ABSTRACT

SCHEDULING POLICIES FOR DISKS AND DISK ARRAYS

by
Chang Liu

Recent rapid advances of magnetic recording technology have enabled substantial increases in disk capacity. There has been less than 10% improvement annually in the random access time to small data blocks on the disk. Such accesses are very common in OLTP applications, which tend to have stringent response time requirements. Scheduling of disk requests is intended to improve their response time, reduce disk service time, and increase disk access bandwidth with respect to the default FCFS scheduling policy.

Shortest Access Time First policy has been shown to outperform other classical disk scheduling policies in numerous studies. Before verifying this conclusion, this dissertation develops an empirical analysis of the SATF policy, and produces a valuable by-product, expressed as $x[m] = m^{-p}$, during the study.

Classical scheduling policies and some well-known variations of the SATF policy are re-evaluated, and three extensions are proposed. The performance evaluation uses self-developed simulators containing detailed disk information. The simulators, driven with both synthetic and trace workloads, report the measurements of requests, such as the mean and the 95^{th} percentile of the response times, as well as the measurements of the system, such as the maximum throughput.

A comprehensive arrangement of routing and scheduling schemes is presented for mirrored disk systems, or RAID1. The performance evaluation is based on a two-dimensional configuration classification: independent queues (i.e. a router sends the requests to one of the disks as soon as these requests arrive) versus a shared queue (i.e. the requests are held in a common queue at the router and are scheduled to be served); normal data layout versus transposed data layout (i.e. the data stored on the

inner cylinders of one disk is duplicated on the outer cylinders of the mirrored disk). The availability of a non-volatile storage or NVS, which allows the processing of write requests to be deferred, is also investigated. Finally, various strategies of mirrored disk declustering are compared against the basic disk mirroring. Their competence of load balancing and their reliability are examined in both normal mode and degraded mode.

SCHEDULING POLICIES FOR DISKS AND DISK ARRAYS

by
Chang Liu

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

Department of Computer Science

May 2005

APPROVAL PAGE

SCHEDULING POLICIES FOR DISKS AND DISK ARRAYS

Chang Liu

Dr. Joseph Y Leung, Dissertation Advisor — Date
Distinguished Professor of Computer Science, NJIT

Dr. Artur Czumaj, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Marvin K Nakayama, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. David Nassimi, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Jian Yang, Committee Member — Date
Assistant Professor of Industrial and Manufacturing Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Chang Liu

Degree: Doctor of Philosophy

Date: May 2005

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2005
- Bachelor of Engineering,
  Tsinghua University, Beijing, China, 2000

Major: Computer Science

Presentations and Publications:

Alexander Thomasian and Chang Liu, "Empirical performance evaluation of the SPTF disk scheduling policy", *submitted to The Computer Journal*, British Computer Society, Aug. 2004.

Alexander Thomasian and Chang Liu, "Performance evaluation of variations of the SATF scheduling policy", *SPECTS'04.*, San Jose, CA, pp. 398-401, Aug. 2004.

Chunqi Han, Alexander Thomasian and Chang Liu, "Affinity Based Routing in Mirrored Disks with Zoning", *SPECTS'04.*, San Jose, CA, pp. 680-687, Aug. 2004.

Alexander Thomasian, Chunqi Han, Gang Fu, and Chang Liu, "A Performance Evaluation Tool for RAID Disk Arrays", *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems 2004.*, pp. 8-17, Sep. 2004.

Alexander Thomasian, Junilda Spirollari, Chang Liu, Chunqi Han, Gang Fu, "Mirrored Disk Scheduling," *SPECTS'03.*, Montreal, Canada, Jul. 2003.

Alexander Thomasian and Chang Liu, "Some new disk scheduling policies and their performance", *SIGMETRICS 2002.*, pp. 266-267, 2002.

Alexander Thomasian and Chang Liu, "Disk scheduling policies with lookahead", *SIGMETRICS Performance Evaluation Review*, pp. 31-40, 2002.

To my beloved mother – you are always by my side.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my committee of Dr. Joseph Y Leung, Dr. Artur Czumaj , Dr. Marvin K Nakayama, Dr. David Nassimi and Dr. Jian Yang. Their huge support provided me sufficient self-confidence to complete my dissertation. I am particularly thankful to Dr. Joseph Leung for offering me great help at my crucial moment. His invaluable guidance and encouragement have contributed significantly to the work presented in this dissertation.

Deepest thanks to Professor Nina Pardi, who has spent a great deal of time and energy in making grammar corrections for my dissertation during the last few months. I cordially thank for her elegant comforts and never-ending patience.

I would like to extend my deepest appreciation to Dr. Maryann McCoul for assisting me to construct a healthy, sunny life. Her invaluable advice and understanding are my lifelong treasure. I am also deeply thankful to Dean Gentul and Dean Seidman for their persistent support.

I would like to thank my dearest friends: Chunqi Han, Li Zhang, Gang Fu, and Yue Li for their warm hands through the years of my work and living. Special thanks go to Congzhe, Chunqi Han and Gang Fu for their cooperation and instructions. I also acknowledge the support of NSF via Grant 0105485 in Computer Systems Architecture.

Finally, I would like to appreciate my dearest sister, Kang, my brother-in-law, Rick, my nephew, Leo and my close friend, Jimmy, for their infinite love and support.

This dissertation is dedicated to my beloved parents, thanks to their company and encouragement in the past, at present and in the future.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

This chapter presents an overview of this dissertation, followed by a description of the organization.

1.1 Overview of Disk Scheduling Problems

There has been a rapid advance in the speed of central processing units (CPU) of computer systems, such that multi-gigahertz microprocessors are inexpensive commodity items. This change has been due to advances in semiconductor technology, which allows millions of transistors, implementing sophisticated computer organizations, to be placed on a single chip.

There has been a rapid increase in the speed of the main memory access. Core memories whose access time was measured in microseconds were replaced with faster semiconductor memories. Since the fastest SRAM (static RAM) memories are quite expensive, a memory hierarchy with caches is used with cheap multi-gigabyte DRAM (dynamic RAM) memories at the lowest level of the hierarchy.

The *magnetic drum*, also referred to as *drum*, was once used as the primary storage device in early ages. Drum was permanently mounted in the device and had small storage capacity. It was an expensive device because of a dedicated read/write head per track.

Half a century ago, magnetic drums were replaced with the more cost-efficient magnetic disks, in which the read/write heads are movable and shared among many tracks. The number of tracks on a disk surface is in the tens of thousands, while the number of disk platters tends to be small, sometimes only one. Major advances in magnetic disk technology have resulted in a rapid increase in disk capacity. This

increase is due to the growth in *areal density*[1], i.e., higher track density and higher recording density.

Zoning, a technique which has been applied to increase disk capacity, makes the recording densities of the inner tracks and the outer tracks roughly equal, so that the outer tracks hold more sectors than the inner ones. This, combined with the lowered latency resulting from improvements in rotation speed (RPM), results in a higher increase in data transfer rate for outer cylinders than for inner ones.

However, while the disk industry publicizes its ever-falling cost per gigabyte, they less often advertise storage performance per disk drive. Customers buying storage systems to support today's high-end online applications must consider not only the areal density, but also the efficiency of accessing data.

The access time to a target data block on a magnetic drum was quite fast: half a rotation time (of a typical 2400 RPM) plus the transfer time. For disk drives, in addition to a delay at the local controller, the access time comprises seek time, rotational latency, and transfer time. OLTP (Online Transaction Processing) applications typically access small data blocks [1]. In this case, the time to bring the read/write head to the data is much higher than the time to read or write the data. For this reason, performance of such a system is usually measured in access rate rather than in megabytes per second. The access rate must match to that supported by the storage system.

However, given that each OLTP transaction makes ten disk accesses and each disk access takes ten milliseconds with a *First Come Fist Served* (FCFS) processing order, more than a hundred disk drives are required to sustain a 1K TPS (transactions per second) OLTP system.

---

[1]Areal density is a measure of the number of bits that can be stored in a unit of area. It is usually expressed in bits per square inch (BPSI) and computed as the product of track density and recording/linear density, expressed in tracks per inch (TPI) and bits per inch (BPI) respectively.

Disk access rate, which is the inverse of the access time, is not increasing in proportion to disk capacity. This is attributable to larger file and database buffers in main memories, whose size has increased rapidly due to inexpensive high capacity RAM chips. The access rate becomes the disk performance bottleneck and hence results in longer access times for customers.

Modern disk drives are equipped with an *onboard disk cache*, which is usually used to prefetch the remainder of a track where sequentiality of access is detected. The onboard cache is of little benefit to disk accesses requiring random accesses. The cache at the disk (array) controller can be used to satisfy read requests (*Read*s for short) and even write requests (*Write*s for short) by overwriting a previously modified block. This process of updating data or parity on the disks from the write cache is referred to as *destaging*. Most operating systems provide a buffer for files and database management systems (DBMSs) provide their own buffer.

The cache effect is not considered in this dissertation due to the following reasons: first, the onboard cache obviates disk accesses when there is a hit, and it is not effective for random accesses; second, cache effect depends heavily on workload characteristics and varies from case to case; third, this study emphasizes improving performance at disk level.

