeling throughput of different disk array configurations. Our discussion mainly focuses on striping, rotational replication, and the combination of the two, SR-Array. Because pure striping and pure rotational replication are degenerated cases of SR-Array, we only give results for the SR-Array throughput model and discuss the degenerate cases when needed. As is the case with latency, modeling configurations involving mirroring is not as easy as modeling SR-Array if we consider rotational position of the heads when scheduling the requests. We discuss these configurations in empirical studies later.

When a storage system handles a heavy load, requests are queued. Outstanding requests can sometimes be reordered. This may reduce the seek time and rotational delay between consecutive requests so that we can achieve low response time and high throughput. Various techniques for reordering or scheduling a queue of requests lead to different request throughputs. The throughput of a technique can be calculated as the inverse of average request time.

The more the number of requests in the queue, the more choices a policy has to schedule the next request and therefore the lower the average latency and the higher the throughput. An effective scheduling policy should achieve lower average latency with more requests in the queue. In studying the scheduling policies on SR-Array,

we consider a scenario that always keeps a constant Q outstanding requests. As one request is satisfied in this scenario, another new request is generated in time to keep a constant Q outstanding requests for the next scheduling decision. This way we may understand better how the queue length affects the overall throughput of a scheduling policy.

In this section, we discuss mathematical models of throughput for some scheduling policies for SR-Array. When calculating throughput, we first calculate the average latency between the completion of two consecutive requests for a scheduling policy. We then use the average latency to compute the aggregate throughput produced by the disk array.

2.4.1 FCFS Scheduling

The naive scheduling policy FCFS serves the requests in the order of their arrival. In this case, every request in this scheduling policy costs the average latency time given in the SR-Array latency model in Equation (2.26) and the optimal latency in Equation (2.28).

Even though Q does not affect the single request latency, it does affect the probability of some disks in a disk array being idle during a particular period. A greater Q reduces the chance of a disk being idle in a given period and therefore increases the disk array utilization. Equation (2.45) in Section 2.4.3 reveals how we consider the individual disk utilization factor in disk array situations and build disk array throughput models based on single disk throughput models.

2.4.2 LOOK Scheduling for One Disk

The LOOK scheduling policy attempts to reorder the requests according to their cylinder order to reduce disk arm movement [26, 43]. In LOOK scheduling, a disk arm always moves in the same direction until there is no more request in the current direction and there is at least one request in the opposite direction. In that case, the disk arm changes direction and starts to sweep across the platters in the opposite direction. The LOOK scheduling policy amortizes at most one full seek distance over all the consecutive requests served between two consecutive arm direction changes.

Both LOOK and its similar alternative, SCAN, have been well studied for random workloads with Poisson arrival time [8, 7, 33, 44]. Here, we analyze LOOK with a fix number of outstanding requests in the disk queue. Our own analysis of LOOK scheduling algorithm on a single disk lays the foundation of our analysis of a similar scheduling algorithm RLOOK on SR-Array in the next sub-section.

Expected Seek Distance of One Sweep of q Requests

In the following analysis, we use q to represent the number of requests scheduled in a single sweep during which the head moves in the same seek direction. We assume that the sweep starts at a random request that has just finished and has a lower cylinder number than all of the other q requests. Obviously, the head moves up the cylinder numbers from the starting request and serves all the q requests. The head stops at the request that has the maximum cylinder number among the q requests. We treat the seek positions of these $q+1$ requests as evenly distributed random numbers between 0 and the maximum seek distance S.

We first calculate the expectation (S_{min}) of the minimum of these $q+1$ random numbers ($a_0, a_1, ...a_q$) between 0 and S. We calculate the sum of every minimum seek

position x multiplied by its occurrences $(q+1)(S-x)^q$ and divide it by the sum of all the possible occurrences.

$$\begin{aligned} S_{min} &= \frac{\int_0^S x(q+1)(S-x)^q\,dx}{\int_0^S (q+1)(S-x)^q\,dx} \\ &= \frac{(q+1)\int_0^S (S-x)\,x^q dx}{(q+1)\int_0^S x^q dx} \\ &= \frac{(q+1)\left(\frac{1}{q+1}-\frac{1}{q+2}\right)S^{q+2}}{S^{q+1}} \\ &= \frac{S}{q+2} \end{aligned} \tag{2.30}$$

We also realize that $S-a_0, S-a_1, ..., S-a_q$ are also $q+1$ evenly distributed random values between 0 and S. Their expected minimum is S_{min} based on the above analysis. Obviously, for $S-a_i$ to be the minimum value among $S-a_0, S-a_1, ..., S-a_q$, a_i must be the maximum value among $a_0, a_1, ...a_q$. Therefore, we get S_{max}.

$$\begin{aligned} S_{max} &= S - S_{min} \\ &= \frac{q+1}{q+2}S \end{aligned} \tag{2.31}$$

Therefore, the average seek distance S_{LOOK} of one sweep is the seek distance between the minimum and the maximum positions of the $q+1$ random positions. The latency of one request T_{1LOOK} on one disk and $T_{LOOK}(Ds, Dr)$ on a $D_s \times D_r$ SR-Array can be derived from S_{LOOK}.

$$S_{LOOK} = S_{max} - S_{min}$$

$$= \frac{q}{q+2}S \tag{2.32}$$

$$T_{1LOOK} = \frac{S_{LOOK}}{q} + \frac{R}{2} + T_o$$

$$= \frac{S}{q+2} + \frac{R}{2} + T_o \tag{2.33}$$

$$T_{LOOK}(D_s, D_r) = \frac{S}{(q+2)D_s} + \frac{R}{2D_r} + T_o \tag{2.34}$$

When $q = 1$, the above result is consistent with the expected single request latency for one disk or an SR-Array discussed in the previous sections.

Expected Seek Distance among Random Requests

As a preparation for later analysis, we prove here that the expected position of the k^{th} minimum among the $q+2$ random positions is $\frac{kS}{q+2}$. We prove this by mathematical induction. We have proved the $k = 1$ case above. Assuming that the $k = i$ case is true and the expected position of the i^{th} minimum is $\frac{iS}{q+2}$, we see that the rest of the $q + 1 - i$ positions that are greater than the i^{th} minimum should be evenly and randomly distributed between the i^{th} minimum and S. Obviously, the $i+1^{th}$ minimum position must be the minimum position of these $q + 1 - i$ positions. Therefore, the expected difference *diff*$(i, i+1)$ between the i^{th} and $i+1^{th}$ minimum should be $\frac{1}{(q+2-i)}$ of the distance between the i^{th} minimum and S.

$$\textit{diff}(i, i+1) = \frac{1}{(q+2-i)}\left(S - \frac{iS}{q+2}\right)$$

$$= \frac{S}{q+2}$$

Therefore, the expected position of $i + 1^{th}$ minimum is $\frac{(i+1)S}{q+2}$ and the case is also true for $k = i + 1$. The proof ends here. As a corollary, the expected distance between the

i^{th} minimum and k^{th} minimum for these $q+1$ numbers is the following, given $k > i$.

$$diff(i,k) = \frac{(k-i)S}{q+2} \tag{2.35}$$

Expected Number of Requests in One Sweep for $Q = 1$

Now, let us examine the scenario where we always keep Q outstanding requests. We can not simply use the result of Equation 2.30 because having Q constant outstanding requests does not mean that every arm sweep serves exactly Q requests. In fact, after we finish one request and the new request generated falls at a location that the arm is currently moving towards, we should also be able to serve that request in the same sweep. Therefore, the expected number of requests in one sweep is a function of Q and is larger than Q.

We define a *forward seek* as a seek from a lower cylinder number to a higher one and a *backward seek* as one that the arm moves in the opposite direction. We also define a *forward sweep* as $n \geq 1$ contiguous forward seeks sandwiched between two backward seeks. Similarly, we define a *backward sweep* as a group of contiguous backward seeks sandwiched between two forward seeks. This way if we number all the sweeps from 1 and start with a forward sweep, all forward sweeps are odd-numbered and all backward sweeps are even-numbered.

Because of the symmetry of forward and backward sweeps, we only need to calculate the expected number of requests in a forward sweep. Note that we assume here that the disk has an infinitely large number of cylinders. Therefore, the probability that two requests in Q requests share n cylinder number is asymptotically zero and we only need to consider different cylinder numbers as an approximation. In the following analysis, we use fractions of the maximum seek distance to represent all the seek positions. Therefore, the "weighted" averages calculated are also in the fraction

of the maximum seek distance.

We begin our calculation by first looking at the $Q = 1$ case and calculating (1) the expected number of requests in one forward sweep; (2) the expected aggregate seek distance during one forward sweep; (3) the expected seek distance per request as the ratio between the above two quantities. We then explain the relationship between the $Q = 1$ case and the $Q \geq 1$ cases and use the result of the $Q = 1$ case to obtain the results for all the $Q \geq 1$ cases.

When $Q = 1$, each time we only have one request in the queue to schedule. A forward sweep starts with the last request of the previous backward sweep. Let a_0 be the seek position of the last request before the last backward seek; a_1 be that of the first request before the first forward seek; a_n be that of the last request before the last forward seek; a_{n+1} be that of the first request before the first backward seek after the whole forward sweep; and a_{n+2} be the resulting seek position of the backward seek performed after a_{n+1}. Expressed mathematically, a sequence that meets this condition must satisfy $a_0 > a_1 < a_2 < ... < a_n < a_{n+1} > a_{n+2}$, which has $n \geq 1$ forward seeks to form a forward sweep.

Notice that this type of sequence must start with the first three requests in the order of $a_0 > a_1 < a_2$, which defines the end of a previous backward sweep and the start of the current forward sweep. Among all $3! = 6$ possible orderings of three independently generated random requests, only two($a_1 < a_0 < a_2$ and $a_1 < a_2 < a_0$) satisfy the condition of $a_0 > a_1 < a_2$. Therefore, at any instance, we have 1/3 probability that a seek from a_0 to a_1 is the end of a backward sweep and the seek from a_1 to a_2 is the beginning of a forward sweep.

Among the 1/3 chances of starting a forward sweep after a backward sweep, the length of the forward sweep could be from 1 to infinity. A forward sweep with length n must have $n + 3$ requests starting from a_0 to satisfy $a_0 > a_1 < a_2 < ... < a_n <$

$a_{n+1} > a_{n+2}$. There are a total of $(n+3)!$ possible ascending-ordered sequences for the $n+3$ requests. Among these $(n+3)!$ possibilities, we count and place those that satisfy the above ordering requirement into the categories enumerated below. For each category, we also calculate the expected forward sweep distance $a_{n+1} - a_1$ (as fractions of the maximum seek distance) by using the result of Equation (2.35).

1. $(n+1)n$ ascending-ordered sequences that have a_{n+1} as the maximum and a_1 as the minimum. There are $(n+1)n$ ways to pick a_0 and a_{n+2}. The sweep distance is $\frac{n+3}{n+4} - \frac{1}{n+4} = \frac{n+2}{n+4}$.

2. n ascending-ordered sequences that have a_{n+1} as the maximum, a_{n+2} as the minimum and a_1 as the second minimum. There are n ways to pick a_0. The sweep distance in this case is $\frac{n+1}{n+4}$.

3. n ascending-ordered sequences that have a_0 as the maximum, a_{n+1} as the second maximum, and a_1 as the minimum. There are n ways to pick a_{n+2}. The sweep distance in this case is also $\frac{n+1}{n+4}$.

4. 1 ascending-ordered sequence that has a_0 as the maximum, a_{n+1} as the second maximum, a_{n+2} as the minimum and a_1 as the second minimum. The sweep distance in this case is $\frac{n}{n+4}$.

Now we weigh the forward sweep of length n with its probability (the sum of all the above four categories) and sum cases for all possible values of n and obtain the expected number of requests for each forward sweep $q(1)$. The probability for a forward sweep to happen is 1/3.

$$
\begin{aligned}
q(1) &= \frac{\sum_{n=1}^{\infty}\left(n \cdot \frac{(n+1)n+n+n+1}{(n+3)!}\right)}{\frac{1}{3}} \\
&= 3\sum_{n=1}^{\infty}\left(\frac{1}{n!} - \frac{3}{(n+1)!} + \frac{5}{(n+2)!} - \frac{3}{(n+3)!}\right) \\
&= 3\left(\frac{1}{1!} + \frac{1-3}{2!} + \frac{1-3+5}{3!}\right) \\
&= \frac{3}{2} \qquad (2.36)
\end{aligned}
$$

$$(2.37)$$

Expected Single Request Latency in One Sweep for $Q = 1$

We also obtain the expected sweep distance for all forward sweeps as $dist(1)$ and the expected single request seek distance $T_S(1)$ as the ratio between $q(1)$ and $dist(1)$. Both $dist(1)$ and $T_S(1)$ are calculated as fractions of the maximum seek distance.

$$
\begin{aligned}
dist(1) &= \frac{\sum_{n=1}^{\infty}\left(\frac{1}{(n+3)!}\left(\frac{n+2}{n+4}(n+1)n + \frac{n+1}{n+4}n + \frac{n+1}{n+4}n + \frac{n}{n+4}1\right)\right)}{\frac{1}{3}} \\
&= 3\sum_{n=1}^{\infty}\left(\frac{1}{(n+1)!} - \frac{4}{(n+2)!} + \frac{7}{(n+3)!} - \frac{4}{(n+4)!}\right) \\
&= 3\left(\frac{1}{2!} + \frac{1-4}{3!} + \frac{1-4+7}{4!}\right) \\
&= \frac{1}{2} \qquad (2.38) \\
T_S(1) &= \frac{dist(1)}{q(1)} \\
&= \frac{1}{3} \qquad (2.39)
\end{aligned}
$$

Note that our average seek distance computed here for $Q = 1$ is exactly the same

as the expected latency for evenly distributed random requests in an early section.

Expected Number of Requests in One Sweep for $Q \geq 1$

Earlier in the $Q = 1$ case, we have divided the request sequence (ordered by the time of request generation) into intermingled forward sweeps and backward sweeps. We now do the same for the $Q \geq 1$ cases. We have proved that when $Q = 1$, the expected number of requests for a sweep is 1.5. We now prove that the expected number of requests for a sweep in the $Q \geq 1$ case is $1.5Q$.

Given Q outstanding requests, we make another request sequence $a_0, a_1, ..., a_{Q-1}$, $a_Q, a_{Q+1}, ..., a_{2Q}$ in an order such that (1) $a_0, ..., a_{Q-1}$ are the first Q random requests generated; and (2) the fulfillment of request a_i generates request a_{i+Q} for all $i \geq 1$. Note that this request sequence is neither necessarily ordered by request generation time, nor necessarily ordered by the request fulfillment time. Instead, we use this sequence to capture the inter-dependency among requests.

By the definition of the sequence, request a_{i+Q} must be served after request a_i but before request a_{i+2Q}. Also, if request a_i falls into a forward sweep, request a_{i+Q} falls into the same forward sweep if and only if $a_{i+Q} > a_i$. Otherwise, it falls into the next backward sweep. Similar reasoning holds if request a_i falls into a backward sweep. This way the request sub-sequence $a_0, a_Q, a_{2Q}, ...a_{nQ}, ...$ behaves exactly as an average $Q = 1$ case, as if they were generated in the $Q = 1$ case. Therefore, an average of 1.5 requests in this sub-sequence fall into a sweep. Because there are Q such independent sub-sequences, the expected number of requests for a sweep is $1.5Q$, which holds true for all $Q \geq 1$ cases.

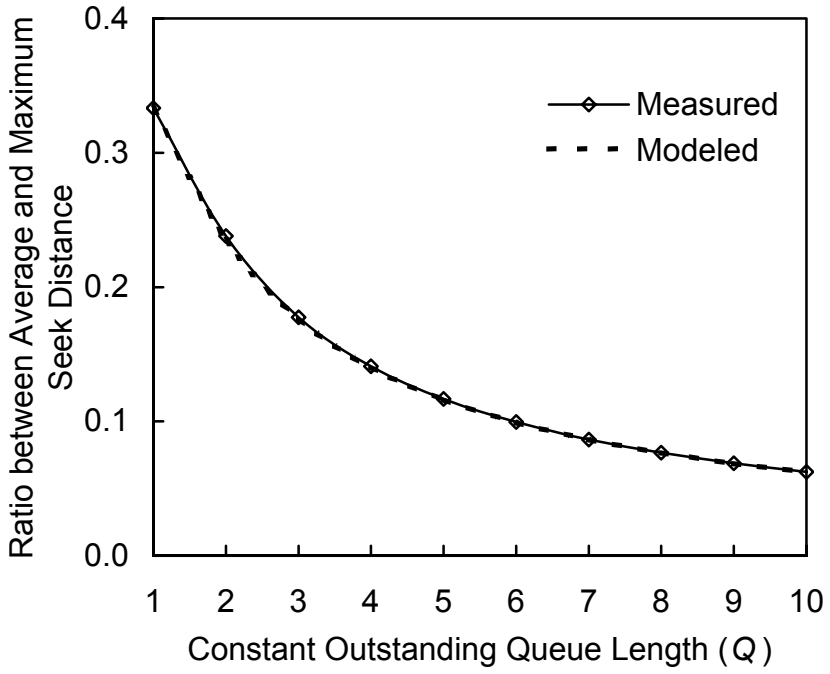


Figure 2.12: Evaluation of the expected single request seek latency using the LOOK scheduling policy: (1) The "Measured" curve represents the experiment result of keeping Q constant outstanding requests; (2) The "Modeled" curve represents Equation (2.40).

Expected Single Request Latency in One Sweep for $Q \geq 1$

Though we can leverage the results of the $Q = 1$ case to calculate the expected single request latency in one sweep for the $Q \geq 1$ cases, we need to modify the $Q = 1$ equation for the expected one sweep distance because the sweep distance is from the minimum of $Q \geq 1$ minimums to the maximum of Q maximums in every sub-sequence. For large Q, the minimum is close to 0 and the maximum is close to S.

Though we can not show a closed-form result of the average seek distance, we can still get a good approximation $T_S(Q)$ for all $Q \geq 1$ cases. Figure 2.12 shows that this seek time model is a good approximation (within 1%) to the result measured from an experiment. We can also get average latency $T_{1LOOK}(Q)$. The expected throughput

for one disk is the inverse of the expected single request latency.

$$T_S(Q) = \frac{S}{1.5Q + 1 + \frac{0.5}{Q}} \tag{2.40}$$

$$T_{1LOOK}(Q) = \frac{S}{1.5Q + 1 + \frac{0.5}{Q}} + \frac{R}{2} + T_o \tag{2.41}$$

2.4.3 RLOOK Scheduling for SR-Array

We have developed an extension of the LOOK algorithm for SR-Arrays which we call RLOOK. Under the traditional LOOK algorithm, the disk arm moves bi-directionally from one end of the disk to the other, servicing requests that are in the cylinders under the path of the arm movement. On an SR-Array disk, in addition to scanning the disk in the seek direction, our RLOOK scheduling also chooses the replica that is rotationally closest among all the replicas during the sweep.