Utilizing the knowledge of the system and the information of individual requests to schedule the disk arm is an important and effective way to reduce the average mean disk access time, hence increasing disk access bandwidth. Many disk scheduling policies to improve disk performance have been proposed over the years, and even more sophisticated algorithms can be implemented with the availability of the more powerful microprocessors associated with disks [2, 3, 4, 5, 6]. The onboard disk cache also presents opportunities, which have been taken into account in only a few studies. Chapter 3 presents a review of some well-known scheduling policies, and their performance comparison is given in Chapter 5. Three new scheduling al-

gorithms are specified and evaluated in Chapter 6. These are joint work with A. Thomasian, and have been published in [7, 8, 9].

A major revolution in secondary storage has been the introduction of *Redundant Arrays of Independent Disks - RAID* [10], whose organizations were initially classified into five levels (see Appendix A). The plan is to aggregate multiple disks and conceptually present the abstraction of a logical disk. RAID1 and RAID5 are the most popular levels and are widely implemented in commercial products. RAID has higher reliability and improved performance, which are achieved by utilizing two architectural techniques: *data redundancy* and *data striping.*

According to the RAID level, a disk array can tolerate one or more disk failures, so that the storage subsystem can continue operating with disk failure(s). This is called system in *degraded* mode as opposite to *normal* mode, in which all the disks are operational.

Striping is a technique addressed to balance disk loads, or eliminate data access skew. The linear address space of a large file is broken into smaller blocks which are distributed over multiple disks. The size of these blocks is called the "striping unit" [11], defined as the maximum amount of contiguous data assigned to a single disk. Striping has the advantages of automatic load balancing and high bandwidth for large sequential transfers. However, these two benefits present tradeoff in selecting an appropriate striping unit: a larger striping unit may result in a file distributed in partial disks (possible one); while a small striping unit may increase the bandwidth for a single logical request by involving more disks, hence reducing the number of concurrent requests that the disk array can handle.

RAID1 or *disk mirroring* replicates data on two disks, and these two data copies are called a *parity pair* or a *mirrored pair*; read requests can be processed at either of the two disks. So that the disk bottleneck problem can be solved by providing this high availability of data, which is achieved via some form of data redundancy.

Some optimizations are also possible to be implemented in processing Reads. For example, requests to outer disk cylinders can be sent to one disk and requests to inner disk cylinders to the other [12]. Note that such a scheme minimizes the disk service time by reducing the seek distance and hence the seek time. Another more dynamic policy sends a request to the disk which offers the shorter seek distance.

The controller in RAID1 disk arrays may be tightly coupled with the two disks, so that it is directly involved in scheduling requests on individual disks. The *Shortest Access Time First - SATF* policy applied to two disks rather than one disk can more than double the performance of a disk. A tightly coupled controller is not applicable to modern disk drives, which carry out their own scheduling. In this case the controller sends read requests to one of two disk drives with an appropriate routing policy. Several routing policies have been considered in [13] and are discussed in Chapter 7 of this dissertation.

When part of the disk array controller cache is *nonvolatile storage*, i.e. NVS, there are additional opportunities to improve performance. Firstly, the requests written into the NVS cache are considered completed from the application viewpoint. This operation is also known as a *fast write*, which allows the processing of Writes to be deferred, so that Reads can be processed at a higher priority than Writes. Secondly, Writes are processed in batches, so that the scheduler can take advantage of disk geometry to minimize the completion time of a set of requests. A two-phase method to process Reads and Writes, i.e., a mirrored disk scheduler that utilizes one disk for processing Reads, while the other disk is processing Writes is proposed and evaluated in [14]. An improved method will be discussed in Chapter 8. This is joint work with A. Thomasian and appears in [15].

Improved performance is achieved in RAID system via high concurrency. However, large disk arrays are highly vulnerable to disk failures, because the reliability of a system is a function of the reliability of its components. For example, a disk array

with 100 disks is 100 times more likely to fail than a single-disk array. Thus, *data redundancy* is included to tolerate disk failures and allow continuous operation without data loss. This is how redundant disk array provides protection and hence its popularity.

1.2 Organization of this Dissertation

This dissertation is organized as follows. Chapter 2 provides a series of hard disk information useful to the disk scheduler. Chapter 3 reviews representative disk scheduling studies for single disk and mirrored disks. Chapter 4 describes the methodology employed in this work, including a discussion of disk model and different types of workloads. Chapter 5 compares the traditional scheduling policies for single disk, and presents analytic methods to calculate the mean response time for basic scheduling policies. Chapter 6 compares the performance of some extensions to the reviewed policies. These extensions include both existing policies and newly proposed ones. Chapter 7 demonstrates a hierarchy of the scheduling methods in mirrored disks, and compares their performance with a synthetic workload. Chapter 8 provides a study of mirrored disk scheduling with a non-volatile shared (NVS) cache. Chapter 9 compares various RAID1 configurations in perspectives of load and system reliability. Chapter 10 concludes the dissertation. Appendix A gives a full description of RAID levels[2]. Appendix B provides the fundamental knowledge of queueing theory. Appendix C illustrates the efficiency of the SATF scheduling policy when implemented over a batch of requests. Appendix D demonstrates the routing and scheduling schemes which are compared in Chapter 8 by flowcharts.

[2]Appears in [16].

CHAPTER 2

USEFUL INFORMATION FOR RESEARCH

This chapter provides two categories of information that are useful to disk scheduling study. The information relating to the hardware includes disk structure and data layout; the information regarding requests includes their type and priority issues.

2.1 Characteristics of Modern Disk Drive

This study considers widely available magnetic disk drives with embedded SCSI (Small Computer Systems Interface) controllers.

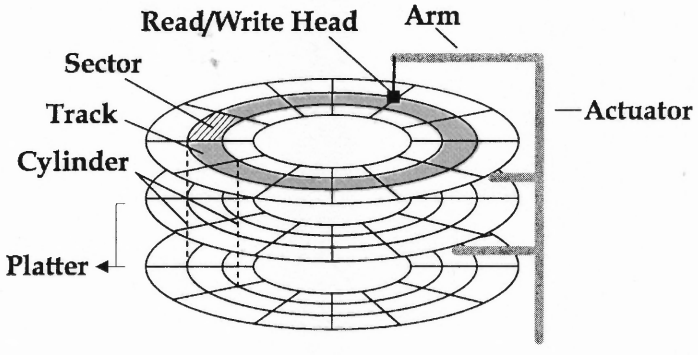


Figure 2.1 Disk drive structure.

As shown in Figure 2.1, a magnetic hard disk is usually made up of a set of *platters* in a stack. The platters rotate at a fixed speed, which is typically measured as revolutions per minute (RPM). For each platter, a pair of electromagnetic *read/write heads* are mounted at the end of a disk arm and positioned above and below each platter. The disk arms move in toward the center surface or out toward the edge, driven by a common shaft called the *head actuator*. Therefore the read/write heads move inbound and outbound. This movement, combined with the platter spin, enables the drive heads reach the entire surface of each platter.

Basically, tracks, sectors, and cylinders are the divisions of the hard drive platters where the data is stored. *Tracks* are concentric circles placed on the surface of each platter. Each track is divided into smaller units called *sectors*. Tracks with the same radius constitute a *cylinder*.

The tracks are numbered starting with zero at the outside of the platter, and increasing toward the platter center. Each sector holds 512 bytes of user data, plus a few dozen additional bytes used for internal drive control: a header which identifies the sector by address information, and a trailer for error detection and correction. All the sectors are numbered sequentially as block addresses and appear as a linear address space to the user.

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

The *disk controller* includes an embedded microprocessor, some buffer memory, and an interface to the SCSI bus [17, 18]. The controller has full knowledge of the data layout and performs mappings between logical block numbers (LBN) on the disk and the physical sectors that store the information. It services disk requests, maintains pending requests residing at the on-board cache and schedules the queues.

2.2 Data Layout

A logical hard disk appears to users as a linear vector of logical block numbers [16]. These numbers must be mapped to the non-faulty physical sectors. The mapping is complicated by zoning, cylinder/track skew and sparing, etc.

zoning

As the disk capacity increases, the gap of recording density between inner and outer tracks becomes even larger. To more efficiently use disk space, more data is stored at outer cylinders. To simplify the bookkeeping problems, zoning technique is introduced to maintain approximately the same bit recording density. The adjacent disk cylinders are grouped into several zones, each of which has a fixed number of sectors, or data bits, per track. This number grows with the radius of the tracks in the zone. Since the data transfer rate is proportional to the rate at which the media passes under the read/write head, the outer zones have higher data transfer rates. Thus zoning increases not only the capacity per unit volume, but also the disk transfer rate.

track and cylinder skew

In the cases when a transfer spans two adjacent tracks, two read/write heads need to be activated one after the other to fulfill the access. The head switching takes a short time ($\approx$ 1 millisecond) and is called the *head switching time.*

Consequently, to ensure that the data of the next track can be read right after the head switching, the first sector of the next track is positioned at the aligned position to the last sector of the previous track with an angle, which is equal to the rotation speed times head switching time. A *track skew* is then defined as the number of sectors that takes up this angle. Similarly, when a transfer spans two consecutive cylinders, a *cylinder switch time* (which is the seek time of one cylinder plus non-overlapped head switch time) occurs and corresponding *cylinder skew* is defined.

spare area

Disks invariably have some flawed sectors that cannot be used. The references to those flawed sectors are remapped to other portions of the disk. This process, known as sparing, is done at the granularity of single sectors or whole tracks.