Expected Single Request Latency

When we have a $D_s \times D_r$ SR-Array, the total of Q requests are distributed to D disks. When Q is sufficiently large, the distribution is close to an even distribution and each disk is likely to maintain a queue length close to $q = Q/D$. Expected single request latency $T_{1LOOK}(D_s, D_r)$ can be derived from Equation (2.41) and (2.26) under the same workload assumption.

$$\begin{aligned} T_{1LOOK}(D_s, D_r) &= pT_R + (1-p)T_W \\ &= \frac{S}{\left(1.5q + 1 + \frac{0.5}{q}\right) D_s} + \frac{pR}{2D_r} + (1-p)(R - \frac{R}{2D_r}) + T_o \end{aligned} \tag{2.42}$$

We can get the optimal configuration for the workload and the optimal expected single request latency as below.

$$\begin{cases} D_s = \sqrt{\dfrac{2S}{\left(1.5q + 1 + \dfrac{0.5}{q}\right) R\,(2p-1)}}D \\ \\ D_r = \sqrt{\dfrac{\left(1.5q + 1 + \dfrac{0.5}{q}\right) R(2p-1)}{2S}}D \end{cases} \tag{2.43}$$

$$\begin{aligned} T_{1LOOK\_best} &= \sqrt{\frac{2SR(2p-1)}{\left(1.5q + 1 + \dfrac{0.5}{q}\right) D}} + (1-p)R + T_o \\ &= \sqrt{\frac{2SR(2p-1)}{1.5Q + D + \dfrac{0.5D^2}{Q}}} + (1-p)R + T_o \end{aligned} \tag{2.44}$$

When Q/D is sufficiently large, $1.5Q + D + \frac{0.5D^2}{Q}$ in T_{1LOOK_best} is relatively close to $1.5Q$, which means that the asymptotic optimal single request latency T_{1LOOK_best} is independent of the number of disks in the SR-Array. Intuitively, in an SR-Array with a fixed configuration and a fixed queue length, as we increase the number of disks, there are fewer requests per disk and therefore the amortized seek distance between two consecutive requests is longer. At the same time, the optimal SR-Array configuration demands a greater D_s to be devoted to seek reduction. The net result of these two opposite effects is a nearly constant single request latency.

Expected Throughput

Having modeled the throughput of a single disk in an SR-Array, we now attempt to model the throughput of an entire SR-Array with D disks and a total of $Q = Dq$ outstanding requests, where q is the average queue length per disk. Because we assume that the requests are randomly distributed in the system, there could be a load imbalance in the form of idle disks especially when Q is not much greater than D. The probability of one disk being idle is $\left(1 - \frac{1}{D}\right)^Q$. Therefore, the total throughput of the system can be approximated as:

$$N_D \approx D\left[1 - \left(1 - \frac{1}{D}\right)^Q\right] \cdot \frac{1}{T_{1LOOK\_best}} \tag{2.45}$$

When Q is much larger than D, $\left(1 - \frac{1}{D}\right)^Q$ is close to zero and the chance of a disk being idle becomes very small. As all the disks face similar load in this case, the total throughput in an optimal SR-Array configuration is approximately proportional to the number of disks D.

Although this approximation is derived only based on the reasoning about the presence of one idle disk, we shall see in Section 4.1.2 in our empirical study that it is in fact a good approximation to the real measured throughput. Those empirical results also show that the throughput of pure striping or rotational replication is sub-linear and worse than that of an optimal SR-Array for a given number of disks.

2.4.4 SATF and RSATF Scheduling

The scheduling policy called Shortest-Access-Time-First(SATF) attempts to pick the next request in the queue with the least sum of seek time and rotation delay from the head position of the current request. Gum[16] shows that the average single request

latency T_{1SATF} is a good approximation for SATF scheduling, if the full seek time S is close to the full rotation time R.

$$T_{1SATF} = \sqrt{\frac{\pi RS}{4q}} + T_o \tag{2.46}$$

Since we have described the RLOOK extension to LOOK, it is easy to understand a similar extension to SATF: RSATF for SR-Array. An RSATF scheduler chooses the next request with the shortest access time by considering all rotational replicas. It is well known that SATF outperforms LOOK [26, 43] by considering rotational delay. We do not model throughput for SATF and RSATF here because we do not have a closed-form result. Instead, our results in Section 4.1.2 show that the performance gap between RLOOK and RSATF is small because both scheduling algorithms consider rotational delays. Once the detailed low level disk layout is understood, RLOOK is easier to implement than RSATF and therefore is an attractive scheduling policy for an SR-Array. RLOOK's analytical model is also better understood.

2.5 Summary

In this section, we have explored how by scaling the number of disks in a storage system we can (1) reduce seek distance, (2) reduce rotational delay, (3) reduce overall latency by combining these techniques in a balanced manner, and (4) improve throughput. To achieve these goals, the storage system needs to be configured based on a number of parameters. We have developed simple models that capture the following parameters that influence the configuration decisions:

- disk characteristics in the form of seek and rotational characteristics (S and R),
- read/write ratio (p),

- load on the system (Q),
- seek locality (L), and
- overhead time(T_o).

We have discussed disk modeling basics and existing approaches of using disk arrays to improve disk system latency. Based on the analysis, we have introduced the SR-Array disk array configuration family as an alternative. We have further derived a latency model for SR-Array and obtained the optimal SR-Array configuration by balancing the reductions on both seek time and rotational delay. Our results reveal that the non-overhead part of the latency, or the scalable part of seek time and rotational delay, can be reduced by a factor of $\sqrt{D}$ at best when we use a D-disk SR-Array with the optimal configuration. This latency time reduction is better than that achieved on simple striping and rotational replication. Later in Chapter 4, we show that the average latency of an SR-Array is also better than mirroring-based data replication schemes, such as RAID-10.

We have also analyzed the overall throughput of a single disk and that of an SR-Array. Specifically, we have developed mathematical models for the LOOK and the RLOOK scheduling algorithms. Although these results are approximations, our models lead us to believe that by balancing the reduction of seek time and rotational delay between consecutively scheduled requests, SR-Array with RLOOK scheduling policy can generate better throughput. We have further showed that when the number of outstanding random requests is fixed and much greater than the number of disks for a disk array system, the throughput of an optimally configured SR-Array is approximately proportional to the number of disks in the array (D) thanks to its ability to adjust the degrees of seek time and rotational delay reduction given different numbers of disks and workload conditions.

Our results show that there does not exist a single "perfect" SR-Array configuration; instead, there may exist one "right" configuration for every workload and cost/performance specification. As we increase the number of disks, and if we properly configure the storage system under the right conditions, the various results in this section suggest the following rule of thumb: by using D disks, we can improve the overhead-independent part of response time by a factor of $\sqrt{D}$ and the fixed queue length throughput by a factor of D.

Chapter 3

Experimental Framework

In this chapter, we describe an experimental framework that is designed to verify our theoretical modeling results and to learn more insights on various configurations and workloads.

3.1 Design Goals

In our theoretical analysis, we have made an assumption that we can access accurate relative position information between disk head and the fast rotating disk platter at any time. Unfortunately, there is no existing system that can accomplish this without hardware support in a disk array system. We have designed and implemented a method that tracks the disk head position accurately on off-the-shelf commercial disk drives without any special hardware support. This mechanism is a crucial requirement in the realization of the SR-Array configuration family. It also enables the implementation of rotational position sensitive scheduling algorithms (such as Shortest Access Time First (SATF) [26, 43]) across all the disks in a disk array. Even if the drives themselves perform sophisticated scheduling internally, it would have been

difficult to choose replicas intelligently without this head-tracking mechanism when there are multiple replicas on different disk drives.

Although accurate head tracking mechanism is a critical component, it is only one of the requirements for a whole disk array system to work. In this chapter, we describe an experimental framework for general disk array studies. It helps us validate our theoretical analysis results and gain more insights on how these disk arrays behave under the various configurations and workloads that we have not analyzed mathematically.

Simplistic simulations that are not based on accurate disk performance models may not be accurate enough for the disk studies that we conduct. Our simulation environment is based on a sophisticated and realistic model that is shared by the simulator and a prototype array implementation on the Microsoft Windows 2000 operating system. We show that the behavior of the simulator closely matches that of the implementation.

3.2 Software Design Overview

3.2.1 Multiple Software Configurations

We design our experimental framework in a layered architecture. Figure 3.1 illustrates the main system components and how they stack against each other.

There are a number of ways to configure the system. At the top level, the user-visible interface can be a user level disk driver so that the whole system can be used as a library. It can also be configured as a normal storage volume (such as the Z: drive in the Window 2000 Operating System) so that the whole system is packaged as an OS kernel device driver and is accessible by normal applications.

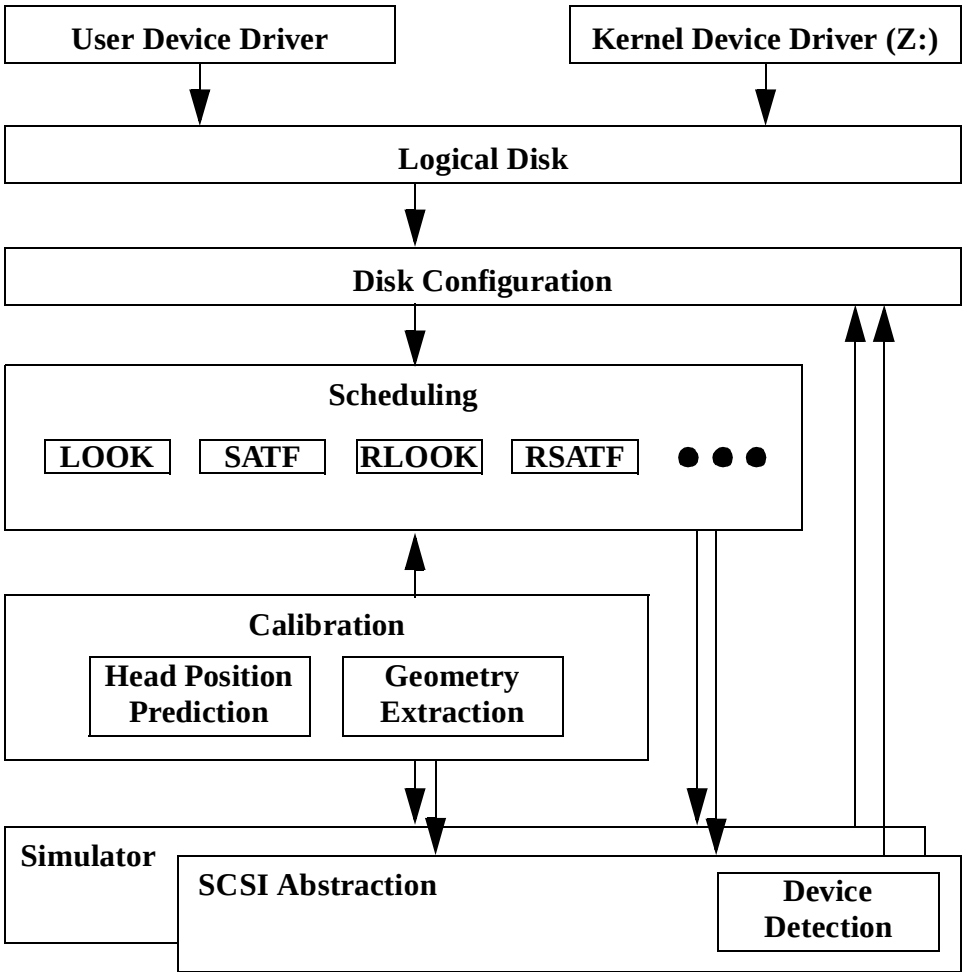


Figure 3.1: Prototype software architecture. The arrows in the figure indicate the information flow direction.

When used as a kernel device driver, the bottom SCSI abstraction layer interacts with real disk drives. When used as a user level driver, the bottom layer can be either a disk timing simulator or real SCSI disk drives controlled by a special I/O Control API supported by the operating system. The remaining components in the middle are shared across all software configurations.

3.2.2 Software Layers

The *SCSI Abstraction Layer* abstracts device-specific operations such as SCSI device detection at boot time, issuing SCSI read/write commands, and the retrieval of command status. We currently support two types of 10,000 RPM SCSI drives: Seagate ST34502LW [40] and ST39133LWV [41]. However, all experimental results reported in this chapter are from the ST39133LWV model. This layer has two modes: kernel mode and user mode. In the kernel mode, it relies on the Windows 2000 layered device driver architecture to issue SCSI commands to the lower layer SCSI disk driver. In the user mode, it can use either the Advanced SCSI Programming Interface (ASPI) developed by Adaptec [2, 39] or the SCSI Pass-Through Interface (SPTI) natively supported by Windows NT and Windows 2000 [39] to issue various SCSI-specific IOCTL (I/O control) calls directly to any SCSI disk drives.

A parallel layer to the SCSI abstraction layer is the *Simulator*. We have decided to integrate the simulator into the architecture to shorten the simulation time for long traces. Using simulation, we not only avoid waiting during the idle time in workloads, but also replace long disk I/O time with simulated time. The simulator also provides the flexibility of exploring the impact of changing disk characteristics. To faithfully simulate the behavior of the disks that we currently use in the prototype, the simulator receives timing information from the calibration layer to configure itself.

The *Calibration Layer* is used for calibrating the disk and extracting information regarding the physical layout of the disk. It keeps track of where the disk head is currently located and calculates the time required to move the head from its current position to a target sector. Section 3.3 provides more details about our head-tracking technique.

The *Scheduling layer* implements several disk scheduling policies, such as FCFS,

LOOK, CLOOK, SSTF, SATF, RLOOK, and RSATF. This layer maintains a read/write command queue for each physical disk and invokes a user determined policy to pick the next request at each scheduling step. We call these queues the *drive queues*. Sections 3.4.1 and 3.4.2 provide details of the scheduling policies of both read requests and write requests.

The *Disk Configuration Layer* provides support for configuring a collection of disks using techniques such as D-way mirroring, striping, SR-Array, and SR-Mirror. It translates I/O requests for a logical disk to a set of I/O commands on the physical disks and inserts them into the appropriate drive queues. The striping unit is 64K bytes in our experiments.

The *Logical Disk Layer* exposes a logical disk to the application. The kernel device driver exposes a mounting point (e.g. drive letter Z: in Windows 2000) to user space. The user level driver exposes the disk in the form of an API to the application.

3.2.3 Benefits of Code Sharing

Our simulator and the SCSI abstraction layer expose the same interface to upper layers so that we can use a single version of the source code for upper layers for both the simulation and real disk modes. This design greatly simplifies the debugging process of our system because we can avoid the trouble of debugging a kernel device driver by performing most of the upper layer debugging and experiments, such as those in the numerous scheduling policies and disk configurations, in a user library.

For similar reasons, the user level device driver is designed to access real disk drives in order to efficiently debug SCSI commands to a real disk drive. The result of this effort is that our prototype device driver becomes stable enough in a short period of time to allow us to start systematically evaluating different disk array configurations

under different workloads. The fact that the simulator and the implementation share most of the source code also contributes to our confidence in the validity of the simulation results.

3.3 Disk Calibration and Head Tracking

Both the SR-Array disk configuration family and the SATF scheduling policy depend on our ability of accurately predicting the timing of head movement. In analyzing the SATF scheduling policy and its variants, previous proposals [6, 48] that depend on the knowledge of head positions have relied on hardware support. Unfortunately, this level of support is not always available in commodity hard drives. We have developed a software-only head-tracking method using off-the-shelf commodity hard drives without special hardware support. This section details several aspects of our implementation, which include (1) measuring timing of the relative movement of heads and disk platters; (2) measuring sector layout of a disk; (3) maintaining head tracking accuracy; and (4) predicting access time during runtime.

3.3.1 Measuring Timing and Layout of A Disk

In order to predict the time required for the head to move from its current position to a target sector, we need the following steps. First, we need an accurate reading of a clock with good enough resolution. Second, we need to know the current relative position between the head and the platter, which requires us to track the full rotation time with good accuracy. Third, we need to know how sectors are laid out on the disk platters. Finally, we need to know how much time is needed for the head to move over a certain cylinder distance. All these tasks are done in the calibration layer.

Reading The Clock

The Windows 2000 Platform SDK provides a high resolution timer API that provides several million counts per second. Therefore, the resolution of this API is at microsecond level. One full rotation time of a 10,000 rotations per minute (RPM) SCSI hard drive is around six milliseconds. If the reading of the full rotation time is off by just one microsecond, it only takes 6000 rotations, or roughly 36 seconds, for our prediction to potentially miss one full rotation. For our prototype, this kind of resolution is not good enough.

We have two ways of improving the clock reading resolution. First, we use a special assembly instruction to read the cycle counter provided by the Intel Pentium processor family. As all the current processor runs at at least 100MHz, the cycle counter can yield a much higher resolution than the microsecond level. Second, we measure the time of tens of thousands of disk rotations. This way, even if the clock reading is off by ten microsecond, the amortized discrepancy of one full rotation is still well below the microsecond level. Both methods are implemented in our framework.

Measuring Full Rotational Time

We first issue two read requests to a fixed *reference sector*. When we issue the read requests fast enough, the time between the finishing times of the two requests is roughly one full rotation time. This time does not have a high degree of accuracy because the differences of the unpredictable amount of overheads in the operating system, the I/O buses, the disk controller and other places can sum up to a significant amount of time (as much as tens of microseconds). We use a similar method to measure the time of multiple full rotations by issuing multiple read requests consecutively. This way, we can improve the accuracy of one full rotation time by amortizing the over-

head difference over multiple rotations. However, this technique requires the head to constantly read the reference sector and therefore it cannot do anything else. We call this *synchronized measurement*. Moreover, when we perform n consecutive requests, it is possible that the disk actually takes more than n rotations to fulfill these requests, especially when n is large. Even if we missed only one rotation out of 1000 rotations, our estimated one full rotation time can be off by six microseconds, which is still significant.

To overcome the drawbacks of the synchronized measurement, we have developed a method, which we call *asynchronous measurement*, to allow the disk to perform other operations between the readings of the reference sector. The basic idea is that no matter how many rotations have passed, the time between two read accesses of the reference sector should always be close to an integral multiple of the full rotation time plus an unpredictable overhead, which is usually several microseconds.

If we assume that we already know that our current estimate of one full rotation time is within 1% of the actual full rotation time, and if we let the disk run for a time that is roughly 25 full rotation time between two reads of the reference sector, our prediction offset will be at most 25% of one full rotation time. Because it is less than 50%, we can still safely use our current estimate of the full rotation time to deduce the *exact* integral number of rotations that have passed. This estimate of the number of rotations is accurate because the error of the single rotation time estimated is not sufficient to cause the estimation to be off by one. We can then divide the time interval by the number of rotations to derive a new but more accurate full rotation time. We can also estimate that the accuracy range of this new estimate is the difference between the old and the new estimates. After that, we can start a new round of measurement but the new allowable time interval can be longer.

By gradually increasing the time interval between adjacent read requests of the

reference sector, we can amortize the unpredictable overhead over tens of thousands rotations and obtain a very accurate estimate of the single full rotation time.

$$N(k+1) = \frac{\alpha R(k)}{\Delta(k)} \tag{3.1}$$

$$\begin{aligned} \Delta(k+1) &= \frac{\gamma}{N(k+1)} \\ &= \frac{\gamma}{\alpha R(k)}\Delta(k) \end{aligned} \tag{3.2}$$

We now describe the above process formally. For the kth measurement iteration, let $R(k)$ be the computed single full rotation time after the iteration, $N(k)$ be the number of rotations that we roughly allow in the iteration, γ be the upper bound of the unpredicted OS and hardware overhead in measuring request finishing time, and $\Delta(k)$ be the upper bound of the estimated difference between $R(k)$ and the real full rotation time. Equation (3.1) computes the number of rotations in the next iteration that guarantees that the aggregated rotational angle error cannot exceed a certain percentage (α) of the full rotation time. This guarantee ensures that we can compute the accurate integral number of rotations during the next iteration. To be safe, we can pick an α that is less than 25%. Equation (3.2) then estimates the upper bound of the difference between $R(k)$ and the real full rotation time for the next iteration. As long as $\alpha R(k) \approx \alpha R > \gamma$, $\Delta(k+1) < \Delta(k)$, and we should achieve a better estimated rotation time $R(k+1)$ in the next iteration.

However, this process can not go on forever because it depends on the assumption that the real full rotation time is stable within a small range. Once we obtain a $\Delta(k)$ that is significantly small, the drift in real full rotation time may make us unable to get an accurate reading of the number of rotations in the iteration and, consequently, we are unable to obtain more accurate estimates. Note that a small change of either

the CPU clock generator or the disk full rotation time can cause the drift because the time we measure is in terms of the number of cycles of the CPU clock.

For the more than ten Seagate disk drives that we have used, their full rotation time is remarkably stable. Our experiments show that periodic re-calibration at an interval of two minutes yields predictions that have an error of only 1% of a full rotation time with 98% confidence. Because the kernel device driver has higher CPU priority, we can achieve this accuracy even under heavy load when the CPU is near 100% usage. We achieve a $\Delta(k)$ that is less than three nanoseconds for a γ that is roughly 60 microseconds.

It is encouraging that we can achieve a high degree of accuracy with a low overhead associated with reading the reference sector every two minutes. To further reduce this overhead, we can exploit the timing information and known disk head locations at the end of user requests. We have not implemented this optimization because our current method is already accurate enough and has very low overhead.

Measuring Sector Layout

Our scheme of measuring full rotational time also yields the approximate time of zero relative rotational angle between the head and the reference sector. Together with the full rotation time, we can calculate, at any instance, the relative rotational angle between the head and the reference sector. In order to get the relative position between the target sector and the current head location, we also need to know the relative position between each sector and the reference sector.

Given the LBA of a target sector, we can use a SCSI command to translate the LBA to the cylinder number, the head number and the sector number within the track. We can also use the same type of command to measure the exact number of sectors per track. We can then derive the relative rotational angle between the target

sector and the first sector in the track, safely assuming that all the sectors in the track are evenly spaced. We also obtain zoning information during the process.

The last piece of layout information that we need is the relative rotational angle between the first sector of the track and the reference sector. We obtain this information by noting the timestamp after reading the first sector. Note that it may not be true that the first sector of every track is located at the same rotational angle. The difference between adjacent tracks is called *track skew*. Track skews allow the disk head to reach the end of one track, take some time to switch and settle on the next track, and continue at the beginning of the next track at full rotation speed without any undue delays.

We obtain information on disk zones, track skew, bad sectors, and reserved sectors through a sequence of low-level SCSI disk operations. Worthington et al. explains a similar process in more detail [53]. The whole process is slow even if we use various optimizations. Therefore, we store the result of layout information at a known place on the disk so that we can retrieve it after reboots without having to re-derive the sector layout map. We also use the layout information to convert a target LBA into a relative position between it and the reference sector as part of the access time calculation.

Measuring Minimum Read Time

The last piece of timing information that we need is the cost of performing track switches and seeks. We measure the minimum time differences between two consecutive reads (and writes) to different tracks and to different cylinders for certain well-spaced cylinder distances. We use interpolation to derive the minimum read/write times for other cylinder distances. We perform the measurement several times. Our experiments indicate that the least access times are accurate within a range of $\pm$ 100

μs for short seeks that move heads less than 10% of the total number of cylinders and $\pm$ 200 μs for long seeks. Considering that a full rotational delay is 6 ms on these disks, we regard this level of accuracy (within 3%) to be satisfactory for the various degrees of rotational replication that we employ.

3.3.2 Runtime Access Time Prediction and Adjustment

During runtime, the scheduling layer uses the information gathered in the calibration layer to determine the locations of the requests and the order in which the requests are scheduled. For those scheduling policies that require the access time to be part of the scheduling decision, such as the SATF scheduling policy, the decision requires an estimation of the current head position and a prediction of the time needed for the head to reach each candidate location. We have already discussed the head position estimation. We now discuss access time prediction and runtime adjustment of the prediction.

Predicting Access Time

The arm of a disk drive stays in a cylinder unless it is directed by software to move to another cylinder. Therefore, our system can always keep track of in which cylinder the head is located. We need to have two pieces of information to predict the access time: (1) the seek time that the head needs to reach the target cylinder; and (2) the rotational delay that the head needs to wait on the target track before the target sector rotates under the head.

We look up the table of the least read/write time to predict the time that the head needs to reach the cylinder. We also know the relative rotational angle between the head and the target sector after the head reaches the cylinder and, therefore, we

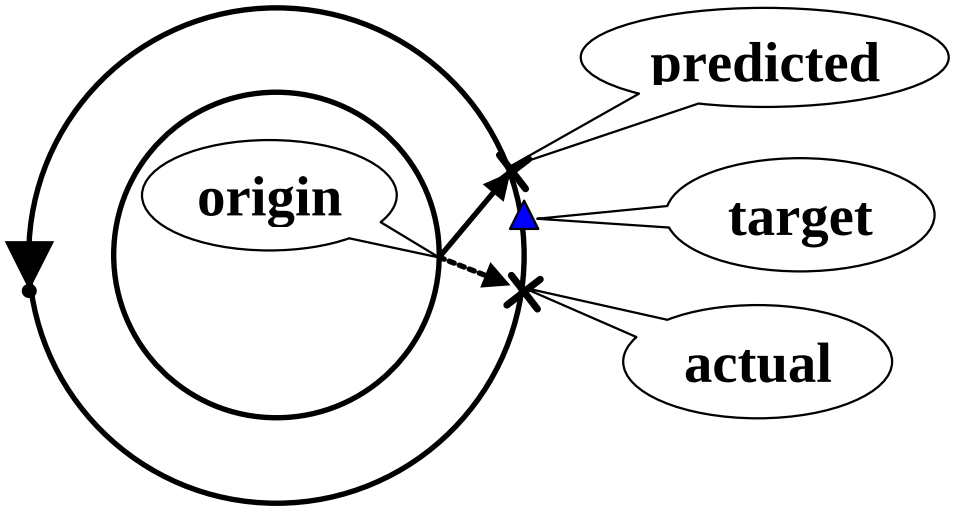


Figure 3.2: An illustration of missed rotation

have an estimate of the rotational delay.

Runtime Feedback and Adjustment

Because our timing measurement might drift in the small range around the actual timing, our estimation of when the head can settle on the target cylinder can be off a little. If our prediction time is too short, the head only needs to spend the extra time waiting on the target cylinder. However, if our prediction is too long and we happen to need the head to read a sector very soon after it settles, the head may just miss the target sector and it may have to wait for almost a full rotation to reach the target again. In this situation, our small prediction error may lead to a large penalty of nearly one full rotation time. Figure 3.2 illustrates the situation of a missed rotation because of a prediction error.

To reduce the number of rotation misses, we add a slack time into the estimation of the seek time. A larger slack time may cause the scheduler to over-predict the access time required for those requests that need very small rotational latency. As

a result, we may unnecessarily delay the scheduling of these requests. To limit the inefficiency resulting from this slack time, we would like the slack time to be as small as possible.

The system could use different slack time for different seek distance and constantly monitor the number of missed rotations and adjust the slack time accordingly.

3.4 Scheduling Details

3.4.1 Scheduling Reads

The head-tracking mechanism, along with the accurate models of the disk layout and the seek profile, allows us to implement sophisticated local scheduling policies on individual disks; these include RLOOK, SATF, and RSATF.

Scheduling on a mirrored system, however, is more complex due to the fact that a request can be serviced by any one of the disks that have the data. We use the following heuristic scheduling algorithm in this situation. When a read request arrives, if some of the disks that contain the data are idle, the scheduler immediately sends the request to the idle disk head that is closest to a copy of the data. If all disks that contain the desired data are busy, the logical disk layer duplicates the request and inserts the copies into the drive queues of all these disks. Each disk drive does the specified scheduling locally. However, as soon as such a request is scheduled on one disk, all other duplicate requests are removed from all other drive queues. When a disk completes processing a request, its local scheduler greedily chooses the "nearest" request from its own drive queue. Although this heuristic algorithm may not be optimal, it can avoid load imbalance and works fairly well in practice.

3.4.2 Delayed Writes

While multiple copies of data reduce read latency, they present a challenge for performing writes efficiently because more than one copies need to be written. We need to make $D_r \times D_m$ copies for a $D_s \times D_r \times D_m$ SR-Mirror. It is however feasible to propagate the copies lazily when the disks are idle. We can issue a write to the closest copy and delay writing the remaining copies. On the other hand, reads of data blocks whose propagation has not yet completed cannot enjoy the full benefit of latency reduction. Fortunately, the likelihood of such occurrence can be reduced by caching the content of writes because reads that hit in cache will no longer be issued to the disk. For back-to-back writes to the same data block, which happens frequently for data that die young [36], we can safely discard unfinished updates from previous writes. This is similar to the approach taken by the AFRAID system [38].

In our implementation, we maintain for each disk a *delayed write queue*, which is distinct from the foreground request queue. When a write request arrives, we initially schedule the first write using the foreground request queue just as we do for reads. As soon as writing one of the replicas is scheduled, we move the remaining update operations for the other replicas into the individual delayed write queues. Entries from this queue are serviced when the foreground request queue becomes empty on the corresponding disk. Delayed writes require us to make a copy of the data because the original buffer is returned to the OS as soon as the first write completes.

To aid crash recovery, the physical location of the first write is stored in a *delayed write metadata table* that is kept in NVRAM. Note that it is not necessary to store a copy of the data itself in NVRAM—the physical location of the first write is sufficient for completing the remaining delayed writes upon recovery; so the table is small. When the metadata table fills up to a threshold (10,000 entries in the current imple-

mentation), we force delayed writes out by moving them to the foreground request queue.

We have implemented the basic delayed write strategy described above. Other potentially better alternatives also exists. In fact, the general idea of delayed write is not new. Polyzois [35] proposes careful scheduling of delayed writes to different disks in a mirror to maximize throughput, a technique that can potentially benefit delayed writes in our system when the replicas are on different disks. The "distorted mirror" [34] provides an alternative way of improving the performance of writes in a mirror. It performs writes initially to rotationally optimal but variable locations and propagates them to fixed locations later. This technique can be integrated with our delayed write strategy as well.

3.4.3 Limitation

Although we model precisely the disk rotation and head seek/settle timing, our model for data transfer and various overhead in SCSI and disk interfaces are less sophisticated than the state of art. We do not model disk prefetching, read caching, write buffering and command overlapping and have these features turned off for disks used in our prototype. While we do full scheduling in the driver for the entire disk array, we restrict that each physical disk has at most one outstanding request at any time. As some modern SCSI disks are capable of performing SATF-like scheduling inside their disk firmware efficiently, the software-based SATF scheduling performance in our prototype can be worse than that of firmware-based solutions for a single disk because of communication overheads. It is possible that we may be able to further improve the performance of our system by modeling the full set of firmware features and still keep our advantages of being able to (1) deal with rotational replicas, and

(2) cooperatively schedule requests among all available disks in mirroring situations.

3.5 Simulation

3.5.1 Simulator

We use an event driven simulator. It has an event queue ordered by event time. The simulator always processes the event at the head of the queue. When the event is processed, the simulator adjusts the system's virtual time and invokes a callback routine associated with the event. The simulator may also monitor the time consumed in the callback routine and advance the virtual clock accordingly. The simulator estimates the completion time of each request just as described in section 3.3.2 but without the "slack time".

Our simulator can also introduce some small random noise in the timing information of seek and rotational delay. This simulates the behavior of mechanics on real disks, which never perform at the exact timing even when repeating the same operations. The type and the size of these random noise can be measured when the system is running on real disk arrays and these measurements can be then configured into the simulator.

3.5.2 Validating The Integrated Simulator

So far, we have described the architecture and the components of our experimental framework. To establish (1) the accuracy of the head-tracking mechanism, and (2) the validity of the simulator, we perform a series of experiments using "Iometer", a benchmark developed by the Intel Server Architecture Lab [25]. Iometer can generate workloads of various characteristics including read/write ratio, request size, and the

Operating system	Microsoft Windows 2000
CPU type	Intel Pentium III 733 MHz
Memory	128 MB
SCSI Interface	Adaptec 39160
SCSI bus speed	160 MB/s
Disk model	Seagate ST39133LWV 9.1 GB
RPM	10000
Average seek	5.2 ms read, 6.0 ms write

Table 3.1: Implementation platform characteristics.

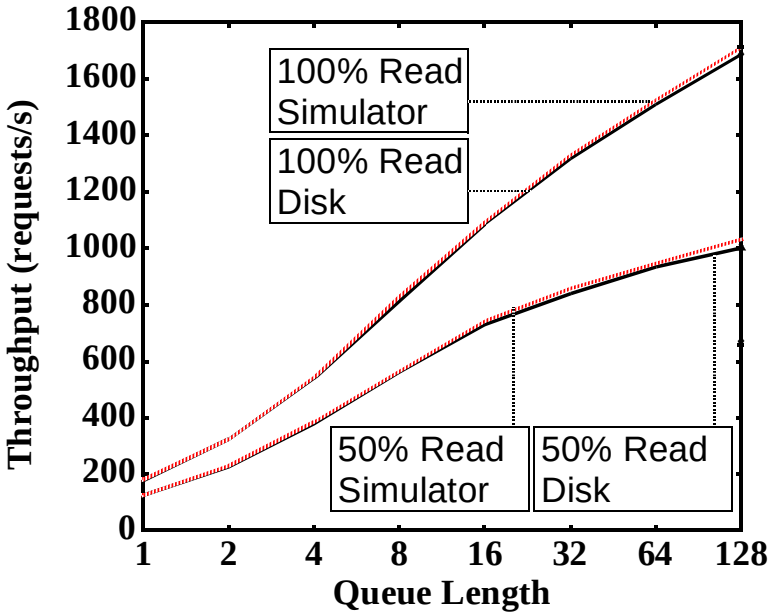


Figure 3.3: Comparison of throughput results from the prototype system and the simulator. We use two random workloads, one with just reads, and another with an equal number of reads and writes. The request size is 512 bytes. The array configuration is a 2×3 SR-Array based on the RSATF scheduler. Writes are synchronously propagated in the foreground. We vary the number of outstanding requests (on the x-axis).

Misses	0.22%
Mean Prediction Error	3 μs
Standard Deviation of Error	31 μs
Average Access Time	2746 μs
Demerit	52 μs
Demerit/Access Time	1.9 %

Table 3.2: Detailed statistics of simulator accuracy when subjected to the "Cello base" file system workload. The configuration is a 2×3 SR-Array based on RSATF scheduling. I/O requests in this experiment are physical I/O requests sent to drives; and access time is that of a physical I/O. Prediction error is the difference between the access time predicted by the scheduler and the actual measured access time of a single request. We calculate demerit using the definition by Ruemmler and Wilkes [36].

maximum number of outstanding requests. We use Iometer to generate equivalent workloads to drive both the device driver and the simulator. Table 3.1 lists some platform characteristics of the prototype. Figure 3.3 shows the results of these Iometer experiments. The throughput discrepancy between the simulator and the prototype under all queueing conditions is under 3%.

To shed more light on the accuracy of our simulation, in Table 3.2, we give more detailed statistics by subjecting the simulator and the prototype to the "Cello base" file system workload (described in Section 4.2.1). We organize the disks into an SR-Array configuration. The small mean prediction error and low standard deviation show that there are essentially only two types of prediction results: 99.8% of the predictions are almost right on target, and 0.2% of the predictions miss their targets by a very small amount of time and incur a full rotation penalty. The net effect of these rare rotation misses, however, is insignificant in terms of overall access time. These results indicate that the simulator faithfully simulates a real SR-Array, allowing us to understand the performance behavior of the SR-Array using simulation-based results in later empirical studies.

3.6 Summary and Related Work

In this chapter, we have described our experimental framework with an accurate disk head position tracking scheme and an integrated simulator. We have shown that the simulator faithfully reflects the timing of requests of a real system. This is due to the fact that most of the code is shared between the simulator and the real prototype, which can run as a real storage volume under Windows 2000.

There have been several simulation studies on scheduling policies for a single disk [12, 19, 26, 43, 49, 51]. We integrate and validate our simulator with a real device driver and study both single disks and disk arrays.

Our implementation is only one of the recent efforts to track disk head position. At the time of this research, the Trail system [24] has independently developed a disk head tracking mechanism that is similar to ours. Trail uses this information to perform fast log writes to carefully chosen rotational positions. Aboutabl et al. has developed a similar disk timing measurement strategy, which is used to model the response time of individual I/O requests [1]. In comparison, our head-tracking mechanism achieves a higher degree of accuracy while incurring much lower overhead. We have also implemented various scheduling policies, which can handle heavy workloads.

The importance of reducing rotational delay has long been recognized. Seltzer and Jacobson have independently examined a number of disk scheduling algorithms that take rotational position into consideration [26, 43] with the assumption that rotational position information is readily available. A number of hard drive manufacturers have also incorporated SATF-like scheduling algorithms in their firmware. An early example was the HP C2490A [18].

Our host-based software solution enables the employment of SATF-like scheduling on hard drives that do not support such scheduling internally. Furthermore, it allows

experimentation with strategies such as rotational replica selection, which would have been difficult to realize even on the hard drives that support intelligent scheduling internally. We have also considered the impact of reducing rotational delay in array configurations in a manner that balances the conflicting goals of reducing seek time and rotational delay at the same time.

Rotational position-aware scheduling algorithms, such as SATF, are not the only applications of an accurate head tracking mechanism. We have already mentioned that the Trail system [24] uses this information to perform fast log writes to carefully chosen rotational positions. A similar write strategy is also used in the earlier Mime system [6], though Mime relies on hardware support for getting its rotational positioning information. Lumb et al. [29] exploit "free bandwidth" that is available when the disk head is seeking during servicing normal requests in a busy system. The free bandwidth is used for background I/O activity. Our delayed write scheme for propagating replicas in our system can also take advantage of this free bandwidth. Wang et al. [48] propose to rely on hardware support to realize "eager writing" in the "virtual-log based" file system. "Eager writing" refers to the strategy of writing data to the closest possible free block. Zhang et al. [55] have extended both this thesis work and the virtual log-based file system work so that they can be applied to disk arrays.