2.3 Request Priority

At a more macroscopic level, improving the performance of OLTP applications, as exemplified by Transaction Processing Performance Council's (TPC) benchmark, subjects disk drives to a challenging workload: random accesses to small data blocks. The maximum transaction throughput is a critical performance measure in TPC, while the mean or a percentile of response time is below a certain limit. This implies that the mean response times of read requests (*Read response times* for short) directly contribute to response time are of interest. The response time of write requests (*Write response time* for short) is not important not only because the application response time is usually only affected by Read response time, but, more importantly, because the write requests can be cached in a fast non-volatile storage (NVS), which allows write requests to be deferred and processed in a batch mode more efficiently. In this dissertation, only the Reads generated by the controller cache misses and the Writes due to destaging from the NVS portion of the cache are under consideration; Reads are given higher priority than Writes.

CHAPTER 3

REVIEW OF RELATED WORK

There have been a very large number of simulation and analytic studies of disk subsystems, in general, and disk scheduling, in particular. Studies in the latter category consider a single disk in isolation, while studies in the former category consider multiple disks and take into account the knowledge of particular elements along the I/O path. Such models have been utilized to analyze a *rotational position sensing (RPS) miss*, but it is not a problem anymore because of onboard caches. The delay in initiating I/Os due to a busy bus has also been discussed in some early analytic studies, but this is not a problem provided that the bus system allows higher priority packets (to initiate I/O) to be interspersed with low priority packets, as is in the case of IBM's serial storage architecture (SSA).

Numerous studies on improving disk performance via implementing more efficient disk scheduling policies (with respect to FCFS), have been proposed in the last few decades. Although these studies provided certain insight into the performance of disk scheduling policies, many deficiencies still exist, which will also be included in the following discussions about the previous work.

3.1 Single Disk Scheduling

Disk arm scheduling methods are an effective way to reduce the disk access time and hence increase disk access bandwidth with respect to the default FCFS policy. Most early disk scheduling policies concentrated on minimizing seek time; the *Shortest Seek Time First*(SSTF) and *SCAN* policies [2] belong to this category.

SSTF processes requests based on the proximity of cylinders, but the overall service time may remain high because of rotational latency. As a greedy policy, SSTF

may result in high variability in response time. SCAN processes the outstanding requests on the path of the disk arm when it is sweeping the disk surface in alternate directions, thus reducing the seek distance as well as providing fairness.

Since SCAN tends to favor inner disk cylinders, *Cyclical SCAN* (CSCAN), which processes requests while the disk arm moves in one direction, was introduced to ensure fairness. *LOOK* and *CLOOK* correspond to SCAN and CSCAN respectively, but the scan is stopped when there are no requests waiting at the position beyond the current read/write head.

CSCAN with lookahead (CSCAN-LAi), a variation of CSCAN, reorders the next i requests in the direction of the scan, such that the sum (or average) of the predicted service times of these i requests is minimized, when the current request is completed. For example, when $i = 2$, the scheduler considers any pair of the pending requests after the current one is completed. The pair which can minimize the sum of predicted service times is selected. The pair selection is resumed in the direction of CSCAN after each current request is completed, i.e., an incoming request may be selected to be processed instead of the request which belongs to the previously selected pair.

V-SCAN, a continuum of scheduling methods, is introduced in [19]. SSTF and SCAN are two endpoints of the continuum, and the bias is tuned by the parameter R: $V(0) = SSTF$ and $V(1) = SCAN$. $R = 0.2$ is a good choice for higher arrival rates. V-SCAN combines SCAN with SSTF, in order to minimize the sum of the mean and the multiple of standard deviation of the response time, which is also known as the percentiles of response time.

Another variation of SCAN, *Window SCAN* (WSCAN) maintains a current direction, but serves all requests inside a window of $f = 10\%$ of disk cylinders (this value of f was observed to be the best) [20].

Shortest Access Time First (SATF) or *Shortest Positioning Time First* (SPTF) serves the enqueued request with the smallest predicted service (or positioning) time [3, 4]. Positioning time is the time to move the read/write head to the beginning of the block being accessed, while access time includes the transfer time. Access time is the better measure when the requested data blocks are highly variable in size and/or traverse track boundaries. When the requests access data blocks with highly variable sizes, SATF policy is susceptible to starvation, since accesses to larger chunks can be postponed indefinitely. SATF or SPTF has been shown to outperform other methods [21, 7] with the assumption that the disk scheduler has an exact knowledge of future disk timing. But this assumption is not valid in actual practice. Deviations in predicted seek time with SATF are discussed in [22], which proceeds to evaluate methods under a real-time condition.

An *SATF with lookahead* (SATF-LAi) policy [7] combines SCAN with SATF, and will be discussed in detail in Section 6.1.

Analytic studies of secondary storage devices are reviewed in [23], and with a few exceptions tend to be quite involved. A major deficiency of most of these studies is that they use unrealistic modeling assumptions for the sake of mathematical tractability, e.g., in analyzing the SCAN policy, the disk arm visits successive cylinders even though there are no pending requests on those cylinders.

One instance when the analysis is quite accurate is the M/G/1 queueing model with the *First Come First Served* (FCFS) discipline. A matrix geometric solution method is given in [24], which takes into account the dependency of successive service times (due to seek distance dependence). A comparison of FCFS results obtained via this analysis and SSTF results obtained via simulation shows that SSTF outperforms FCFS.

Some notable simulation studies of disk scheduling methods are [25], [3] and [21]. The simulation study in [25] considers all the basic scheduling policies except

SATF. The mean service time, mean response time and the squared coefficient of variation are reported.

A closed system with a fixed queue length of Q is considered in [3]. The number of requests is varied according to ($1 \leq Q \leq 1000$). The "arrival time" of each request is set to the departure time of the request it replaces in the closed system. The "useful" device utilization is equal to the product of the system throughput $T(Q)$ and transfer time, which constitutes useful work: $U(Q) = T(Q)X_{transfer}$. Scheduling methods which can take better advantage of opportunities provided by more requests in the queue result in a higher system throughput and a lower maximum response time of requests. These policies include CSCAN, *Weighted (Access) Time First* (WSTF), which modifies the I/O time based on the waiting time of a request, and *Grouped Shortest (Access) Time First* (GSTF), which services requests in a group of cylinders in SATF mode before switching to another group of cylinders. These policies will be described in detail in Section 6.1.

A comprehensive and authoritative study of disk scheduling methods, which combines random number-driven and trace-driven simulations ("to allow comparison with previous work") is reported in [21]. Similar to [17], this study takes into account detailed disk characteristics. Simulation results are verified against disk measurements in both studies. However, detailed logical-to-physical mapping information provides only a 2% improvement in accuracy compared to the case in which the mapping information is ignored. This is a valuable insight into carrying out performance analysis studies of disk scheduling methods.

3.2 Mirrored Disk Scheduling

A brief survey of previous studies of mirrored disk system is given in this section. These studies can be classified according to whether the study deals with estimating a global performance measure, such as the response time, or with a local performance

measure, such as the seek distance. There are more studies in the latter category than the former category, to which this study belongs.

The expected seek distance in (non-zoned) mirrored disks with uniform accesses over C cylinders for read (resp. write) requests is $S_r \approx C/5$ (resp. $S_w \approx 7C/15$) [26] (uniform accesses in the range (0,1) yield $S_r = 5/24$ and $S_w = 11/24$). Since Reads are more frequent than Writes, there is a significant improvement in performance with respect to single disks in which $S_r = S_w \approx C/3$. There have been more refined analyses which take into account the fact that disk arms converge to the same cylinder after processing a write request, i.e., the advantage of having two disks is lost.

A greedy policy to minimize the average seek time in mirrored disks is to choose the arm nearer to the target cylinder t ($0 \leq t \leq 1$) and place the other arm at $t/3$ if $t \geq 1/2$ and at $1-(1-t)/3$ in the other case. With independent uniformly distributed requests, the new mean seek distance is 5/36 versus 5/24. One of the two arms may be dedicated to serving the inner cylinders and the other arm to the outer cylinders, but even better performance is attainable without this restriction [23]. Even shorter seek times can be achieved if both arms are prepositioned at 1/4 and 3/4, which yield a mean seek distance of 0.125 [27]. When the fraction of Writes is w, based on a symmetry argument, the arms are placed at $1/2 \pm s$, and the optimum value for s is $s_{optimum} = 0.25(1-2w)/(1-w)$ (note that $s = 0$ for $w \geq 1/2$).

Minimizing seek distance by cylinder remapping via simulated annealing algorithm shows that different permutations of disk cylinders substantially reduced the expected seek distance [28].

Latency is a matter of concern especially if the disk arms are synchronized. The latency for completing Writes on k replicated disks increases from the usual 1/2 to $k/(k+1)$ of disk rotation time (T_{drt}). The latency for reading from k replicated

disks is reduced to $1/(k+1)$ of T_{drt} if all arms locate the target cylinder and the data is read by the arm with the lowest latency [29].

Disk access bandwidth can be increased by providing multiple arms, but in one method n arms are placed $180°/n$ apart to reduce latency [29]. Another option is to have n synchronized disks with the disk arms spaced $180°/n$ apart [29]. Replicating data blocks on the same track to reduce latency is another approach to reduce latency [29], [30].

A non-volatile storage (NVS) or non-volatile random-access memory (NVRAM) can be used to defer the destaging of modified disk blocks. An advantage of this approach is that a "dirty" block in NVS may be updated several times during its residency in NVS, thus obviating multiple disk Writes. These blocks can be destaged opportunistically as part of disk arm movement to satisfy read requests, or can be scheduled in batches, or at least can combine several neighboring requests into one, so that the positioning time for writing is minimized.