In summary, based on our accurate head tracking mechanism, we have implemented a prototype experimental framework that supports various configurations of the SR-Array configuration family with various scheduling policies. We have also integrated a simulator in this framework and have validated its accuracy.

Chapter 4

Empirical Study

We now use the framework discussed in the last chapter to study various disk array configurations under different workloads. We devote one section to each of the following three objectives, with Section 4.4 summarizing the conclusions of our study.

First, we would like to verify some mathematical models that we have developed in Chapter 2. These models are based on a few simplified assumptions that are close to but not the same as reality. For example, we model the seek time or the minimum access time as being proportional to the seek distance. This overestimates the time required when the seek distance is relatively small, such as when it is less than 10% of the maximum seek distance. Our model does not consider zoning. This is based on the assumption that a simplified model without zoning considerations is a good approximation of reality. Once we validate these models, we can then rely on the models to estimate the best disk array configuration for a certain workload. The results presented in Section 4.1 demonstrate that our mathematical models are indeed quite close to our measured results for controlled workloads.

Second, we would like to demonstrate that the SR-Array configuration family can improve the disk array performance more effectively than some existing disk

array configurations on some workloads. We also study the performance differences among various scheduling policies because the performance of different disk array configurations can be affected by the choices of scheduling algorithms. Section 4.2 uses trace driven simulation to demonstrate the strengths of SR-Array configuration family and its scheduling policies.

Third, because SR-Array can achieve low latency and high throughput, we would like to demonstrate the possibility that the use of more disks can sometimes be a more cost-effective way than the use of memory caching in improving overall system performance for certain workloads. Section 4.3 uses trace driven simulation and the relative cost between disk drives and memory to compare the performance improvements of the two schemes.

4.1 Validation of Mathematical Models

In this section, we verify our latency and throughput model built for SR-Array in Chapter 2. We generate workloads assumed by the models and feed them into the simulator. Before running the simulation, we also run the same workloads on our experimental platform with a few real disk array configurations and validate our simulator results. Section 3.5.2 provides more details on how we validate our simulator.

For both latency and throughput models, we use workloads generated by the Iometer micro-benchmark, which we have introduced in Section 3.5.2. This benchmark generates workloads suitable for validating our throughput model as we can specify the constant outstanding request queue length and the ratio between read and write requests in a workload. We also use the benchmark for our latency study by limiting the outstanding request queue length to one when generating the workloads.

We have designed our system to minimize the performance impact of foreground

parameter	value
S	$9.2ms$
R	$6.0ms$
T_o	$2.7ms$
T_{wo}	$0.7ms$

Table 4.1: Parameters used in disk models (gathered from the Seagate ST39133LWV)

propagation of newly written replicas. Our mathematical models do not take into account the availability of idle time for delayed write propagation in a real world workload. This simplified model is therefore a conservative performance estimate. The workloads generated by Iometer keep all the disks constantly busy. To validate the simplified models, we disable the delayed writes feature in our implmentation.

During our simulation, we generate 10,000 requests that are random in terms of locality. We only measure the performance of the middle 7,000 requests to avoid the effects of warming up and cooling down during the simulation. Because the configurations employing a single disk, simple striping and simple rotational replication are all special cases covered by the more general SR-Array configuration family, the data points of these cases are included in our evaluation of the SR-Array configuration family. In the real world, we may not necessarily encounter workloads like what we use in these micro-benchmarks. However, by running these controlled experiments, we hope to gain some insights of how various disk array configurations perform and understand the strengths and weaknesses of different configurations.

Table 4.1 lists the disk parameters used in the SR-Array models for the real disk array implemented in our prototype system. We have deduced these parameters from measurements as shown in Figure 2.3. As the "Linear approximation" line shown in that graph, the overhead part (the intersection with Y-Axis) of the minimum read time is $T_o = 2.7ms$. For a seek across the maximum seek distance on the disk we used, the minimum read time is $11.9ms$. This means that we model the minimum

read time as a straight line between $2.7ms$ and $11.9ms$ and deduce $S = 9.2ms$ for our model equations. All disks rotate at 10,000RPM, which translates to a full rotation time of $R = 6ms$.

In the models introduced in Chapter 2, we assume that the minimum read time is always the same as the least write time for a given seek distance. However, this is not always true. In fact, on our hard drives, we measured that the minimum write time is always roughly $0.7ms$ more than the corresponding minimum read time for a given seek distance. Therefore, we adjust our equations correspondingly in Chapter 2 to account for the more time spent on writes.

4.1.1 Latency Model

To validate the SR-Array latency model, we generate a new random request immediately after the last request is serviced. This way the system always has one outstanding request. We measure the average service time of all the requests under different configurations and read/write ratios. Because we only allow one outstanding request at a time, the choices of scheduling policies in our latency experiments do not affect the performance results. Therefore, we do not specify the scheduling policy used in the result figures.

We first measure the read only workloads of SR-Array ($D_s \times D_r \times 1$) configurations using different number of rotational replicas D_r and different number of disks. Figure 4.1 shows the experiment results and model results according to Equation (2.19). These experiments confirm that Equation (2.19) models the latency results well with varying number of disks in the array. We also see that the curves approach asymptotes as more disks are used and the benefit of a higher degree of striping reaches diminishing return.

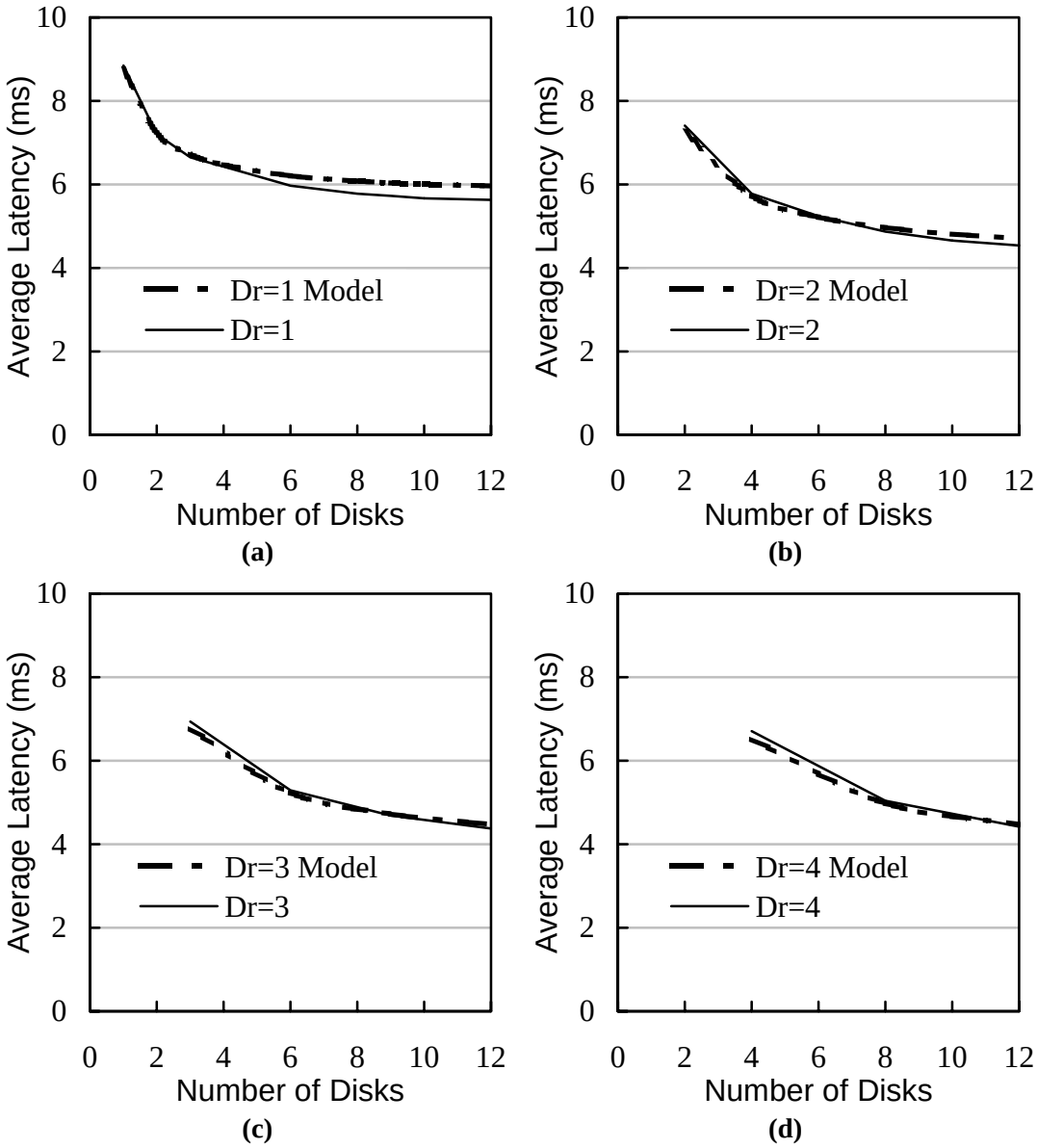


Figure 4.1: Validation of the SR-Array read-only latency model: average latency as a function of the number of disks used in an SR-Array and the degree of rotational replication: (a) $D_r = 1$; (b) $D_r = 2$; (c) $D_r = 3$; and (d) $D_r = 4$. The workload is random reads of one 4K-byte block. Each figure includes a measured curve and a model curve based on Equation (2.19).

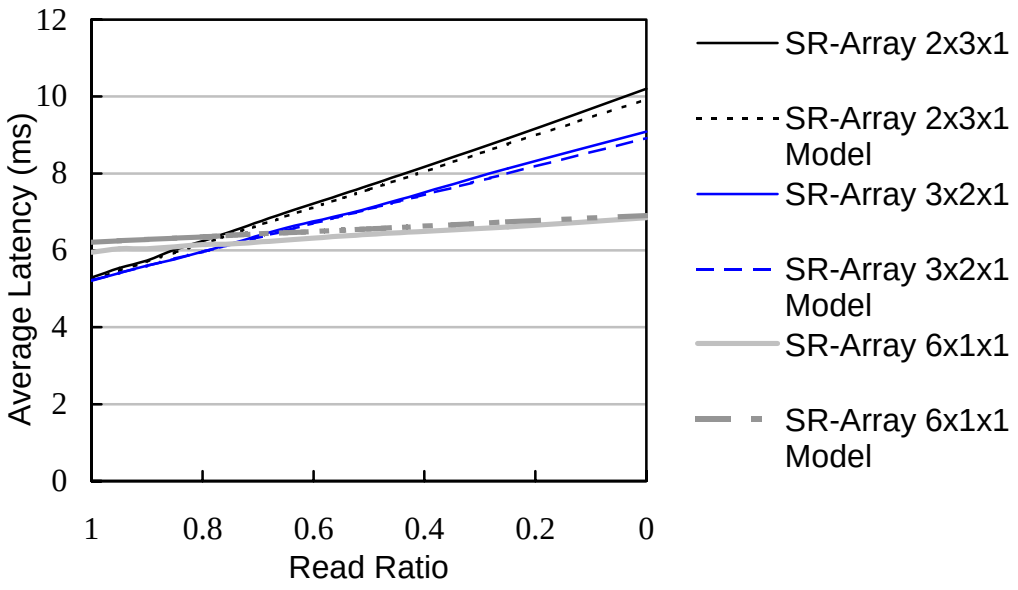


Figure 4.2: Validation of the SR-Array read/write latency model: average latency as a function of the read/write ratio in different 6-disk SR-Array configurations: $2 \times 3 \times 1$, $3 \times 2 \times 1$ *and* $6 \times 1 \times 1$. *The workload is random reads/writes. Each configuration includes a measured curve and a model curve based on Equation (2.26).*

Now we keep the number of disks in the array to be a constant of six and vary the read/write ratio under different SR-Array configurations. The results in Figure 4.2 confirm that Equation (2.26) is a good improvement over Equation (2.19) when considering the read/write ratio. These experiments also show that the $3 \times 2 \times 1$ SR-Array is the best among the three configurations when the read ratio is larger than 0.8 and striping with no rotational replication (the $6 \times 1 \times 1$ configuration) has the best latency when the read ratio is below 0.8. The curve of striping is not completely flat because each write is roughly $0.7ms$ more expensive than a read.

In Equation (2.11), we show that the average rotational delay is always a half rotation time in a 50% read workload no matter how many rotational replicas are used. We verify this property by using a workload containing an equal number of reads and writes. We change D_r and keep D_s constant for each curve. The flat latency

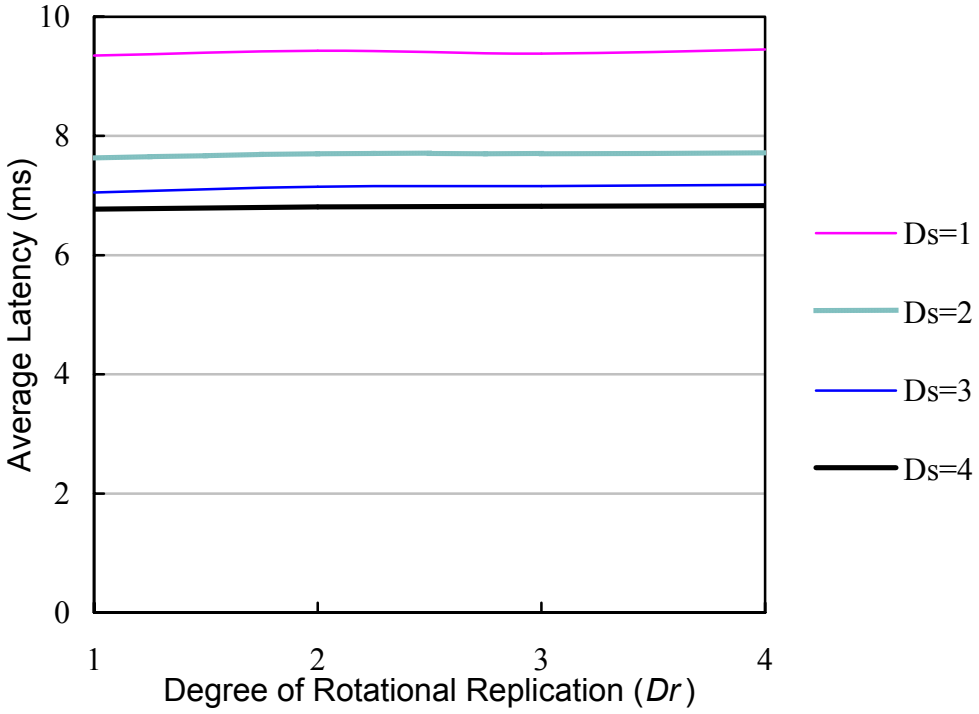


Figure 4.3: Validation of an invariant of rotational replication: average latency as a function of the degree of rotational replication (D_r) in SR-Array configurations with different degrees of striping (D_s). The workload is random single block requests with equal number of reads and writes.

curves in Figure 4.3 confirm our theory. As we increase D_s, we see the corresponding curves grow closer, indicating a diminishing return of latency reduction.

The above experimental results confirm that our latency model of Equation (2.19) and (2.26) are accurate in predicting average single request latency in an SR-Array. Because the model is accurate on all different SR-Array configurations with different read/write ratios, it can accurately predict the configuration with the best latency given a fix number of disks in the SR-Array.

Now, we compare the latency of SR-Array ($D_s \times D_r \times 1$) with other existing disk array configurations: striped-mirroring ($D_s \times 1 \times D_m$), striping ($D_s \times 1 \times 1$)

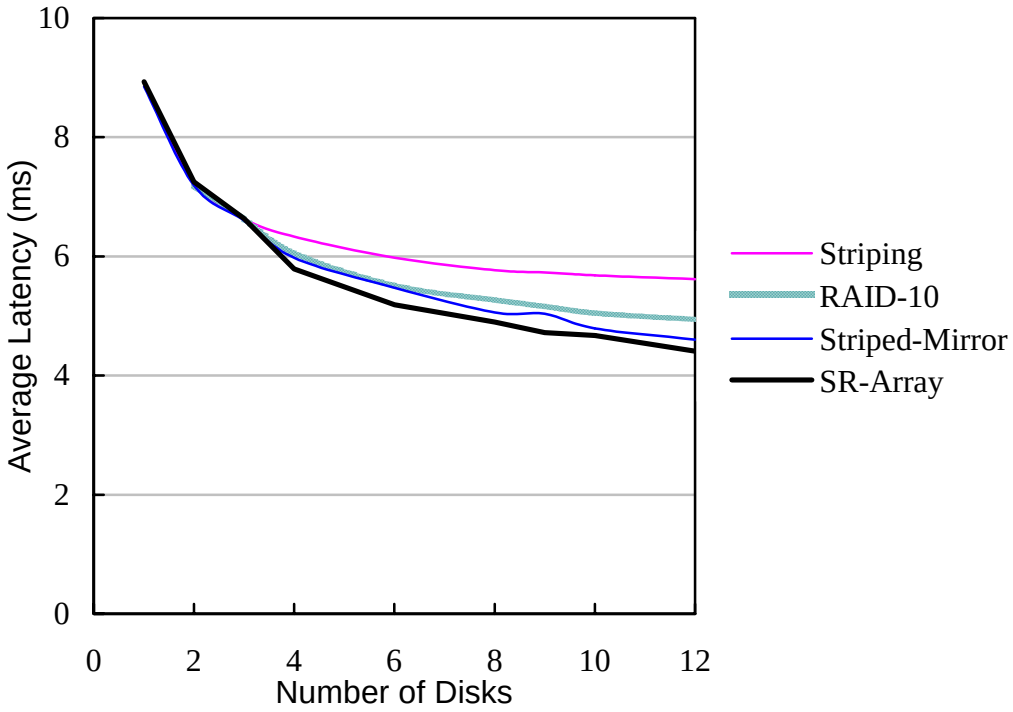


Figure 4.4: Comparison of latency of disk array configurations: average latency as a function of the number of disks in different disk array configurations: striping, RAID-10, striped-mirroring and SR-Array. The workload is random single block requests.

and RAID-10 ($D_s \times 1 \times 2$). The workload used here is read-only. Figure 4.4 shows that the latency can be best reduced when we allow the flexibility of picking the best configuration in both SR-Array and striped-mirroring. The larger the number of disks in the system, the larger the improvement because of the ability of balancing seek time and rotational delay reduction in these configuration strategies. On the other hand, striping results in the poorest latency because of the lack of rotational delay reduction. RAID-10 is in the middle because it allows limited rotational delay reduction using only two replicas instead of more.

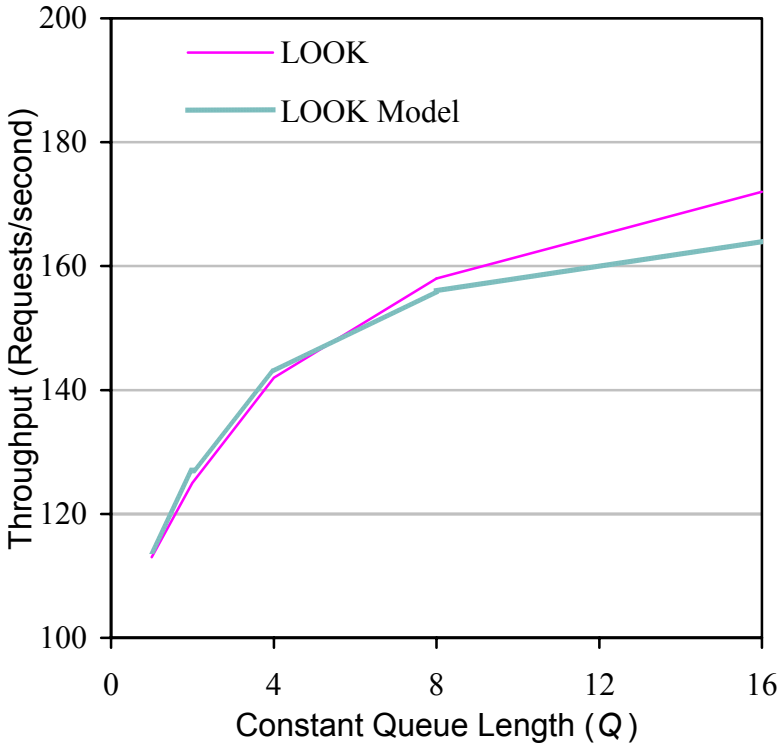


Figure 4.5: Validation of the LOOK scheduling model for one disk: throughput as a function of the request queue length for one disk with the LOOK scheduling policy. The figure includes a measured curve and a model curve based on Equation (2.41). The workload is random single block read requests with a constant number of requests outstanding.

4.1.2 Throughput Model

Our throughput model of the LOOK and RLOOK scheduling policies on SR-Array considers several factors: (1) the number of disks, (2) the disk array configuration, (3) the length of the outstanding request queue, and (4) the read/write ratio in the workload. We start our validation with read-only workloads on one disk. Then we examine SR-Arrays with more than one disk. After comparing the results with other configurations (such as RAID-10), we consider write requests. Lastly, we also validate our model by varying the read/write ratio.

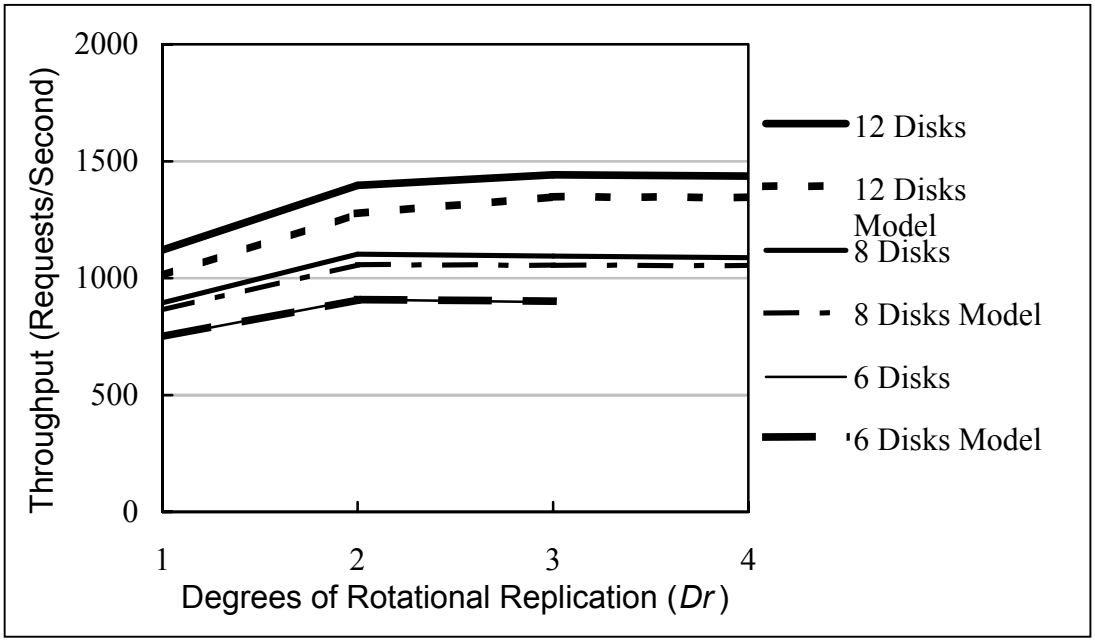
We first validate a single disk LOOK model by running random read requests. Figure 4.5 compares the read throughput results against those predicated by our

mathematical model in Equation (2.41). Although not shown in the figure, our data confirms that the average number of requests serviced during a single sweep in one direction is roughly $1.5Q$, consistent with the calculation in Section 2.4.2. Our model is close (less than 3% difference) to the measured data when Q is less than 10. When Q becomes larger than 10, the average seek distance is small enough such that the difference between the minimum access time and its linear model that we have used in earlier mathematical analysis, though still less than 8%, grows larger. This is expected for seek distances that are less than 1,000 cylinders based on what we have seen in Figure 2.3.

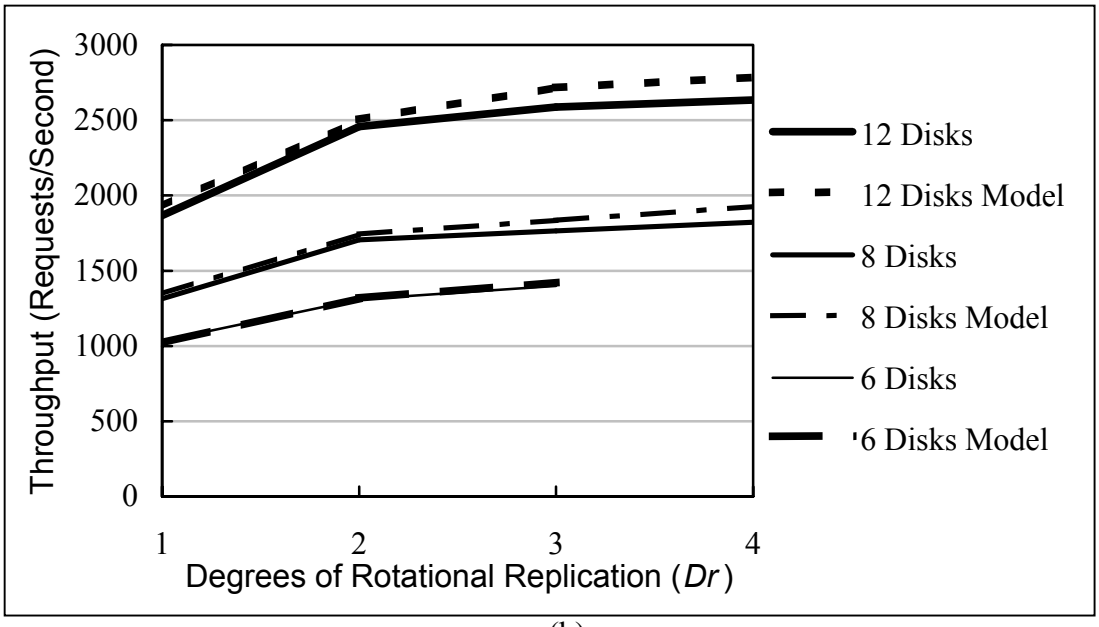
Now we run the same type of experiments on SR-Array. Figure 4.6 presents the measured and modeled results for 6, 8 and 12 disks. Each curve has several points representing different SR-Array configurations using the same number of disks. The workloads keep a constant number (8) of outstanding requests in the experiments of Figure 4.6(a) and 32 in Figure 4.6(b). This figure confirms that our mathematical model is largely consistent with the measured results. Most of the measured data points have errors that are less than 3% of what the model predicts. It also shows that increasing the degree of rotational replication (D_r) can improve the read throughput in these workloads, although one reaches diminishing returns quickly.

The result of 12 disks with a queue length of 8 shows the largest model error (which is still less than 10% for all experiments). This is mainly due to our inaccurate modeling of cases with short queue lengths (8 in this case). Despite the model errors, our model still captures the relative performance trend of different SR-Array configurations with the same number of disks. Because of the relative accuracy of the model, it can be safely used to predict the configuration that has the best throughput for a particular workload.

Figure 4.7 evaluates the relative performance of the same read-only workloads

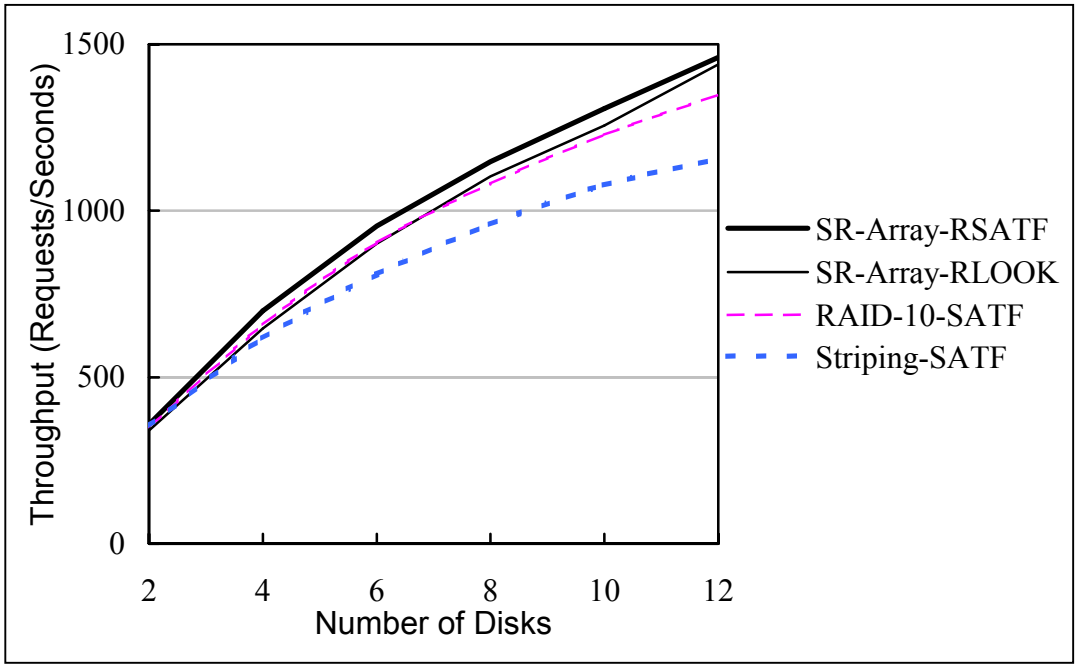


(a)

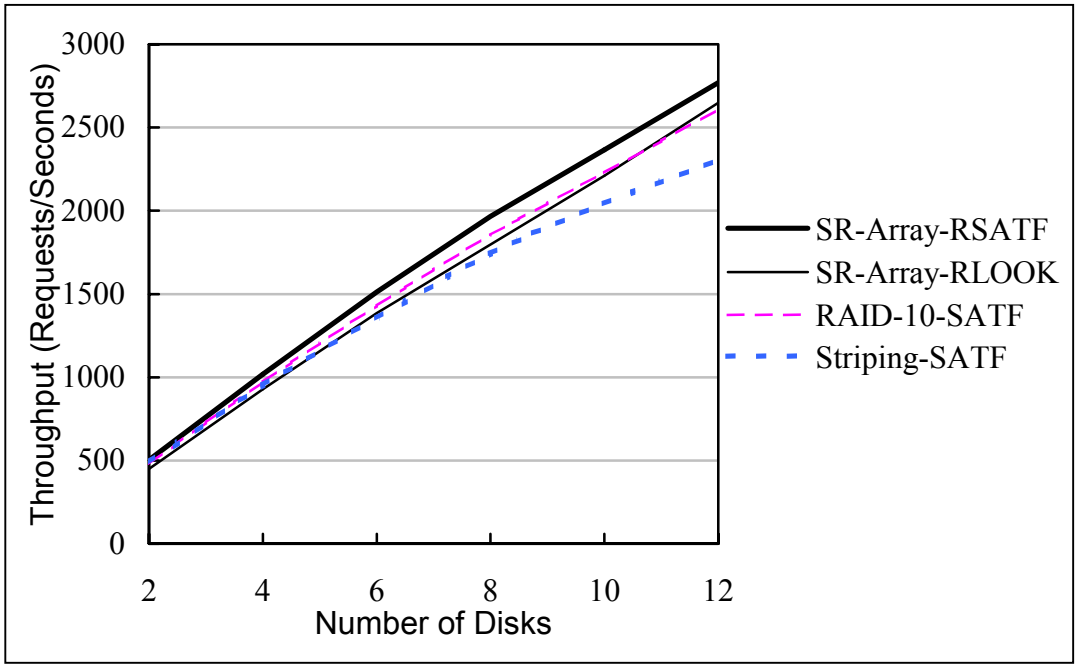


(b)

Figure 4.6: Validation of the RLOOK scheduling model for SR-Array: throughput as a function of the degree of rotational replication in 6-, 8- and 12-disk SR-Array configurations. The workloads are single block random read request with (a) 8 and (b) 32 requests outstanding. Each figure includes measured curves and model curves based on Equation (2.44) with $p = 0$ for a read-only workload.



(a)



(b)

Figure 4.7: Comparison of read throughput on different disk array configurations: throughput as a function of the number of disk in disk arrays and scheduling policies: SR-Array with RSATF- and RLOOK-scheduling, RAID-10 with SATF-scheduling and striping with SATF-scheduling. The workloads are random single block read requests with (a) 8 and (b) 32 outstanding requests.

using different disk array configuration strategies with the same number of disks. The two SR-Array curves represent the best throughput for a given scheduling policy (RSATF or RLOOK) among possible SR-Array configurations using the same number of disks.

Striping has the worse performance even with the rotational position sensitive SATF scheduling policy. RAID-10 and SR-Array have similar read throughput results. (The difference is less than 5%.) We can also see that RLOOK is not as good as RSATF scheduling for SR-Array. The difference is small because both scheduling policies consider rotational positions. RSATF is a little better because it also considers the minimum of the overall latency when scheduling requests. The experiments also validate equation (2.45) and its explanation in Section 2.4.3 that the throughput of SR-Array is close to being a linear function of the number of disks when Q is large.

We have seen that striped-mirroring configurations, such as RAID-10, with SATF scheduling have roughly the same read throughput as that of SR-Array. We now consider write throughput. Because we use a throughput micro-benchmark that has no idle time, we do not consider propagating replicas in the background. In the experiments, all writes to the corresponding replicas must be served before each request is considered complete. This is an conservative evaluation of write throughput for SR-Array.

Figure 4.8 shows the write throughput with 32 outstanding write requests. Unlike the read-only workload, RAID-10 performs clearly the worst. On this 100% write workload, striping, a strategy without replication, clearly wins. Because striping is also a special case of SR-Array, the best SR-Array is striping regardless whether SATF or LOOK scheduling is used. (SATF is 30% better than LOOK in this case.) Although both SR-Array with 2-way rotational replication and RAID-10 have 2 replicas for each piece of data, the former performs 20% better than the latter. This is because the

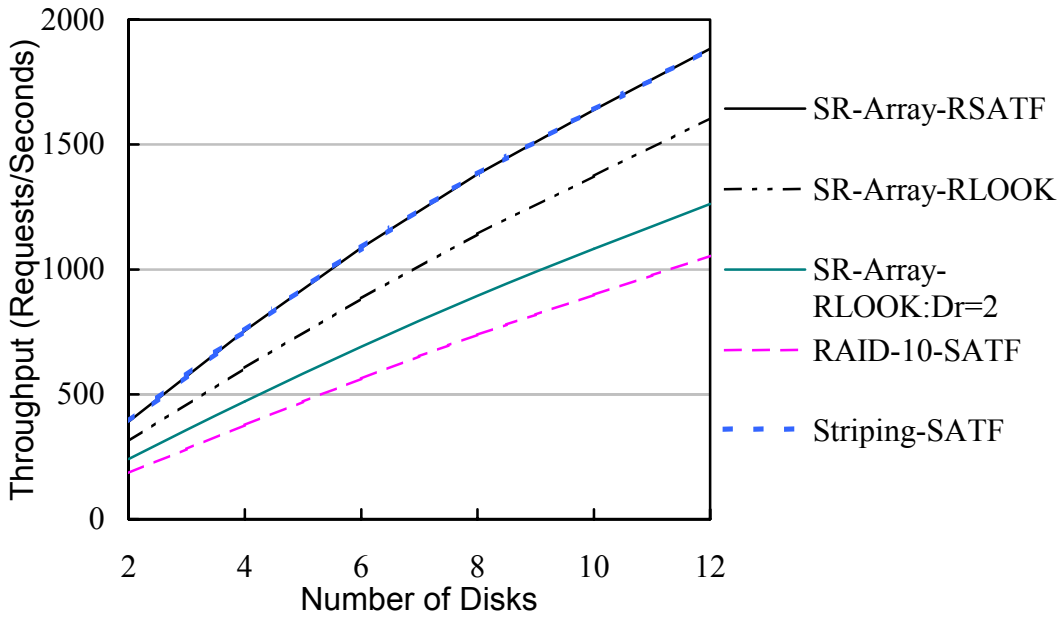
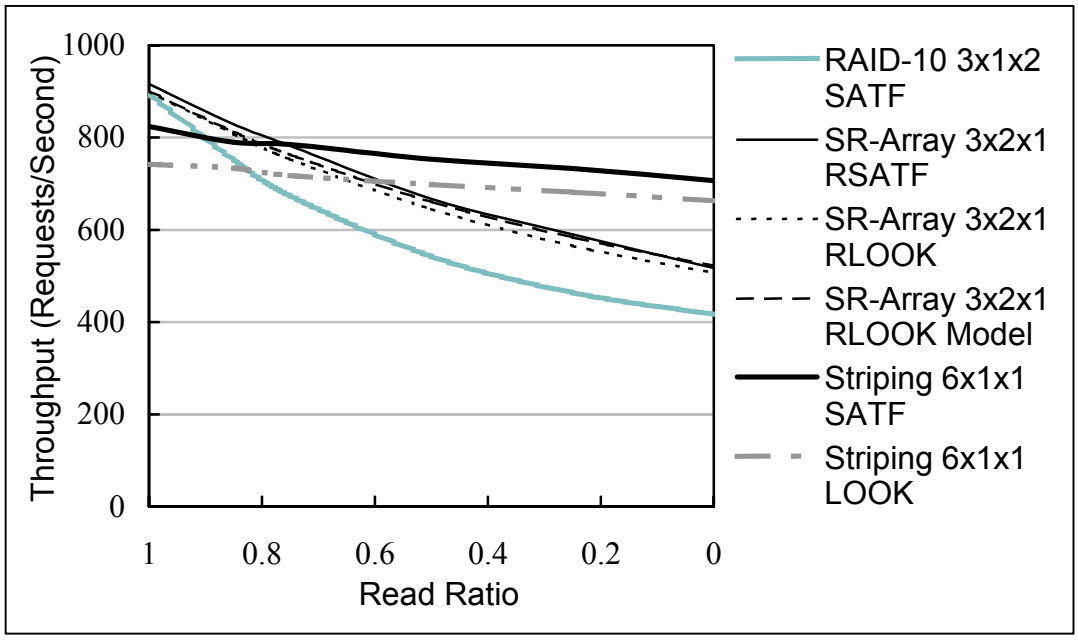


Figure 4.8: Comparison of write throughput on different disk array configurations: throughput as a function of the number of disks in disk arrays and scheduling policies: SR-Array with RSATF- and RLOOK-scheduling, RAID-10 with SATF-scheduling and striping with SATF-scheduling. The workloads are random single block write requests with 32 outstanding requests.