With *distorted mirrors*, a *write anywhere* policy is used on the secondary disk to minimize positioning time, while the data on the primary disk is written in place, so that efficient sequential accesses are possible [31]. In *doubly distorted mirrors*, each disk has master and slave partitions, while in *improved traditional mirrors* the location of backup data blocks is determined via a mathematical formula.

One method to improve RAID1 performance is to alternatively use one disk for reading and the other disk for efficiently destaging blocks from NVS [14]. The analysis is based on the assumption that a perfect overlap of these operations is achievable.

Analytic and simulation models of several routing and scheduling policies in mirrored disk systems are provided in [32]. A weakness of this study is that some of the policies are difficult to be realized, for example, preempt the processing of a request when the same request is complete at the other disk. Furthermore, disk service times are assumed to be exponential for the sake of mathematical tractability,

so that the effect of disk arm scheduling on reducing disk service time cannot be modelled.

CHAPTER 4

METHODOLOGY

Simulation method is an essential complement to the disk performance evaluation besides analytical method, because analytical solutions are not readily available for realistic models of most disk scheduling policies [23]. The analysis and simulation are based on four components: a detailed disk model for hard drive, which is an essential element of the discrete event-driven simulator; synthetic and trace workloads used as input to the simulator; various system configurations and disk scheduling methods being implemented; and the metrics used to evaluate disk performance under various scheduling methods. This chapter specifies each of the four components.

4.1 Simulation Model

The models that have been used in this study range from simple statistical queueing systems for analysis purpose, to highly-specified and validated disk drive models for simulation purpose. The disk is considered in isolation in this dissertation, due to its emphasis on disk scheduling policies.

In analysis, a disk can be modeled as a single server with Poisson arrivals and general service times. Although analysis cannot be efficiently carried out because disk service times are correlated (discussion will follow), the disk can be modeled as an M/G/1 queueing system (with independent disk service times) when FCFS or a priority queueing discipline is in effect. A detailed description of Poisson process and useful queueing formulas are provided in Appendix B.

To validate analytical results and to evaluate disk performance under analysis-unsolvable conditions, detailed event-driven simulators for single disk and mirrored disks have been developed in C++. These simulators undertake both synthetic work-

loads (generated by a random number generator in this study) and traced workloads, which are designated as *random nunmber-driven* and *trace-driven* simulation respectively.

The disk model, which is available in both zoned and non-zoned, is the core part of each simulator. The detailed data layout on disk, including zoning, spare cylinders, the seek characteristic, track and cylinder skews, etc., are taken into account. The effect of the track buffer is not simulated, since it will not benefit random accesses. The basic specifications are listed in Table 4.1.

Table 4.1 Disk Parameters

Zoning type	Non-zoned	Zoned			
Company	IBM	Quantum	Seagate	IBM	Seagate
Disk	Lightning	Atlas 10K	Cheetah 9LP	Ultrastar 18ES	Elite
Year	1990	1999	1998	1998	1992
Capacity(MB)	326.5/1306.1	9.1K	9.1K	9.1K	1.4K
RPM	4317.8/8635.6	10,025	10,045	7,200	5,400
Diameter (")	3.5	3.5	3.5	3.5	5.25
# of Band	1	24	11	55	14
Max Sectors/Track	48/96	334	254	390	85
Min Sectors/Track	48/96	229	167	247	61
Mean $T_{Service}(ms)$	21.78/16.72	8.70	8.65	11.55	17.78

The simulator for zoned disks initializes itself by reading the detailed characteristics of appropriate disks from [33]. The simulator for non-zoned IBM lightning disk (faster one) is manually constructed by the following characteristics:

i The number of cylinders are 1898 in total, with 14 tracks per cylinder.

ii The transfer time for each 512 bytes sector is approximately 0.29 ms: $T_{sector} = T_{rotate}/96 \approx 0.0072$ ms, where T_{rotate} is the disk rotation time calculated by $T_{rotate} = 60/8635.6 \approx 6.95$ ms. So the transfer time for a 4 KB block $x_{Transfer} = T_{rotate}/12 \approx 0.58$ ms.

iii The cylinder skew is 28 sectors ($\frac{2}{6.9/96} \approx 28$); a higher skew would also be acceptable.

iv The track skew is 16 sectors.

Disk service time is the sum of seek time, rotational latency and transfer time, given as $x_{Seek}, x_{Latency}, x_{Transfer}$. The service time for a write request is a little longer than the service time of a read request due to the head settling time, which is not considered in this dissertation because of the high rotation speed. Transfer delays in path elements, parity calculation time, and disk controller overhead are also ignored, because they are small and overlap with each other, and such hardware details are not available. Next, the methods to obtain related values of the forementioned parameters will be described for zoned and non-zoned disks, respectively. [1]

4.1.1 Seek Time

The i^{th} moment of seek time ($\overline{x^i_{Seek}}$) requires the seek time characteristic and seek distance distribution $P_D(d)$.

For non-zoned disks, every cylinder stores the same amount of data, therefore, $P_D(0) = 1/cyl$ and $P_D(d) = \frac{2(cyl-d)}{cyl(cyl-1)}$, $1 \leq d \leq cyl-1$, where cyl is the number of cylinders [34]. The mean seek distance is 1/3 of total cylinder number. For zoned disks, all cylinders do not contain the same amount of data. Therefore, this equation does not hold. The seek distance distribution for zoned disk is calculated as follows.

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

$$P_K(k) = \begin{cases} \frac{\text{\# all sectors on cylinder k}}{\text{\# sectors on disk}} & 1 \leq k \leq cyl \\ 0 & otherwise \end{cases} \tag{4.1}$$

where cyl is the number of cylinders.

[1] Section 4.1.1 has appeared previously in [16] and is repeated here with the approval of the original author.

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

$$P_{D|K}(d|k) = \begin{cases} P_K(k+d) + P_K(k-d) & d \neq 0 \\ P_K(k) & d = 0 \end{cases} \tag{4.2}$$

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

$$P_D(d) = \sum_{k=1}^{cyl} P_{D|K}(d|k) \times P_K(k). \tag{4.3}$$

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

$$\overline{x_s^i} = \sum_{d=0}^{cyl-1} P_D(d) \times [\texttt{seek\_time}(d)]^i \tag{4.4}$$

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

The seek time characteristic is

$$t(d) = \begin{cases} 2.0 & d = 1, 2 \\ 0.01(d/2 - 1) + 0.46\sqrt{d/2 - 1} & d > 0 \end{cases}$$

and it follows that the average seek time is $\overline{x}_{Seek} = 12.69$ ms.

For zoned disks, the seek time characteristic as well as detailed zoning information can be obtained by using the DixTrac tool developed at CMU .

4.1.2 Rotational Latency

Under the assumption of zero-latency or roll-mode read/write capability, i.e., transfers start on sector boundaries, and can also start with the sectors in the middle of a block,

the mean rotational latency with zero latency Reads $\overline{x}_{Latency}$ for non-zoned disk is computed as follows:

Let $q = x_{Transfer}/T_{rotate}$ denote the probability that the read/write head is in the middle of a block when the seek is completed. Then $\overline{x}_{Latency} = q(T_{rotate} - x_{Transfer} + T_{sector}/2) + (1-q)(T_{rotate} - x_{Transfer})/2$. So the mean latency with roll-mode reads, $\overline{T}_{Rot} = 3.45$ ms, is smaller than the mean latency without it, which is half of the disk rotation time $\overline{T}_{Rotate}/2 = 3.47$ ms. This difference is significant for accesses to larger blocks on a track, e.g., approximately half a track rotation if the full track is being accessed.

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

$$\overline{x^i_{Latency}} \approx \frac{T^i_{rotate}}{i+1} \qquad (i = 1, 2, 3) \tag{4.5}$$

4.1.3 Transfer Time

The transfer time for a 4 KB block at non-zoned lightning disk has shown to be approximately 2.3 ms. For zoned disks, the transfer time is calculated as follows.

Let s_k denote the number of sectors on a track at cylinder k. The transfer time of a block consisting of j sectors on cylinder k is

$$x_{Transfer}(j|k) = \frac{j \times T_{rotate}}{s_k} \tag{4.6}$$

The probability that the block is on cylinder k is $P_K(k)$ as (4.1), so that the i^{th} moment of transfer time is:

$$\overline{x^i}_{Transfer}(j) = \sum_{k=1}^{cyl} x^i_t(j|k) P_K(k) \tag{4.7}$$

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

4.1.4 Service Time and Throughput

The mean disk service time is $\overline{x}_{Service} = \overline{x}_{Seek} + \overline{x}_{Latency} + \overline{x}_{Transfer}$.

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

$$\begin{aligned} \overline{x^2_{Service}} &= \overline{(x_{Seek} + x_{Latency} + x_{Transfer})^2} \\ &= \overline{x^2_{Seek}} + \overline{x^2_{Latency}} + \overline{x^2_{Transfer}} \\ &+2\overline{x}_{Seek} \cdot \overline{x}_{Latency} + 2\overline{x}_{Seek} \cdot \overline{x}_{Transfer} + 2\overline{x}_{Latency} \cdot \overline{x}_{Transfer} \end{aligned} \tag{4.8}$$

The maximum arrival rate sustainable by FCFS policy is $\Lambda^{max}_{FCFS} = 1/x_{Service}$. Note that in mirrored disk experiments, Reads and Writes are distinguished. The mean response time of a logical write request is the expected value of the maximum of the two physical Writes.