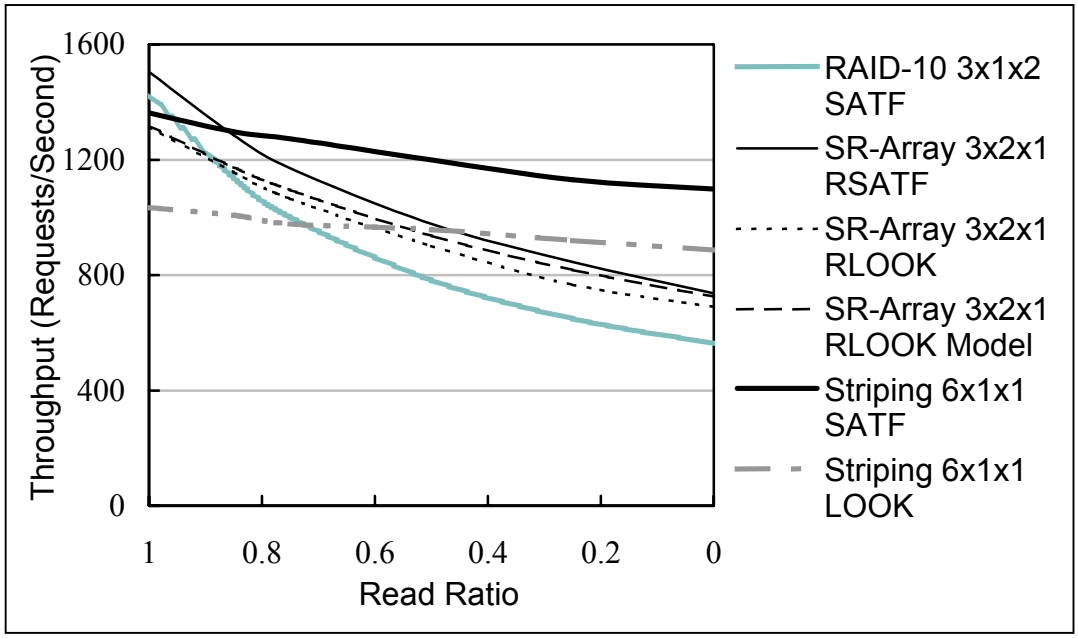
SR-Array configurations require a single seek followed by writing all the replicas on nearby tracks for each write request, while the striped mirror configurations (such as RAID-10) need to perform multiple seeks on multiple disks and therefore spend more time on arm movement, even though SATF scheduling attempts to reduce the latency between requests.

In order to study the effect of writes more closely, we vary the read/write ratio in the workload for different configurations of disk arrays with 6 disks. Specifically, we study SR-Array with 2-way rotational replication ($3\times2\times1$) and RAID-10 ($3\times1\times2$), both of which have 2 replicas for each piece of data. We also examine striping ($6\times1\times1$) with SATF and LOOK scheduling for comparison.

Figure 4.9 shows the results for queue lengths 8 and 32. We can draw several



(a) $Q = 8$



(b) $Q = 32$

Figure 4.9: Comparison of throughput of 6-disk array configurations: throughput as a function of the read/write ratio: RAID-10 with SATF; $3 \times 2 \times 1$ SR-Array with RSATF, RLOOK and RLOOK model based on Equation (2.44); and striping with SATF and LOOK. The workloads are random single block requests with (a) 8 and (b) 32 outstanding requests.

conclusions. First, RLOOK and RSATF for $3 \times 2 \times 1$ SR-Array have similar performance for all read/write ratios and RSATF is always slightly better. Second, our RLOOK model, Equation (2.44), approximates measured RLOOK results well for all read/write ratios and both queue lengths. Third, the $3 \times 2 \times 1$ SR-Array curve intersects with the curve corresponding to LOOK-based striping at an x-axis value between 50% and 60%, which again confirms a conclusion that can be drawn based on a model given earlier: when the number of reads equals that of writes, the benefit derived from rotational replication by reads is roughly equal to the cost paid on foreground replica propagation for writes. The point of intersection is higher than 50% because 6-way striping has better seek time reduction than the $3 \times 2 \times 1$ SR-Array (with 3-way striping). Furthermore, the intersection in Figure 4.9(b) is closer to 50% than that of Figure 4.9(a). This is because the seek time reduction becomes less significant with a larger queue of outstanding requests in Figure 4.9(b). Fourth, a $3 \times 1 \times 2$ RAID-10 is consistently worse than an $3 \times 2 \times 1$ SR-Array and the gap widens to more than 20% for workloads with more writes. We also note that the striping performance drops slightly with more writes. This is because on average, a write request experiences approximately 700 microseconds more than a comparable read request.

4.2 Study on Trace Workloads

In the last section, we have used micro-benchmarks to evaluate SR-Array performance and its mathematical model. Now we use disk traces gathered from real systems to study different configurations of disk arrays. The I/O access patterns in these traces are more complicated than those we can model accurately in the similar microbenchmarks. Nevertheless, we hope to see that the relative performance rela-

	Cello base	Cello disk 6	TPC-C
Data size	8.4 GB	1.3 GB	9.0 GB
I/Os	1,717,483	1,545,341	3,598,422
Duration	1 week	1 week	2 hours
Avg. I/O rate	2.84/s	2.56/s	500/s
Reads	55.2%	35.8%	54.8%
Async. writes	18.9%	16.1%	0
Seek locality (L)	4.14	16.67	1.04
Read after write (1 hour)	4.15%	3.8%	14.8%

Table 4.2: Trace characteristics. The "seek locality" row is calculated as the ratio between the average of random seek distances on that disk and the average seek distance observed in the trace. Section 2.1.4 gives more details on this ratio. The "read after write (1 hour)" row lists the percentage of I/O operations that are reads that occur less than one hour after writing the same data.

tionships among different configuration are still similar to what we have seen before.

We use two traces. Cello is a two month trace taken on a server running at HP Labs [36] in 1992. The server has eight disks and is used for running simulations, compilations, editing, and reading mail and news. We use one week's worth of trace data (for the period of 5/30/92 through 6/6/92). TPC-C is a disk trace (collected on 5/03/94) of an unaudited run of the Client/Server TPC-C benchmark running at approximately 1150 tpmC on a 100 Warehouse database. These traces have been used in measuring disk performances in several previous studies [52, 36].

4.2.1 Logical Data Sets

The 9.1 GB Seagate disks that we use in our experimental platform are much larger and faster than any of the original disks used in the traces; therefore, we do not map the original smaller disks one-to-one onto our much larger disks. Instead, we group the original data into three *logical data sets* and study how to place each data set in a disk array made of new disks.

The first data set consists of all the data from Cello disks with the exception of disk 6, which houses "/usr/spool/news". We merge these separate Cello disk traces based on time stamps to form a single large trace. The data from different disks are concatenated to form a single data set. We refer to this workload as "Cello base" in our study. The second data set consists solely of Cello disk 6. This disk houses the news directory; it exhibits access patterns that are different from the rest of the Cello disks and accounts for 47% of the total accesses. We refer to this workload as "Cello disk 6". The third data set consists of data from 31 original disks of the TPC-C trace. We merge these traces to form a single large trace; and we concatenate these disks to form a single data set as well. We refer to this workload as "TPC-C".

Table 4.2 lists the key characteristics of the trace data sets. We see that the "Cello base" and "Cello disk 6" workloads average fewer than three requests per second. Both workloads are light and contain ample idle time, although at times there are bursty periods. In contrast, "TPC-C" is a heavy workload averaging 500 requests per second.

We also record the amount of asynchronous writes in the traces. Most of the these asynchronous writes are generated by the file system sync daemon at 30-second intervals. Although response time of asynchronous operations is not important to applications, these I/Os affect the response time of other requests, especially under heavy load when there is no idle time.

The last row reports the fraction of I/Os that are reads to recently written data. Although this ratio is high for TPC-C, it does not rise higher for intervals longer than an hour. Together with the amount of available idle time, this ratio impacts the effectiveness of the delayed write propagation strategy and influences the array configurations.

We play the traces to our simulator by submitting read/write requests according

to their timestamps in the trace. We measure the response time of each request, which is the time elapsed between the submission and completion of the request. We use the average response time for all the requests in the trace to measure the performance of the disk system.

To test our system under different load conditions, we also uniformly scale the rate at which the trace is played based on the timestamp information. For example, when the scaling rate is two, the traced inter-arrival times are halved.

4.2.2 Playing Traces at Original Speed

Playing Cello Traces

As a base, we place the Cello base data set on one Seagate disk and the Cello disk 6 data set on another. Although we have fewer disks in this case than in the original trace, the original speed of the Cello traces is still sufficiently low that we are effectively measuring individual I/O latency most of the time. There is also sufficient idle time to mask the delayed write propagations. Therefore, we are able to apply the read-only latency model (Equation (2.19) and (2.20)) in Section 2.3.2 to configure the SR-Array. When applying the formulas, we also account for the different degree of seek locality (L) in Table 4.2 by replacing S with S/L. We perform replica propagation in the background for all configurations. Although all write operations from the trace are played, we exclude those of asynchronous writes when reporting response time; most of the asynchronous writes are generated by the file system sync daemon at 30-second intervals and their response times are not as important.

Figure 4.10 shows the performance improvement on both Cello workloads as we scale the number of disks under various configurations. The curve labeled as "SR-Array" shows the performance of the best SR-Array configuration for a given number

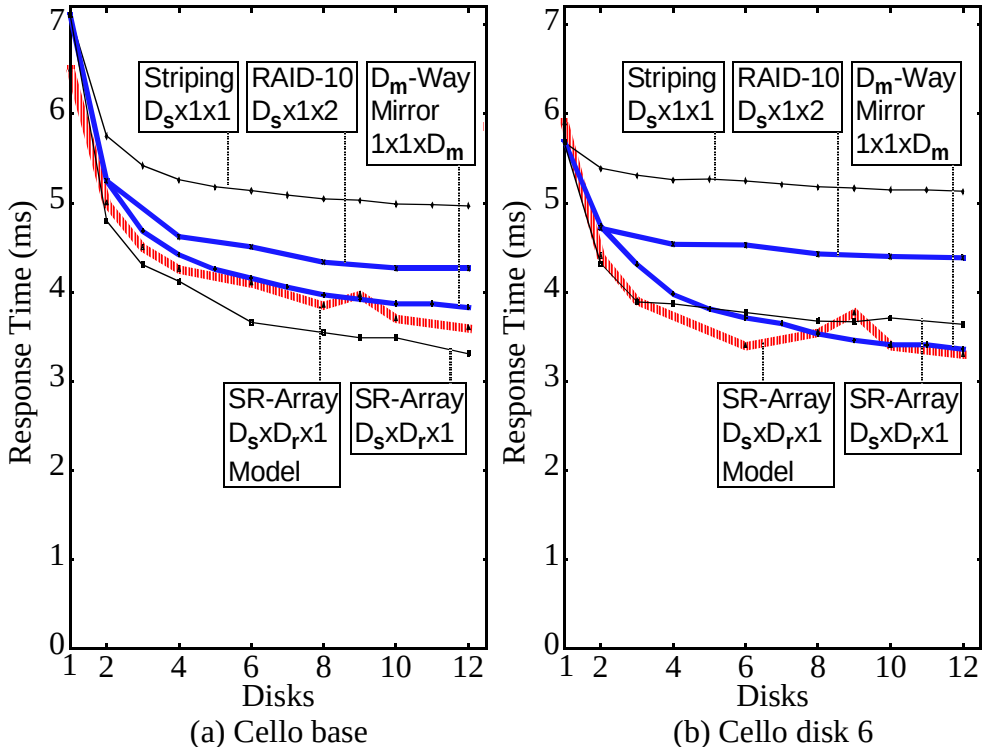


Figure 4.10: Comparison of average I/O response time of the Cello file system workloads on different disk array configurations. The SR-Array uses the RSATF scheduler and the remaining configurations use the SATF scheduler. The two configurations labeled as "RAID-10" and "D_m-way mirror" are reliable configurations and are denoted by thicker curves. This convention is used throughout the rest of the figures in this chapter.

of disks. The SR-Array configuration performs well because it is able to effectively distribute disks to both the seek and the rotational dimensions in a balanced manner. In contrast, the performance of simple striping is poor due to the lack of rotational delay reduction. This effect is more apparent for larger numbers of disks due to the diminishing returns from seek distance reduction. The performance of RAID-10 is intermediate because the two replicas allow for a reduction in the rotational delay to a limited extent. D-way mirroring is the closest competitor to an SR-Array because of its high degree of flexibility in choosing which replica to read. Note that our SATF-based implementations of RAID-10 and D-way mirroring are highly optimized versions, which considers rotational positioning across all disks in the array when

choosing a disk to serve a request with. This optimization is not achievable solely based on internal SATF-like scheduling within one disk.

The figure also shows that the latency model of Section 2.3.2 is a good approximation of the SR-Array performance for light workloads. The anomalies on the model curves are due to the two following practical constraints: (1) D_s and D_r must be integer factors of the total number of disks D, and (2) our implementation restricts the largest degree of rotational replication to six. This restriction is due to the difficulty of propagating more copies within a single rotation, as rotational replicas are placed on different tracks and a track switch costs about 900 μs. Due to these constraints, for example, the largest practical value of D_r for $D = 9$ is only three, much smaller than the non-integer solution of Equation (2.20) (5.8 for Cello base and 11.6 for Cello disk 6).

While the Cello base data set consumes an entire Seagate disk, the Cello disk 6 data set only occupies about 15% of the space on a single Seagate disk; so the maximum seek delay of Cello disk 6 is small to begin with for all configurations. Consequently, a larger D_r for an SR-Array is desirable as we increase the number of disks. With these large D_r values, however, the practical constraints enumerated above start to take effect. Coupled with the fact that seek time is no longer a linear function of seek distance at such short seek distances, this explains the slightly more pronounced anomalies of the SR-Array performance with a large number of disks on the Cello disk 6 workload.

Figure 4.11 compares the performance of other possible SR-Array configurations with that of the configuration chosen by the model. For example, when the number of disks is six, the model recommends a configuration of $D_s \times D_r = 2 \times 3$ for Cello base. The three alternative configurations are 1×6, 3×2, and 6×1. The figure shows that the model is largely successful at finding good SR-Array configurations.

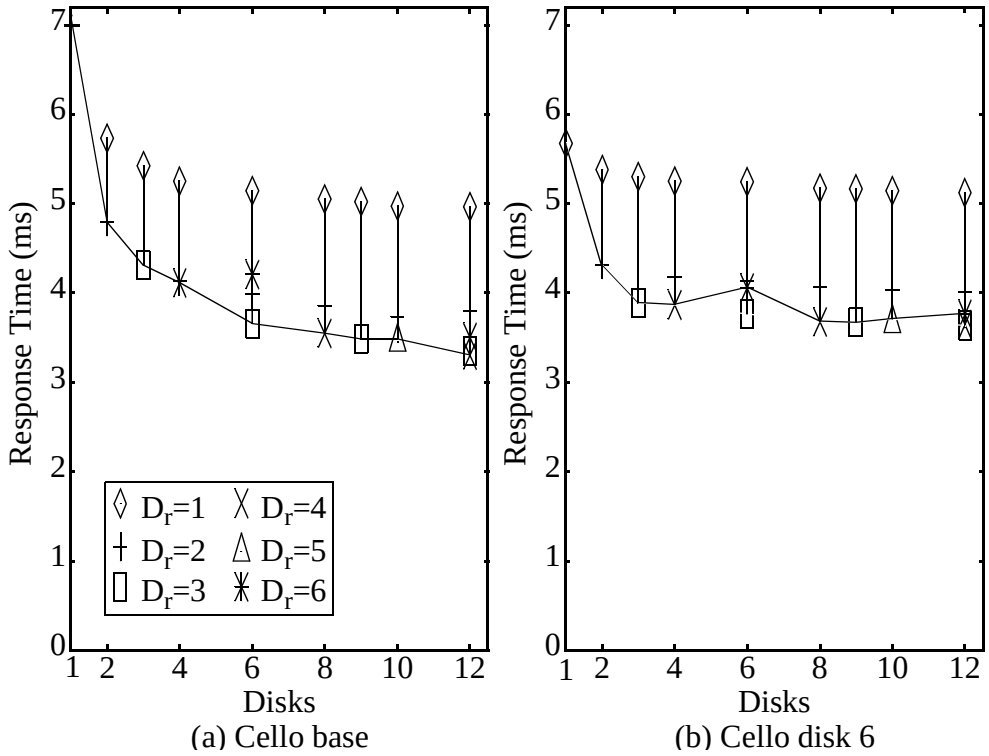


Figure 4.11: Configurations of the SR-Array for the two workloads of Figure 4.10. The curves show the performance of the SR-Array configuration recommended by the model of Equation (2.20). Each point symbol in the graph shows the performance of an alternative SR-Array configuration with a different number of rotational replicas (D_r).

For example, on Cello base, with six disks, the SR-Array is 1.23 times as fast as a highly optimized RAID-10, 1.42 times as fast as a striped system, and 1.94 times as fast as the single disk base case.

Playing TPC-C Trace

Although a single new Seagate disk can accommodate the entire TPC-C data set in terms of capacity, it cannot support the data rate of the original trace. Indeed, the original traced workload is striped to more than 30 disks so that it can sustain such a high request rate. Only a fraction of the space is used on each of the original traced disk. Our experiment first combines the traces from all disks. Because the disks

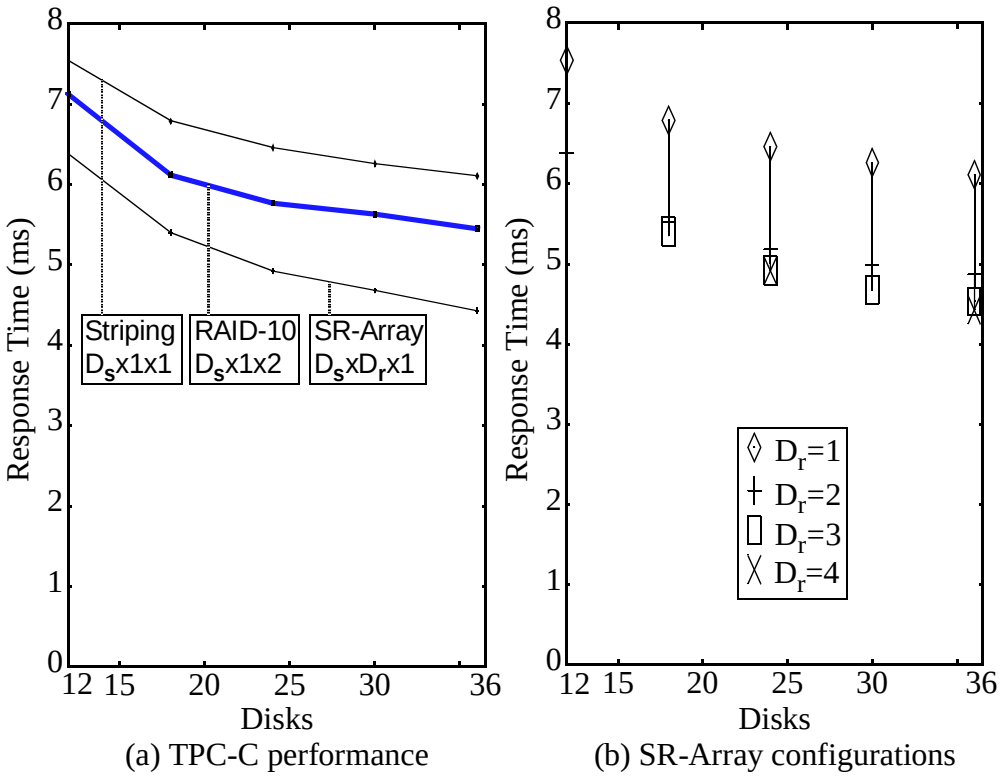


Figure 4.12: Average I/O response time of the TPC-C trace. (a) Comparison of striping, RAID-10, and SR-Array. (b) Comparison of alternative configurations of an SR-Array.

we use are much faster than those original ones, we start with 12 disks for each of the array configurations. Figure 4.12 shows the performance as we scale the number of disks beyond the starting point. The data rate experienced by each disk under this workload is much higher than that under the Cello system. The workload also contains a large fraction of writes so it also stresses delayed write propagation as idle periods are shorter.

Compared to Figure 4.10, two curves are missing from Figure 4.12. One is $D-$way mirroring—it is impossible to support the original data rate while attempting to propagate D replicas for each write. Another missing curve is the latency model—the high data rate renders the latency model inaccurate. The spirit of Figure 4.12, however, is very much similar to that of Figures 4.10 and 4.11: a properly configured

SR-Array is faster than a RAID-10, which is faster than a striped system.

What is more interesting is the fact that striping, the only configuration that does not involve replication, is not the best configuration even under the heavy write traffic exhibited by this workload. There are at least two reasons. First, even under this higher I/O rate, there are still idle periods to mask replica propagations. Second, even without idle periods, there exists a tradeoff between the benefits received from reading the closest replicas and the cost incurred when propagating replicas, as explained in Section 2.2.3; a configuration that can successfully exploit this tradeoff excels. For example, with 36 disks, a $9 \times 4 \times 1$ SR-Array is 1.23 times as fast as a $18 \times 1 \times 2$ RAID-10, and 1.39 times as fast as a $36 \times 1 \times 1$ striped system.

4.2.3 Playing Traces at Accelerated Speed

Although the original I/O rate of TPC-C is higher than that of the Cello traces, it does not stress the 12-disk arrays discussed in the last section. We now raise the I/O rates to stress the various configurations. For example, when the "scale rate" is two, we halve the inter-arrival time of requests.

Before we compare the different array configurations, we first consider the impact of different scheduling policies. Figure 4.13 evaluates four schedulers: LOOK and SATF for striping, and RLOOK and RSATF for an SR-Array. Given a particular request arrival rate, the gap between RLOOK and RSATF is smaller than that between LOOK and SATF. This is because both RLOOK and RSATF take rotational positioning into consideration. Although it is a well known result that SATF out-performs LOOK [26, 43], we see that SATF alone is not sufficient for addressing rotational delays if the array is mis-configured to begin with. For example, under the Cello base workload, a $2 \times 3 \times 1$ SR-Array significantly outperforms a $6 \times 1 \times 1$ striped

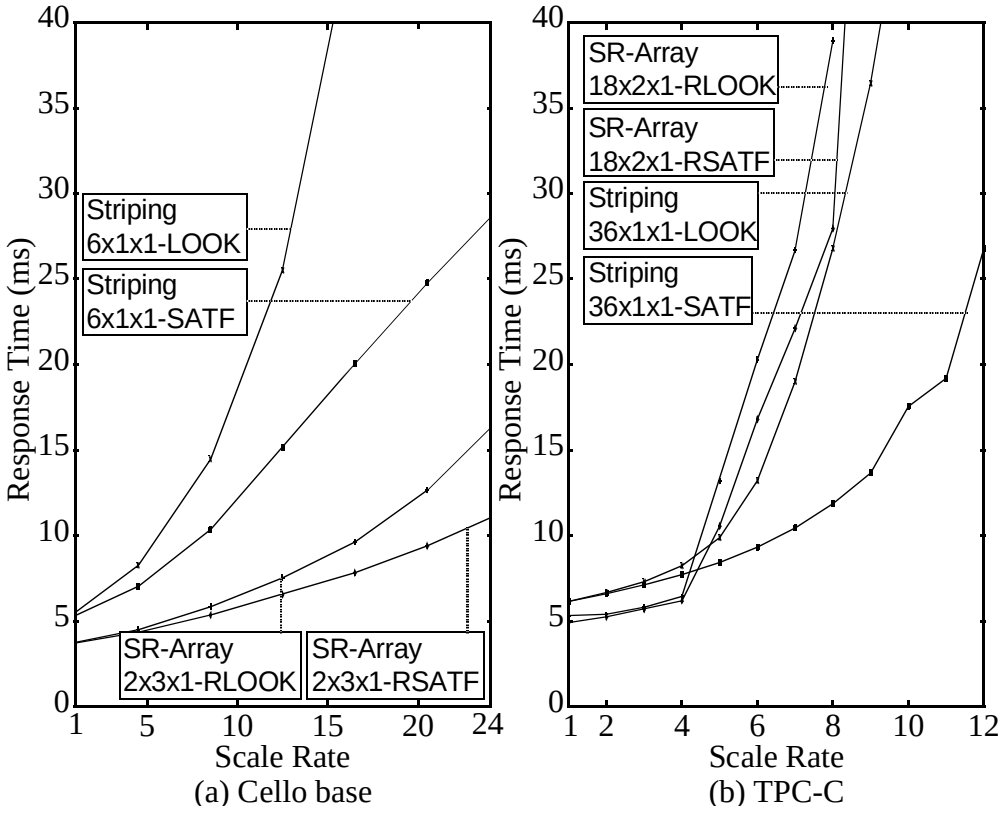


Figure 4.13: Comparison of scheduling policies with accelerated traces for different configurations. We use (a) 6 disks for Cello base, and (b) 36 disks for TPC-C.

system even if the former only uses an RLOOK scheduler while the latter uses an SATF scheduler. In the rest of this sub-sections, unless specified otherwise, we use the RSATF scheduler for SR-Arrays and the SATF scheduler for other configurations.

Figure 4.14 shows the performance of the various configurations while we fix the number of disks for each workload and vary the rate at which the traces are played. Under the Cello base workload (shown in Figure 4.14(a)), the 6-way mirroring and the 1×6 SR-Array deliver the lowest sustainable rates. These configurations make the largest number of replicas, which makes it difficult to mask the replica propagation under high request rates. The 6-way mirroring is better than the 1×6 SR-Array, because it can afford the flexibility of choosing any disk to service a request, so it can

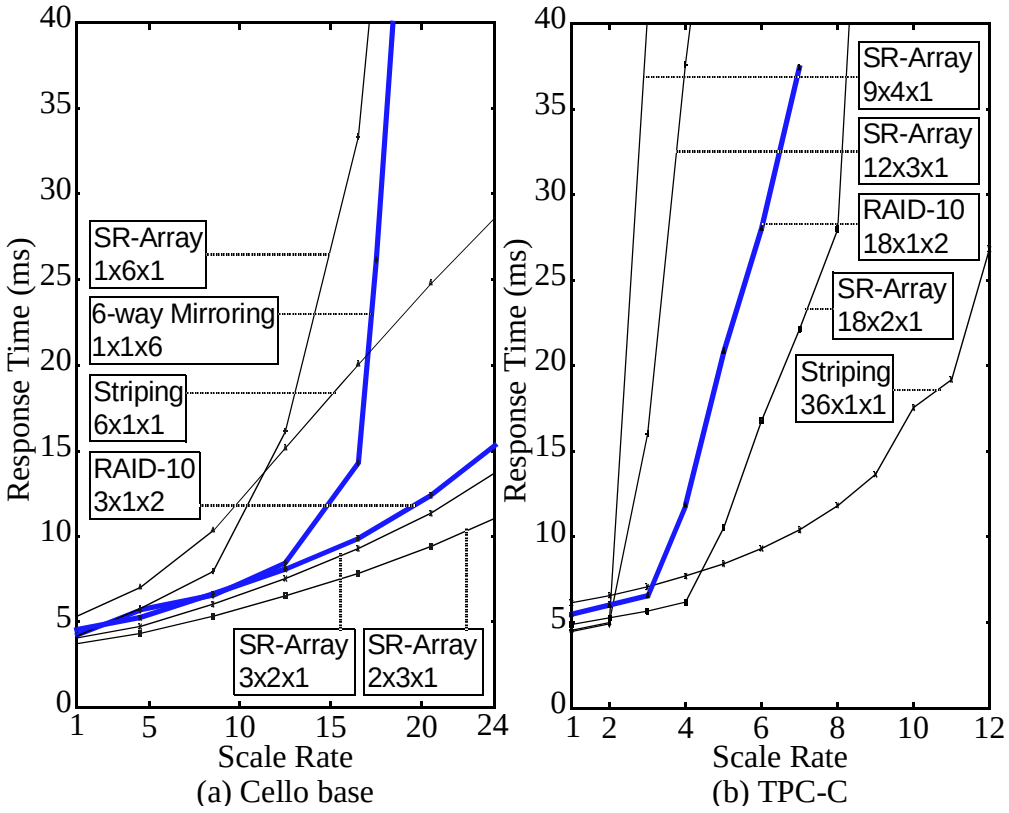


Figure 4.14: Comparison of I/O response time on different configurations with accelerated traces. We use (a) 6 disks for Cello base, and (b) 36 disks for TPC-C.

perform better load balancing. The 2×3 SR-Array is best for all the arrival rates that we have examined; this is because the benefits derived from the extra rotational replicas outweigh the cost. If we demand an average response time no greater than 15 m*s*, the $2 \times 3 \times 1$ SR-Array can support a request rate that is 1.3 times that of a $3 \times 1 \times 2$ RAID-10 and 2.6 times that of a $6 \times 1 \times 1$ striped system.

The situation is different for the TPC-C workload (shown in Figure 4.14(b)). Under the original trace playing rate, the $9 \times 4 \times 1$ SR-Array is best. As we raise the request arrival rate, we must successively reduce the degree of replication; so the role of the best configuration passes to the $12 \times 3 \times 1$, $18 \times 1 \times 2$, $18 \times 2 \times 1$, and finally, $36 \times 1 \times 1$ configurations, in that order. If we again demand an average response time

no greater than 15 ms, the $36 \times 1 \times 1$ configuration can support a request rate that is 1.3 times that of a $18 \times 2 \times 1$ configuration and 2.1 times that of a $18 \times 1 \times 2$ RAID-10 configuration.

4.3 More Disks v.s. More Memory Caching

We have seen that it is possible to achieve significant performance improvement by scaling the number of disks. We now compare this approach against one alternative: simply adding volatile memory as cache. We assume that the memory cache performs LRU replacement. Synchronous writes are forced to disks in both alternatives. In the following discussion, we assume that the price per MB ratio between memory and disk is M. At the time that we started this research, 256 MB of memory costs $300, an 18 GB SCSI disk costs $400, and these prices give an M value of 57. At present, 1 GB of memory costs $300, and a 72GB SCSI disk costs $400. The similar improvements in the capacity to cost ratios of memory and disk result in an M value that is roughly stable.

Note that we are not necessarily advocating the use of a fast disk array over a large memory cache for *all* workloads. We merely show that for *some* workloads, a fast disk array can be a more cost-effective solution.

Figure 4.15(a) examines the impact of memory caching on the Cello base workload. At the trace scale rate of one, we need to cache an additional 1.5%, or 126 MB, of the file system in memory to achieve the same performance improvement of doubling the number of disks; and we need to cache 4%, or 336 MB, of the file system to reach the performance of a four-disk SR-Array. M needs to be less than 67 and 75 respectively in order for memory caching to be more cost effective, which it was at the time this research started.

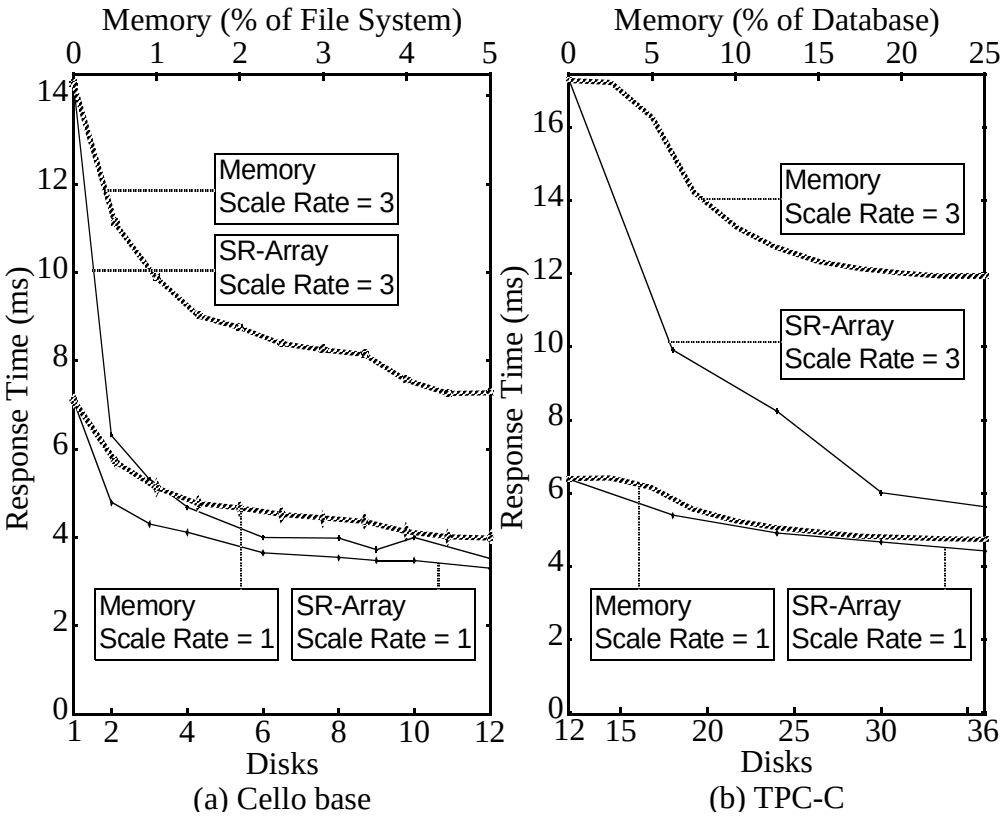


Figure 4.15: Comparison of the effects of memory caching and scaling the number of disks. The two SR-Array curves show the performance improvement achieved by scaling the number of disks (bottom x-axis) and they correspond to playing the traces at the original speed and three times the original speed. The two Memory curves show the performance improvement achieved by scaling the amount of memory caching (top x-axis).

At the trace scale rate of three, using similar reasoning, we can conclude that M needs to be less than 20 in order for memory caching to be more cost effective than doubling the number of disks. Beyond this budget, at this I/O rate, the diminishing locality and the need to flush writes to disks make the addition of memory less attractive. The addition of disks, however, speeds up all I/O operations, albeit at a diminishing rate.

Figure 4.15(b) examines the impact of memory caching on the TPC-C workload, which has much less locality. We start with a 12-disk SR-Array. At a scale rate of one, M needs to be less than 80 in order for memory caching to be a cost effective

alternative to increasing the number of disks to 18 or 24. Adding memory would be a more attractive alternative for this workload at that time.

At a scale rate of three, M needs to be less than 24 for memory caching to be more cost effective than increasing the number of disks to 18. Beyond this budget, adding memory provides little additional performance gain, while increasing the number of disks from 18 to 36 can provide an additional 1.76 times speedup.

There are reasons that make the use of a fast disk array a better alternative than increasing memory caching. One is when the working set is much larger than that can fit in the amount of memory that one can afford. In this case, a fast disk array can uniformly speed up all requests while a memory cache can only speed up a fraction of the requests. The second is when there are many writes that must be made persistent. A large volatile memory cache, in itself, does not reduce the disk write traffic by much.

4.4 Summary and Related Studies

We can draw several conclusions from this empirical study.

- Our latency models and throughput models of SR-Array developed in Chapter 2 are consistent with the results measured on our experimental platform.

- An SR-Array can provide better performance compared to existing disk array techniques for certain workloads.

- The SR-Array latency model can be applied to light workloads like "Cello" to fairly accurately predict the best SR-Array configuration or configurations whose performance is close to the best.

- The RLOOK and RSATF scheduling policies have similar performance, at least for the workloads that we have studied. This makes RLOOK a good alternative to RSATF because it is simpler to implement.

- For certain workloads, adding more disks to form a fast disk array can be more cost effective than adding more memory as a cache.

- The best disk array configurations are different for different workloads; no single configuration is universally the best.

The tradeoff between storage system capacity and performance has long been recognized. Hou and Patt [22] have performed a simulation study of the tradeoff between mirroring and RAID-5. The HP Ivy project [28] is a simulation study of how a high degree of replication can improve read performance. Our study differs from Ivy in several ways. First, Ivy only explores reducing seek distance and leaves rotational delay unresolved. Second, Ivy only examines mirroring. The third difference is a feature of Ivy that we intend to incorporate into our system in the future: Ivy dynamically chooses the candidate and the degree of replication by observing access patterns.

Another way of trading capacity for better performance is to exploit the temporal locality in the workload by employing a hierarchical storage system: the storage level that is closer to the host uses more physical storage space to store (or cache) a smaller amount of logical data to achieve better performance while the levels that are farther away from the host use less physical storage for a comparable amount of logical data so these levels are slower. One such system is the HP AutoRAID system, which incorporates both mirroring and RAID-5 into a two-level hierarchy [50]. The mirrored upper level provides faster small writes at the expense of consuming more storage, while the RAID-5 lower level is more frugal in its use of disk space. The primary focus of this system is solving the small write problem of RAID-5. We have

not implemented a hierarchical system in our experimental framework, although the flexible tradeoff between capacity and performance provided by SR-Array makes it a natural candidate as higher levels in a hierarchical storage system. We have taken the tradeoff between capacity and performance a step further by (1) improving latency and throughput of *all* I/O operations, (2) being able to benefit from more than twice the excess capacity while the upper level of AutoRAID is limited to a RAID-1, which is limited to a single strategy of doubling the storage capacity, and (3) providing a means of systematically configuring the extra disks to achieve the best result.