4.2 Workloads

4.2.1 Synthetic workloads

Synthetic workloads are mainly used to compare previously proposed work and to obtain a starting point for further experiments. The worst case workload, i.e. random accesses to small data blocks, will be mostly utilized. The workload generated by an OLTP application in an airline reservation system is utilized throughout this study

[1]. The analysis of this airline system traces shows that 96% of accesses are to 4 KB blocks and the remaining 4% are to 24 KB blocks. A case in which all accesses are to 4KB blocks is assumed for simplification.

The service time of "modern" disks is dominated by the positioning time for random requests due to the high rotation speed, so that exact sizes of smaller requests have very little effect on performance. When the stripe unit is much larger than the maximum block size being accessed, the possibility that a request will cross stripe unit boundaries and access two disks is quite small. Based on these facts, the requests are assumed to be randomly distributed over all disk blocks in this dissertation.

Transactions, which are generating I/O requests, are assumed to run at a high degree of concurrency. Although individual transactions do not generate requests according to exponential interarrival times, the superposition of these I/O request arrivals from a large number of sources can be approximated by the Poisson process.

4.2.2 Traces

Simulations driven by random number generator offer more flexibility and control of workloads, but results from simulations with trace workloads are more meaningful and have more credibility, since they create a more accurate emulation of real workloads.

The trace-driven simulation, with the workloads provided at [35], is used for studying the policies that are of interest. In trace files, each record consists of five fields: application ID, request type (Read or Write), logic block address (LBA, in sectors), size (in bytes) and arrival time (in seconds). The trace files were converted into binary format in experiments for faster loading.

Three traces: *Financial2* ("F2" for short), *WebSearch2*, and *WebSearch3* ("W2" and "W3") are selected for experiments. The basic characteristics are shown in Table 4.2, followed by the details of the arrival and locality patterns. The arrival pattern is shown by plotting the number of arrivals per time-interval versus evolving time,

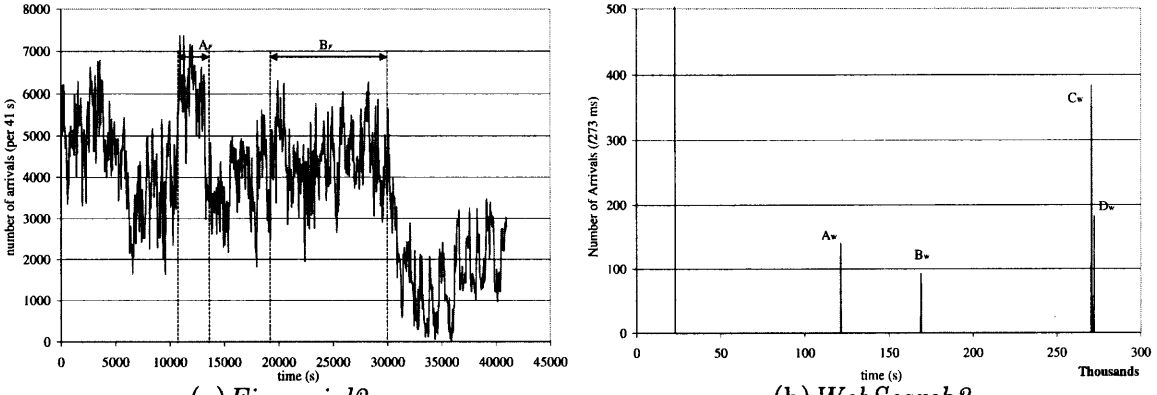
Table 4.2 Characteristics of Traces

Trace Name	Financial2	WebSearch2	WebSearch3
Length (hours)	11.4	4.3	83.0
Number of requests	3699195	4579809	4260449
Maximum LBA (sector)	1215233	34967824	34967792
Average request size (KB)	2.45	15.43	15.78
Overall Read Percentage(%)	82.35	99.98	99.98
Coefficient of variation of inter-arrival time	5.53	1.19	1048.9

and the locality pattern is shown by plotting the access frequency within each LBA-interval versus disk space. The principle for selecting a suitable trace to study a scheduling policy is specified next.

Arrival pattern

Figure 4.1 demonstrates the arrival patterns for *Financial2* and *WebSearch3*, both of which show that the arrival rates vary significantly over time. To better compare the performance of the queueing policies, the time intervals with heavy loads rather than a whole time range are considered in some traces for experiment.



(a) *Financial2* (b) *WebSearch3*

Figure 4.1 Arrival patterns of traces.

Financial2 is used for evaluating priority queueing policies, since it has a significant mixture of Reads and Writes. Two intervals $A_F(10800, 13000)$ and $B_F(20000, 30000)$

are selected, since they provide rather steady high arrival rates as shown in Figure 4.1 (a). The average arrival rates over the considered range are 149.37 and 106.77 requests/second for A_F and B_F , respectively.

To evaluate the policies designed for the burst of high load, two periods C_W and D_W in *WebSearch3* were experimented with. They both demonstrate short periods of overload as shown in Figure 4.1 (b).

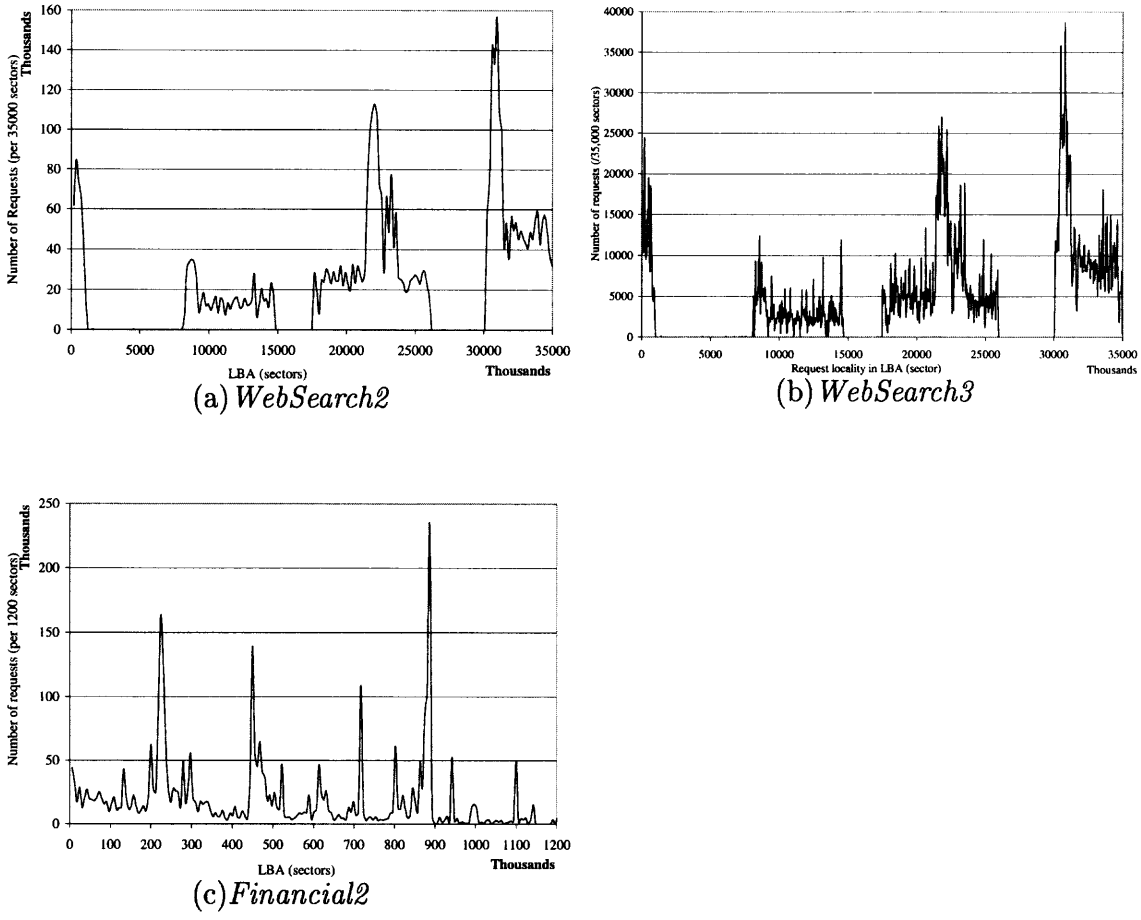
The arrival pattern of *WebSearch2* is similar to *Financial2* but shows less variability in its arrival rate. Its average arrival rate is 297.5/second, which is much higher than the maximum access rate sustainable by a single disk. Instead of using multiple disks, a preprocess, increasing the interarrival time of successive requests, was carried out to enable an affordable workload for one disk. The time stamp of each trace record is scaled up by multiplying a predefined factor. For example, when the scale factor is 2, the arrival rate is halved. By tuning this factor, the arrival rate can be varied to obtain response time properties.

It should be noted that it is usually safer to decrease, rather than increase the interarrival time, since increasing the arrival rate may create problems such as a request is generated before the completion of the previous request, which violates the original trace. This is especially so when requests are processed in a non-FCFS order, which introduces a high variability in the response time of requests.

Locality pattern

The locality of the data being accessed is another critical factor in disk performance. To obtain the favorable locality information, the disk space is evenly divided into 1,000 subareas, proportional to the maximum LBA of the data block accessed, and then the frequency of accesses is measured for each subarea. Figure 4.2 displays the locality patterns of three traces.

The locality distribution of *WebSearch2* and *WebSearch3* are quite similar: the most frequently accessed data (over 99%) is located in just four main areas for



(a) *WebSearch2* (b) *WebSearch3*

(c) *Financial2*

Figure 4.2 Locality patterns of traces.

both traces. While for *Financial2*, whose pattern is shown in Figure 4.2(c), the data accesses distribute across the entire disk but are skewed.

The locality information is especially meaningful for data reallocation, which is another technique used to improve disk performance.

4.3 Metrics

Mean response time of disk request R is the primary metric used for measuring the disk performance in this dissertation. R_{95th}, i.e., the 95_{th} percentile of response time is also reported, when reducing variance of response time is of interest. For the

experiments with Reads and Writes, the disk performance is measured by the mean and the 95_{th} percentile of Read response time: R_r and $R_{95th(r)}$, because the application response time usually is only affected by Read response time.

In the following graphs, the performance is presented by $R(R_r)$ or $R_{95th}(R_{95th(r)})$ versus a normalized arrival rate λ_n, which indicates the arrival rate normalized by Λ_{max}^{FCFS}[1]. The λ_n is incremented up to a certain point where $R(R_r)$ exceeds 500 ms, or the number of enqueued (read or write) requests exceeds 1000. This point is the "maximum throughput" λ_n^{max}, which is another important performance metric used in this dissertation.

[1] Λ_{max}^{FCFS} is the maximum arrival rate for the same workload processed with a FCFS policy, which is the inverse of its service time: $\Lambda_{max}^{FCFS} = 1/\overline{x}_{Service}$, since the utilization factor ρ for the FCFS policy has the maximum value of one.

CHAPTER 5

PRELIMINARY STUDY OF SINGLE DISK SCHEDULING

This chapter briefly reviews the traditional scheduling policies as discussed in the following section, and re-evaluates their performance using simulation [1]. An empirical study of FCFS and SATF using mathematical analysis follows. This is joint work with A. Thomasian, which has been published in [36], and the a new result has been included in [37].

5.1 Traditional Disk Scheduling Policies

Chapter 1 has shown that the improvement in disk access time (to random disk blocks) has been less significant due to the mechanical nature of the access mechanism. The transfer time is a less significant part compared to seek time and rotational latency, due to the rapid increase in disk rotation speed. Disk arm scheduling methods can be used to reduce disk access time and hence increase disk access bandwidth with respect to FCFS, by reducing seek time, rotational latency, or both. The methods under consideration are reviewed as follows.

- Shortest Seek Time First (SSTF) always selects the pending request which will incur the shortest seek time given the current read/write head position.

- SCAN moves the disk arm back and forth across the entire range of disk cylinders and serves the requests along the path.

- Cyclical SCAN (CSCAN), or one-directional scan, processes all disk requests from the outermost to the innermost cylinders (or vice versa), intending to process requests in a more uniform way.

[1] This contains joint work with A. Thomasian, which has been published in [7] and [8].

- CSCAN with lookahead (CSCAN-LAi) considers the next i requests at a time after processing the current request. These i requests are reordered to be processed so that the sum of their predicted service time is minimized.

- Shortest Access/Postioning Time First (SATF/SPTF) selects the request which minimizes the access or positioning time to process. Since this dissertation considers only requests with a fixed size, SATF and SPTF will be used interchangeably unless otherwise noted.

FCFS is the baseline policy against which all other disk scheduling policies are compared.

5.2 Simulation Results

The random number-driven simulation with synthetic workloads (see Section 4.2.1) was used to evaluate the aforementioned policies. IBM Lightning disk model (see Section 4.1 for characteristic details) is used for experiment. Faster disks, such as IBM Utrastar 18 ES are also experimented with, and they yield the same results unless otherwise noted.

The calculation of the mean service time is based on the assumptions of a roll-mode capability (see Section 4.1.2) and a Poisson arrival. For random accesses to 4 KB blocks the mean access time is $\overline{x}_{access} = 16.72$ msec., such that the maximum disk access rate for FCFS is $\Lambda_{max}^{FCFS} = 1/\overline{x}_{access} \approx 60$ requests per second.

As presented in Figure 5.1, the results of the performance of the compared policies are consistent with the results in [21]. SATF is the best performer and FCFS performs worst. CSCAN is outperformed by SCAN and SSTF, but an investigation with a "faster and higher capacity" Lightning disk shows that the SCAN policy slightly outperforms the CSCAN policy. This is attributable to the fact that requests at extreme disk cylinders encounter longer delays when there are more cylinders. The

analytically result obtained by $R = W + \overline{x}_{access}$ for the FCFS policy is also displayed in graph. The mean waiting time W was calculated using the Pollaczek-Khinchine formula for the M/G/1 queueing model. Figure 5.1 demonstrates that this formula predicts R accurately.

The SATF policy provides a significant improvement in performance with respect to SSTF and CSCAN [21], which can be ascribed to the fact that SATF is a greedy policy, that minimizes service time, hence increasing the maximum feasible arrival rate to disk.

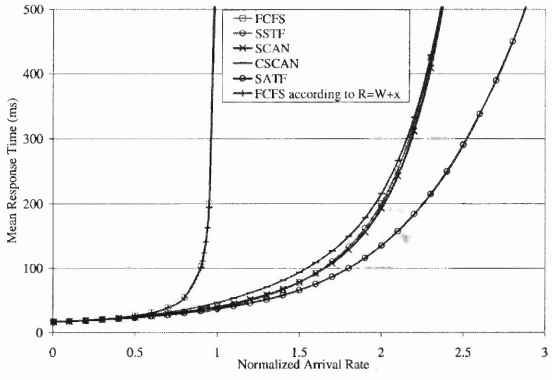


Figure 5.1 Classical scheduling policy performance.

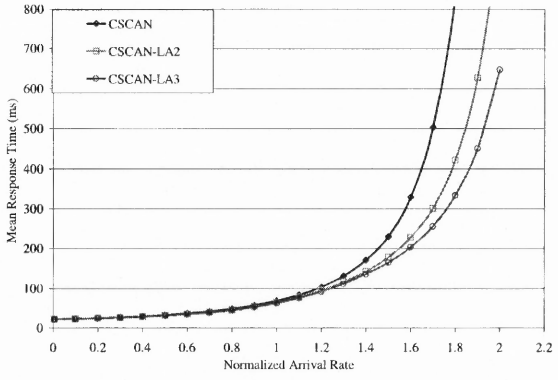


Figure 5.2 Performance of CSCAN and its variations.

Figure 5.2 displays the performance of CSCAN-LA2 and CSCAN-LA3. CSCAN and its variants attain a throughput twice as high as Λ_{max}^{FCFS}. The mean response time is improved noticeably as i is increased.

5.3 Empirical Performance Evaluation of the Disk Scheduling Policies

The previous section compares the disk performance of various scheduling policies by simulation; this section will calculate the disk request response time using analytical methods.

The analysis assumptions are introduced first, followed by a specification of the method for analyzing FCFS policy, which is the only policy that can be validated with few favorable assumptions. Finally two methods to compute the mean response time with SATF policy are proposed.

5.3.1 Introduction

A review of analytic methods to analyze the effect of scheduling methods in drums and disks and a survey of outstanding studies in this area is given in [23]. It follows that there has been more success in analyzing drums than disks, which is of course due to the simplicity of the former. Analytic studies of disks tend to make unrealistic assumptions for the sake of mathematical tractability, e.g., the read/write heads visit all tracks as part of the SCAN policy.

The following sections study the single disk in processing requests to small, randomly placed blocks of data, which are common in OLTP transactions. Requests, which have originated from a large number of concurrent transactions, are approximated by a Poisson arrival process.

5.3.2 Performance Analysis of the FCFS Scheduling Policy

A FCFS policy is one method that can be exactly analyzed by using the Pollaczek-Khinchine (P-K) formula for the M/G/1 queueing model (see Appendix B.3) [38, 39]:

$$R_{FCFS} = \overline{x}_{disk} + W_{FCFS} = \overline{x}_{disk} + \frac{\lambda \overline{x^2}_{disk}}{2(1-\rho)}.$$

The analysis is based on the assumption that the arrivals are Poisson (with rate λ) and the requests access disk space uniformly.

The mean and second moments of service time, denoted by $\overline{x}_{disk}$ and $\overline{x^2}_{disk}$, are independent from the number of enqueued requests. The disk utilization is $\rho =$

$\lambda \bar{x}_{disk}$. Disk service time is the sum of seek, latency, and transfer time, so that the mean/variance of service time is the sum of the means/variances of its components.

The analysis of a modern disk with zoning is more complicated (see Section 2.2). More data resides at outer cylinders, so that the uniform accesses to disk blocks do not result in uniform access to disk cylinders. The reduction in mean seek time is relatively small with FCFS policy.

The accuracy of this formula was verified via the self-developed disk simulator [40], and the validation results for four disk drives are given in Figure 5.3.

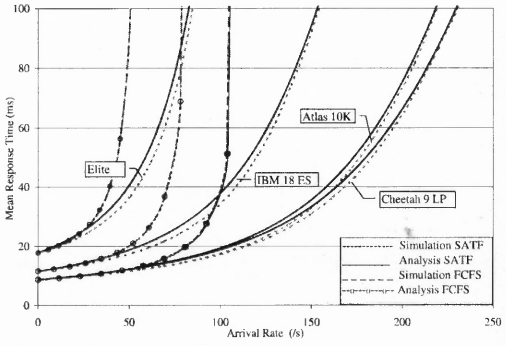


Figure 5.3 The mean response times for various disks versus arrival rate.