One of our goals of studying the impact of altering array configurations is to understand how to configure a storage system given certain cost/performance specifications. The "attribute-managed storage" project [13] at HP shares this goal, although its focus is at the disk array level as opposed to individual drive level.

Chapter 5

Conclusion

This dissertation proposes a novel way of designing disk arrays that can flexibly and systematically reduce seek time and rotational delay. We have demonstrated, via modeling and experiments with a real implementation, that the SR-array approach can improve disk I/O performance significantly over existing disk array configuration techniques.

We analyze the existing techniques in disk array design, such as striping, mirroring and rotational replication. In particular, we articulate the relationship between the benefit of better read performance as a result of exploiting rotational replicas and the cost of degrading write performance as a result of replica propagation. By combining these existing techniques, we develop a new way of configuring disk arrays called SR-Array that can reduce both seek time and rotational delay in a balanced way.

We present mathematical models for both latency and throughput of SR-Array by considering both workload and disk drive characteristics. Our study focuses on the workloads that can occur frequently in some office file systems and database transaction systems such as those mainly involving large working sets that do not fit in main memory, small I/O sizes of a few sectors, and poor locality. Moreover, we derive

close-form results on how to configure an SR-Array to obtain the best performance. This gives us insights into how capacity can be best traded for performance. By using D times more disks in an SR-Array, the optimal configuration can improve the non-overhead part of latency by a factor of up to $\sqrt{D}$. The optimal configuration can improve the throughput by a factor of D under our new RLOOK scheduling policy.

Our theoretical results are confirmed by empirical studies to be accurate and very helpful in deriving the best SR-Array configurations for a given number of disk drives. We report that SR-Array compares favorably in terms of performance with other disk array configuration strategies for these workloads with no single configuration being the best for all workloads. With the ability of improving performance by adding extra disk drives, SR-Array can be a more cost effective way of improving performance than adding memory caching for certain workloads when the relative cost ratio between disk drives and memory reach a low enough threshold.

It is difficult to obtain the above empirical results without our flexible experimental framework designed for studying a variety of disk array configuration strategies. Our framework can operate in either a disk driver mode, which drives real disks in a real implementation, or in a simulation mode, which allows us to perform accurate and fast performance studies for various workloads and array configurations. The framework not only helps us confirm that SR-Array is theoretically effective but also makes us believe that it is practically feasible.

An important part of the framework is an accurate disk head position tracking mechanism, which is one of the first reported efforts that we are aware of to accomplish rotational position tracking on commercial off-the-shelf SCSI disk drives without any special hardware support. This mechanism is crucial in supporting rotational replication on the same or neighboring tracks, which is required in implementing the SR-Array configuration family. In addition to supporting the disk array studies in

this thesis, the head tracking mechanism and the framework can help implementing the ideas in other disk- or disk array-related studies[48, 29, 55].

Chapter 6

Future Directions

6.1 A "Continuous Replication" Disk Array

In this dissertation, we have examined several ways of systematically trading disk capacity for improved performance. These strategies, however, treat all data blocks in a uniform way, such as employing a single replication factor for all blocks. This approach simplifies the design and implementation of our prototype, but it may not be the best way of utilizing the available physical capacity. A better approach may be to treat different data blocks differently based on their access patterns so that, for example, more frequently read blocks would enjoy a higher degree of replication and/or are placed on "faster" disks. In a way, in addition to being an extension of the SR-Array, this approach is also an extension of the AutoRAID approach, which supports only two replication factors (1× in RAID-5, or 2× in RAID-1) and allows data to be dynamically moved between these two levels. By allowing greater flexibility in the degree of replication and in ways different blocks are replicated, we may be able to support a smooth and variable continuum of cost/performance and reliability specifications in a single disk array, at a data granularity that is as fine as a single

block, and at a time granularity that is as short as a few I/O operations.

6.2 Altering Disk Geometry

As we have discussed earlier, a primary motivation behind this thesis is the phenomenon that as disk capacity rapidly increases, disks are becoming increasingly unbalanced in the relationship between capacity and latency. The techniques of using extra disks to reduce latency, as is done in the SR-Array, are in effect emulating faster disks using existing slow disks. A more direct and potentially more cost-effective approach to balancing capacity with lower latency is to build "better" drives to start with so that we do not have to resort to either replication or discarding space.

By using only a portion of the disk, striping is in effect emulating a smaller disk using a large disk. A more direct approach is to simply reduce the disk diameter. Reducing disk diameter directly improves both maximum seek distance and rotational speed. (Without exceeding allowable data rates, which appear to be the bottleneck of current disk drive technology, a reduction of diameter by a certain factor can be matched by an increase of the same factor in rotational speed.) There are also secondary benefits such as the fact that a smaller arm can be made more rigid and is easier to control.

Unlike the use of extra disks in the SR-Array, however, reducing diameter alone does not allow us to flexibly adjust the amount of resources devoted to the improvement of seek and rotational dimensions. It is, however, possible to adjust the amount of improvement that each dimension receives by adjusting the inner and outer radius of the platters while maintaining constant area. A larger inner radius results in fewer tracks and better seek characteristics at the expense of slower rotational speed, while a smaller inner radius has the opposite effect.

The techniques that we have examined so far all involve increasing the number of spindles for the same amount of usable capacity. We now speculate on the most direct and potentially most cost-effective way of building a more "balanced" drive: increasing the number of heads per surface. Each head is mounted on its own independent arm. The arms are spaced in such a way that they cannot collide. Although such a drive can amortize the cost of the components such as the spindle and the power supply over a larger number of heads, due to the data rate constraint, each arm in such a drive is likely to need its own data channel. To provide reliability and to allow more flexible cost/performance-driven configurations, we can complement these more expensive drives with cheaper conventional drives in a disk array.

This approach also affords us the largest degree of freedom in terms of choosing which one of the seek and rotational dimensions we would like to devote more resources to. At one extreme, each head can be dedicated to I/Os destined to a subset of the cylinders. This is analogous to striping in that it improves seek performance without affecting rotational delay. At the other extreme, each head may seek to any cylinder and we always choose the head that is rotationally closest to the target sector. This improves rotational delay without affecting seek delay. It is up to the system software to choose a strategy that strikes a balance between these two extremes by judiciously scheduling the independent heads.

The technique of using extra disks, as is done in the SR-Array, is the least cost-effective because replication and discarding space are both wasteful. Maintaining replicas also introduces complexity. Reducing platter diameter is less wasteful because no bits are wasted; but this technique results in an increase of both the number of heads and spindles for the same amount of capacity, so it is more costly than the third technique of just adding heads. We note that there does not exist a single "perfect" drive that has the "right" diameter and the "right" number of heads per surface.

Instead, there may exist one "right" drive for every cost/performance specification.

6.3 Re-distributing Responsibilities among Storage System Components

Traditionally, storage system designers have strived to *virtualize* the storage devices so that they provide a uniform abstract interface of a single conventional disk. Consistent with this tradition, an important goal of the disk array prototype developed as part of this thesis is its transparency: our prototype disk array supports existing file systems and applications ("from below") and existing off-the-shelf disk drives ("from above").

Although this division of responsibility makes the system simple and modular, it may not necessarily be the best for performance. As we have discussed above, one possible extension of the SR-Array approach is to exploit application-specific information or access patterns. Instead of attempting to deduce this information transparently at the storage level, however, one may wish to export a new storage system interface that allows sophisticated applications (such as databases and web servers) to make the most of an underlying storage system that is far more sophisticated than a single drive. For example, a database may choose to place its logs on a storage subsystem that is configured quite differently from one storing some of its relational tables. Of course, one must exercise care to provide the right level of abstraction so we limit the complexity exposed.

In addition to exploring the interface between applications and storage systems, in a similar vein, in light of the sophisticated storage systems that we have examined, one may also wish to re-examine the interface between the storage systems and the underlying disk drives. One example is the disk head rotational position tracking

mechanism that we have built entirely in software. Although we have achieved a remarkable degree of accuracy, it is a source of considerable complexity. An even more accurate and simpler system might have been achieved had we had firmware support. Again, we must exercise care to ensure that the firmware support is generic and flexible enough to support a wide range of storage functionalities (such as the various array scheduling algorithms) without exposing too much low-level complexity.

Bibliography

[1] Aboutabl, M., Agrawala, A., and Decotignie, J.-D. Temporally Determinate Disk Access: An Experimental Approach (Extended Abstract). In *Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Madison, Wisconsin, June 1998), pp. 280–281.

[2] Adaptec. Windows ASPI Package Support, http://www.adaptec.com/worldwide/support/suppdetail.html?prodkey=ASPI-4.70.

[3] A.R.Calderbank, Jr., E., and L.Flatto. Optimum head separation in a disk system with two read/write heads. *Journal of the ACM 31*, 4 (October 1984), 826–838.

[4] Bitton, D., and Gray, J. Disk Shadowing. In *Proc. of the Fourteenth International Conference on Very Large Data Bases* (Los Angeles, CA, August 1988), Morgan Kaufmann, pp. 331–338.

[5] Borr, A. Transaction Monitoring in Encompass: Reliable Distributed Transaction Processing. In *Proc. of the Seventh International Conference on Very Large Data Bases* (Cannes, France, September 1981), IEEE Press, pp. 155–165.

[6] Chao, C., English, R., Jacobson, D., Stepanov, A., and Wilkes, J. Mime: a High Performance Parallel Storage Device with Strong Recovery Guarantees. Tech. Rep. HPL-CSP-92-9 rev 1, Hewlett-Packard Company, Palo Alto, CA, March 1992.

[7] Coffman, E. G., and Hofri, M. On the expected performance of scanning disks. *SIAM Journal on Computing 11*, 1 (February 1982), 60–70.

[8] Coffman, E. G., Klimko, L., and Ryan, B. Analysis of scanning policies for reducing disk seek times. *SIAM Journal on Computing 1*, 3 (September 1972), 269–279.

[9] Dishon, Y., and Lui, T. S. Disk Dual Copy Methods and Their Performance. In *Proc. of Eighteenth International Symposium on Fault-Tolerant Computing (FTCS-18)* (Tokyo, Japan, 1988), IEEE CS Press, pp. 314–318.

[10] Frank, H. Analysis and optimization of disk storage devices for time-sharing systems. *Journal of the ACM 16*, 4 (October 1969), 602–620.

[11] G.E.Houtekamer, A.A.Vlasblom, and A.J.Breimer. A model of the i/o subsystem of the phillips p4500 minicomputer. *Microprocessing and Microprogramming 18* (1986), 491–498.

[12] Geist, R., and Daniel, S. A continuum of disk scheduling algorithms. *ACM Transaction on Computer Systems 5*, 1 (February 1987), 77–92.

[13] Golding, R., Shriver, E., Sullivan, T., and Wilkes, J. Attribute-managed Storage. In *Workshop on Modeling and Specification of I/O* (San Antonio, TX, October 1995).

[14] Gotlieb, C. C., and MacEwen, G. H. Performance of movable-head disk storage systems. *Journal of the ACM 20*, 4 (October 1973), 604–623.

[15] Growchowski, E. Emerging Trends in Data Storage on Magnetic Hard Disk Drives. In *Datatech* (September 1998), ICG Publishing, pp. 11–16.

[16] GUM, B. B. *Algorithms and Arrays for Computing on Massive Data Sets*. PhD thesis, Department of Computer Science, Princeton University, November 2001.

[17] Hennessy, J. L., and Patterson, D. A. *Computer Architecture: a quantitative approach*. Morgan Kaufmann Publishers, Inc., 1990.

[18] Hewlett-Packard Company, Palo Alto, CA. *HP C2490A 3.5-inch SCSI-2 Disk Drives Technical Reference Manual (HP Part No. 5961-4359), 3rd edition*. Boise, Idaho, 1993.

[19] Hofri, M. Disk scheduling: Fcfs vs. sstf revisited. *Communications of the ACM 23*, 11 (November 1980), 645–653.

[20] Hospodor, A. Mechanical access time calculation. *Advances in Information Storage Systems 6* (1995), 313–336.

[21] Hospodor, A. D., and Hoagland, A. S. The changing nature of disk controllers. *Proceedings of IEEE 81*, 4 (April 1993), 586–594.

[22] Hou, R., and Patt, Y. N. Trading Disk Capacity for Performance. In *Proc. of the Second International Symposium on High Performance Distributed Computing* (Spokane, WA, July 1993), pp. 263–270.

[23] Hsiao, H.-I., and DeWitt, D. J. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In *Proc. of the 1990 IEEE International Conference on Data Engineering* (February 1990), pp. 456–465.

[24] Huang, L., and Chiueh, T. Trail: Write Optimized Disk Storage System. http://www.ecsl.cs.sunysb.edu/trail.html.

[25] Intel Server Architecture Lab. Iometer: The I/O Performance Analysis Tool for Servers. http://developer.intel.com/design/servers/devtools/iometer.

[26] Jacobson, D. M., and Wilkes, J. Disk Scheduling Algorithms Based on Rotational Position. Tech. Rep. HPL-CSP-91-7rev1, Hewlett-Packard Company, Palo Alto, CA, February 1991.

[27] King, R. P. Disk arm movement in anticipation of future requests. *ACM Transactions on Computer Systems 8*, 3 (August 1990), 214–229.

[28] Lo, S.-L. Ivy: A Study on Replicating Data for Performance Improvement. Tech. Rep. HPL-CSP-90-48, Hewlett-Packard Company, Palo Alto, CA, December 1990.

[29] Lumb, C., Schindler, J., Ganger, G. R., Riedel, E., and Nagle, D. F. Towards Higher Disk Head Utilization: Extracting "Free" Bandwidth from Busy Disk Drives. In *Proc. of the Fourth Symposium on Operating Systems Design and Implementation* (San Diego, CA, October 2000).

[30] Matloff, N. S. A multiple disk system for both fault tolerance and improved performance. *IEEE Transactions on Reliability R-36*, 2 (June 1987), 199–201.

[31] McKusick, M. K., Joy, W. N., Leffler, S. J., and Fabry, R. S. A fast file system for UNIX. *Computer Systems 2*, 3 (1984), 181–197.

[32] Ng, S. W. Improving disk performance via latency reduction. *IEEE Transactions on Computers 40*, 1 (January 1991), 22–30.

[33] Oney, W. Queueing analysis of the scan policy for moving-head disks. *Journal of the ACM 22*, 3 (July 1975), 397–412.

[34] Orji, C. U., and Solworth, J. A. Doubly Distorted Mirrors. In *Proc. of ACM SIGMOD Conference* (May 1993), pp. 307–316.

[35] Polyzois, C., Bhide, A., and Dias, D. Disk Mirroring with Alternating Deferred Updates. In *Proc. of the Nineteenth International Conference on Very Large Data Bases* (Dublin, Ireland, 1993), Morgan Kaufmann, pp. 604–617.

[36] Ruemmler, C., and Wilkes, J. UNIX Disk Access Patterns. In *Proc. of the Winter 1993 USENIX* (San Diego, CA, Jan. 1993), Usenix Association, pp. 405–420.

[37] Ruemmler, C., and Wilkes, J. An Introduction to Disk Drive Modeling. *IEEE Computer 27*, 3 (March 1994), 17–28.

[38] Savage, S. AFRAID – A Frequently Redundant Array of Independent Disks. In *Proc. of the 1996 Winter USENIX* (January 1996), pp. 27–39.

[39] Sawert, B. *The Programmer's Guide to SCSI*. Addison Wesley Professional, 1998.

[40] Seagate. Cheetah 9LP Family: ST39102LW/LC, ST34502LW/LC, Product Manual, August 1998.

[41] Seagate. Cheetah 18LP Family: ST318203LW/LWV/LC/LCV, ST318233LWV/LCV, ST39103LW/LWV/LC/LCV, ST39133LWV/LCV, Product Manual, February 2000.

[42] Seeger, B. An analysis of schedules for performing multi-page requests. *Information Systems 21*, 5 (July 1996), 387–407.

[43] Seltzer, M., Chen, P., and Ousterhout, J. Disk Scheduling Revisited. In *Proc. of the 1990 Winter USENIX* (Washington, D.C., Jan. 1990), Usenix Association, pp. 313–323.

[44] Teorey, T. J., and Pinkerton, T. B. A comparative analysis of disk scheduling policies. *Communications of the ACM 15*, 3 (March 1972), 177–184.

[45] Teradata Corp. *DBC/1012 Database Computer System Manual Release 2.0*, November 1985.

[46] Transaction Processing Performance Council. *TPC Benchmark C Standard Specification*. Waterside Associates, Fremont, CA, August 1996.

[47] Wang, R. Y., Anderson, T. E., and Patterson, D. A. Virtual Log Based File Systems for a Programmable Disk. Tech. Rep. UCB/CSD 98/1031, University of California at Berkeley, December 1998.

[48] Wang, R. Y., Anderson, T. E., and Patterson, D. A. Virtual Log Based File Systems for a Programmable Disk. In *Proc. of the Third Symposium on Operating Systems Design and Implementation* (New Orleans, LA, February 1999), Operating Systems Review, Special Issue, pp. 29–43.

[49] Wilhelm, N. C. An anomaly in disk scheduling: a comparison of fcfs and sstf seek scheduling using an empirical model for disk accesses. *Communications of the ACM 19*, 1 (January 1976), 13–17.

[50] Wilkes, J., Golding, R., Staelin, C., and Sullivan, T. The HP AutoRAID Hierarchical Storage System. *ACM Transactions on Computer Systems 14*, 1 (February 1996), 108–136.

[51] Worthington, B. L., and Ganger, G. R. Scheduling for modern disk drives and non-random workloads. Technical Report CSE-TR-194-94, Department of Computer Science and Engineering, University of Michigan, March 1994.

[52] Worthington, B. L., Ganger, G. R., and Patt, Y. N. Scheduling Algorithms for Modern Disk Drives and Non-Random Workloads . In *Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Nashville, TN, May 1994), Performance Evaluation Review 22(1), pp. 241–251.

[53] Worthington, B. L., Ganger, G. R., Patt, Y. N., and Wilkes, J. On-Line Extraction of SCSI Disk Drive Parameters. In *Proc. of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Ottawa, Canada, May 1995), Performance Evaluation Review 23(1), pp. 146–156.

[54] Yu, X., Gum, B., Chen, Y., Wang, R. Y., Li, K., Krishnamurthy, A., and Anderson, T. E. Trading capacity for performance in a disk array. In *Proceedings of the 2000 Symposium on Operating Systems Design and Implementation* (San Diego, 2000), USENIX Association, pp. 243–258.

[55] Zhang, C., Yu, X., Krishnamurthy, A., and Wang, R. Y. Configuring and scheduling an eager-writing disk array for a transaction processing workload. In *Proceedings of the First USENIX Conference on File and Storage Technologies* (Montery, California, Jan 2002), pp. 289–304.