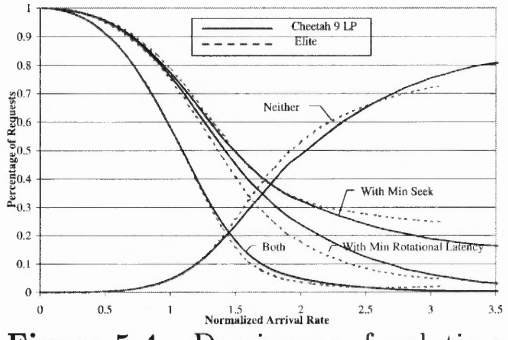


Figure 5.4 Dominance of seek time and rotational latency with SATF.

5.3.3 Performance Analysis of the SATF Scheduling Policy

The SATF scheduling policy has been verified to outperform other policies and is implemented in some disk controllers. This section presents two methods to compute the mean request response time R_{SATF} at a given arrival rate λ, when applying SATF policy.

In order to gain an insight into the behavior of SATF, a simulation for SATF scheduling was carried out to determine whether the seek time or rotational latency dominates the selection of the next request to be processed. Simulation results for two drives with very different characteristics (see Table 4.1) are shown in Figure

5.4, which demonstrates that high fraction of cases when the seek time, rotational latency, or both, determine the best SATF request sustains up to $\lambda_n = \lambda \bar{x}_{disk} \approx 1.0$ ($\bar{x}_{disk}$ is the service time according to FCFS scheduling). These fractions drop rapidly beyond this point, and neither seek time nor latency time has a dominating effect on determining the best SATF request. This is because both seek time and rotational latency have less effect on positioning time at the higher arrival rates, so the scheduler is less likely to select requests with minimum seek time or rotational latency for faster disks (Cheetah 9LP in this case). The percentage values of the considered cases follow the relation:

$$Min\_SeekTime + Min\_RotationalLantency - (Min\_)Both + Neither = 1.$$

Thus it is important to determine the minimum of the sum of two random variables: seek time and rotational latency. This determination is complicated by the fact that, the minimum is a function of both the number of outstanding disk requests and the current position of the read/write head. A simulation with SATF to estimate the mean request waiting time at a given λ_n, was also fulfilled. The P-K formula does not hold for SATF, since in view of the whole disk system, the waiting time is not independent of service time.

First method calculates the mean response time by applying Little's result $R = \bar{N}/\lambda$, in which $\bar{N}$ is obtained by using a steady-state equation. The computation of R_{SATF} is specified below.

COMPUTATION OF MEAN RESPONSE TIME IN SATF

Input: Arrival rate λ, normalized service time x[m].
Output: Mean response time R_{SATF}.
$P[0] = 1.0$ // initialization
$S = 1.0$ // to sum probabilities
while $(P[M] \leq \epsilon)$ do //
$P[m] = \lambda P[m-1] \times x[m]$
$S+ = P[m]$ // sum probabilities
$\bar{N}+ = mP[m]$ //sum number in system
end

$S = \bar{X}_{disk} S + P[0]$
$\bar{N} = \bar{N}/S$ // mean number of requests
$R_{SATF} = \bar{N}/\lambda$ // apply Little's result

The four disks are of very different characteristics, which are summarized in Table 4.1. The parameters required to analyze the disk performance with SATF policy are obtained by simulation. The results for SATF and FCFS policies are presented in Figure 5.3, which shows that SATF outperforms FCFS by a wide margin, since FCFS saturates when λ approaches $\overline{x}_{disk}$ or $\lambda_n \rightarrow 1$. The SATF policy can attain a much higher throughput than FCFS, since starting with $\overline{x}_{disk}$, the mean service time decreases as the arrival rate increases for SATF policy.

As a step in developing an analytical model for SATF, the mean service time $\overline{x}_{SATF}[m]$ with m enqueued requests was obtained by using simulation for each disk. This service time is normalized with $\overline{x}_{disk}$ in Figure 5.5, which exhibits the normalized SATF service time $x[m] = \overline{x}_{SATF}[m]/\overline{x}_{disk}$ versus m. As expected $x[m]$ is a decreasing function of m and it does not reach a limiting value even at $m = 100$.

Given the normalized service times, a birth-death model was built up, where the state is the number of requests m. The forward transition in all states is the arrival rate λ, and the departure rate is $\mu[m] = 1/\overline{x}_{SATF}[m]$. The steady-state equations are given as follows:

$$\lambda P[m-1] = \mu[m] P[m], \qquad m \geq 1.$$

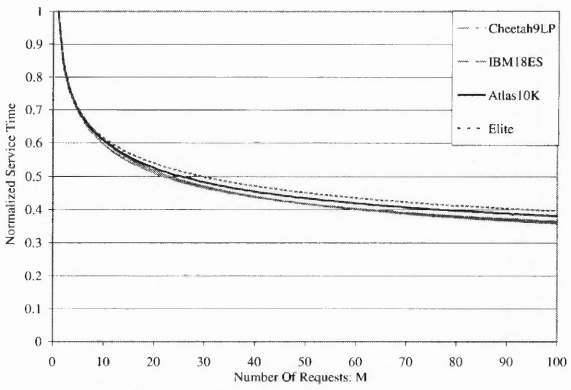


Figure 5.5 Normalized mean service time for different disk drives.

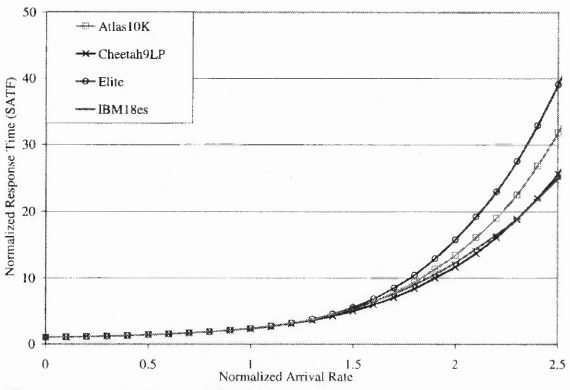


Figure 5.6 Normalized mean response time for different disk drives.

A steady-state solution exists and the system will not be saturated when $\mu[M_0] > \lambda$ for a sufficiently finite M_0. In practice the number of requests considered for scheduling is limited to the maximum number of requests that can be held on the disk M', say $M' = 128$. The $\mu[m]$ was steadied at $\mu[M'](m \geq M')$ when $m \geq M'$, since the additional requests do not contribute to the mean service time.

Instead of computing $P[m] = \lambda P[m-1]/\mu[m], m \geq 1$ (with $P[0] = 1$), $P[m] = \lambda P[m-1]x[m]$ was employed, but the results were compensated through multiplying by $\overline{X}_{disk}$ at the end. The "probabilities" are summed in a variable S, which is then used for normalization.

The above experiment was implemented for the four drives in Table 4.1. The mean response times, obtained by analysis alongside simulation results, are displayed in Figure 5.3, which shows a good match in the mean response time obtained by analysis and by simulation. However the analysis is approximate, since the disk service times are not exponential.

Given $x[m], 1 \leq m \leq M$, for one disk, R_{SATF} for another disk for a given λ can be easily calculated using the procedure outlined above, which requires the mean service time of the new drive.

A by-product, which will be useful for successive study, is obtained in the above experiment. Curve fitting to the normalized mean disk access time $x[m](1 \leq m \leq M)$ yields:

$$x[m] = m^{-p},$$

which is a good match for all four disks as shown in Table 5.1.

Table 5.1 p Value for Four Disks and Curve Fitting Standard Deviation

Disk	p	Standard Error$\sqrt{\hat{\sigma}^2}$
IBM18es	0.2181	0.0005
Elite	0.2098	0.0014
Atlas10K	0.2142	0.0074
Cheetah9LP	0.2249	0.0006

The standard error is given according to $\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2/(n-1)$, where $n =$ number of points ($n = M = 100$ in experiment).

Second method utilizes the fact that the normalized R_{SATF} characteristics of various disks (versus the normalized arrival rate) are almost indistinguishable up to very high arrival rates, so that R_{SATF} for another disk drive can be obtained by denormalizing with respect to the $\overline{x}_{disk}$ of the new disk.

Figure 5.6 presents the normalized mean response time characteristic (R_n) versus the normalized arrival rate (λ_n) for the four disk drives. The normalization for each disk was with respect to its own mean service time $\overline{x}_{disk}$, which is listed in Table 4.1. The normalized response times of various disks are almost indistinguishable up to $\lambda_n = 1.5$, i.e., 1.5 times the arrival rate sustainable by FCFS. This leads to yet another method to estimate the mean response time characteristic of a new disk, which is based on denormalizing both the mean response time and the arrival rate with the mean service time $\overline{x}_{disk}$ (of another disk).

To study the robustness of the method, a series of experimental results as various interarrival distributions: 4-stage Erlang distribution(with c_v^2=1/4) and a two-branch hyperexponential distribution (with $p = 0.1, \mu_2 = 9\mu_1$ and coefficient of variation squared $c_v^2 = 50/9$), are reported. The mean response times with different arrival patterns for SPTF are plotted in Figure 5.7, which demonstrates that the response time is higher for larger values of c_v^2 as would be expected, but the response times remain close. This is especially so for higher arrival rates with already long queue length. It is also shown that mean queue lengths are affected little by the c_v.

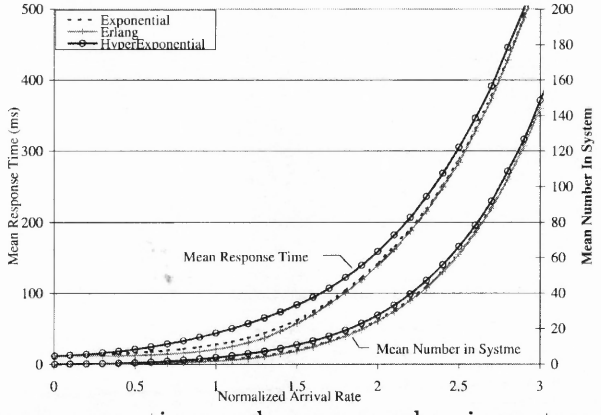


Figure 5.7 Mean response time and mean number in system for different arrival processes with IBM 18 ES disk drive.

5.3.4 Summary of Conclusions

It has been verified that SATF outperforms other well-known disk scheduling methods by simulation, but FCFS is the only policy that can be analyzed by using mathematical model based on certain favorable assumptions. A detailed analysis of the FCFS policy by using Pollaczek-Khinchin (P-K) formula for an M/G/1 queueing model is presented, followed by two methods to evaluate SATF performance when requests randomly access small data blocks on the disk. The first method builds up a birth-death model with disk access rate and the number of requests for scheduling. A by-product was produced in the experiments: the normalized mean disk access time is inversely proportional to the fifth root of the number of requests at the disk. The

second method is based on the observation that normalized response time characteristics with SATF scheduling for various disks are indistinguishable.

CHAPTER 6

PERFORMANCE EVALUATION FOR VARIATIONS OF SATF POLICY

Many studies have shown that SATF is a hard-to-beat scheduling policy [21, 8], but some improvements are possible, especially in reducing the percentiles of response time. Some variations based on SATF are developed with the intention to improve performance under certain circumstances. This chapter reviews some representative variations and proposes three new variations: SPTF-CP(t), SPTF-LAi and SAPTF[1]; the performance evaluation follows. The comparison results are obtained by using simulation with both synthetic and trace workloads. This chapter contains the joint work with A. Thomasian, which has been published in [9].

6.1 Variations of SATF

Various variations are specified in roughly chronological order; the introduction of three new variations follows.

Existing Scheduling Methods

Weighted Shortest Time First (WSTF)[3] multiplies the positioning time by a weighing factor $F = max(0, 1 - w/M)$ to prioritize the processing of long delayed requests and minimize the standard deviation of waiting (and response) time. w denotes the then-current waiting time of a request and M is a "maximum" response time allowed.

Note that as $M \rightarrow \infty$, WSTF becomes the same as SATF. Small values of M may result in $F = 0$ for several requests, which is equivalent to a random service policy.

[1]This is joint work with A. Thomasian and appears in [9].

Grouped Shortest Time First (GSTF)[3] is a combination of SCAN and SATF, which attempts to attain equitable response times for individual requests with respect to SATF. The disk is divided into groups with the size of N cylinders, and SATF is applied within each group. The requests are serviced in successive groups of cylinders cyclically, i.e., using a SCAN policy.

Batched Shortest Access Time First (BSATF)[4] dedicates serving requests in one queue at a time. There are two queues, and requests are served from one queue while the other is being filled by new arrivals. The SATF policy is applied to two queues alternatively.

The experimental results showed that the above two policies (GSTF and BSTF) yield poor performance with respect to SATF using both random and traced workloads. GSTF exhibits early saturation due to the fact that following the SCAN policy results in a significant deviation from SATF policy in selecting the request to be serviced, and thus an increase in service time. BSTF yields poor performance due to the fact that there are much fewer requests to be considered in minimizing the service time with respect to SATF. The results for these two policies are not presented in this dissertation.

Aged Shortest Positioning Time First (ASPTF)[21] modifies the predicted positioning time of each request (T_{pos}) by subtracting a weighted value proportional to its waiting time (T_{wait}): $T_{eff} = T_{pos} - WT_{wait}$. The resulting value T_{eff} is used for selecting the next request to serve. W is varied within range $[0, 30]$ and $W = 0$ corresponds to pure SATF. This algorithm is equivalent to the algorithm ASATF (Aged Shortest Access Time First) proposed in [4], in which a suggested weight translated approximately to ASPTF(6.3) is provided.

Sliding Window - Optimal Access time (SW-OAT)[4] considers all possible permutations of processing a fixed number of requests, defined as a window, and determines the optimal schedule. The scheduling step is repeated when the serviced

request is dispatched, and the oldest of the outstanding requests is pulled into the window. SW-OAT is not pursued further, since SPTF applied over the requests within the window outperforms it. The latter policy is referred to as *Sliding Window - SPTF (SW-SPTF)* in [5].

Window-Based SPTF (WB-SPTF)[6], similar to SW-SPTF, applies SPTF over only a portion of requests, i.e., a window scaled by time instead of the number of requests. The window consists of the requests whose arrival times fall within a predefined interval from the oldest arrival in the current queue. The dynamic WB-SPTF algorithm handles overload situations by adjusting the window size according to the load conditions of the system.

Newly Proposed Scheduling Methods

SPTF with conditional priorities (SPTF-CP(t)) [7] is introduced based on the observation that in head-of-the-line (HOL) priority queueing, an unconditional process of high priority requests may result in a significant degradation in performance with respect to SATF, due to the deviation from the SATF paradigm. SATF-CP(t) minimizes degradation by multiplying the positioning time of high priority requests with a factor $0 \leq t \leq 1$ to prioritize their processing with respect to low priority requests.

Designate a *winner* to be the request with the shortest positioning time in the queue from the SPTF viewpoint. SATF-CP(t) processes high priority winner unconditionally, but a low priority winner is processed only when its service time (x_{low}) is less than that of the best high priority request (x_{high}) multiplied by a threshold value $t(0 \leq t \leq 1)$, i.e., $x_{low} < x_{high} * t$. Appropriate values for t is determined by experimentation.

SATF with lookahead (SATF-LAi)[7] or *SPTF with lookahead of i requests (SPTF-LAi)*, considers i, rather than just one request at a time. In the case of $i = 2$, after the completion of request X the scheduler selects a pair of successive requests, A followed by B, such that the sum of their predicted service times is minimized.

Denote $t_{X,Y}$ to be the service time processing requests X and Y consecutively. This algorithm chooses requests A and B such that $t_{X,A} + \alpha t_{A,B}$ is smaller than all other request pairs ($O(n^2)$ when there are n requests in the queue). The second request B is given less weight by using a discount factor $\alpha(0 \leq \alpha \leq 1)$, since request B may not be processed after request A due to new arrivals.

The case $\alpha = 0$ corresponds to "pure" SATF while $\alpha = 1$ is more appropriate for a "strict" lookahead policy, in which request B is processed unconditionally after request A before any other (perhaps more favorable recent) requests. Experiments show that lookahead with discount factor $\alpha(0 \leq \alpha \leq 1)$(named "flexible" lookahead correspondingly), provides better performance than "strict" lookahead, where two requests are scheduled and processed in each round.

This algorithm can be generalized to consider i $(2 \leq i \leq n)$ requests at a time, i.e., $t_{X,A} + \alpha t_{A,B} + \alpha^2 t_{B,C} + \ldots + \alpha^i t_{Y,Z}$. The intuition behind this method is that after serving request A, the read/write head is in a favorable position to process other $i-1$ requests. Given the computational cost of $O(n^i)$ and the fact that more new requests come in during unit time at higher arrival rates, the cases $i \geq 3$ are not pursued in this dissertation.

Shortest Adjusted Processing Time First (SAPTF) attempts to reduce the variance of response time based on waiting time. This is accomplished by increasing the priorities of requests according to Table 6.1 and applying SATF-CP paradigm. For each request, its *adjusted positioning time* - APT is calculated based on the level its waiting time belongs to, and multiplied by a discount factor $f_i = t^i$ at level i. The level of a request is determined by comparing its waiting time w against a function of the mean and standard deviation of the current waiting times: $W + (i-1)\sigma_W, 1 \leq i \leq 4$, as shown in the table. By carefully choosing t, it is possible to reduce the variance and the maximum response time significantly. The statistic values of W and σ_W are updated every few time units, taking into account the variability in the workload.

Table 6.1 Levels for Waiting Time and Their Thresholds f_i

Waiting time range	Threshold.
$0 < w \leq W$	$f_0 = t^0 = 1.0$
$W < w \leq W + \sigma_W$	$f_1 = t^1$ = t
$W + \sigma_W < w \leq W + 2\sigma_W$	$f_2 = t^2$
$W + 2\sigma_W < w \leq W + 3\sigma_W$	$f_3 = t^3$
$w > W + 3\sigma_W$	$f_4 = t^4$

6.2 Simulation Results

A popular zoned disk drive, the 7200 RPM IBM Ultrastar 18ES (see Table 4.1 for characteristic details), was used for experiment. The disk drive is available in two capacities: 9.1 GB and 18.2 GB [33]. The disk with the capacity of 9 GB was used for random-number driven simulation, and the one with 18 GB was used for trace driven simulation.

This disk drive can process a maximum of 87 requests per second to small, randomly placed data blocks. But for some traces, the measured arrival rate has far exceeded Λ_{max}^{FCFS}. For such cases, the rate of the arrivals was scaled down by enlarging their arrival time so as to fit the single disk model (see Section 4.2.2 for detailed scaling method).

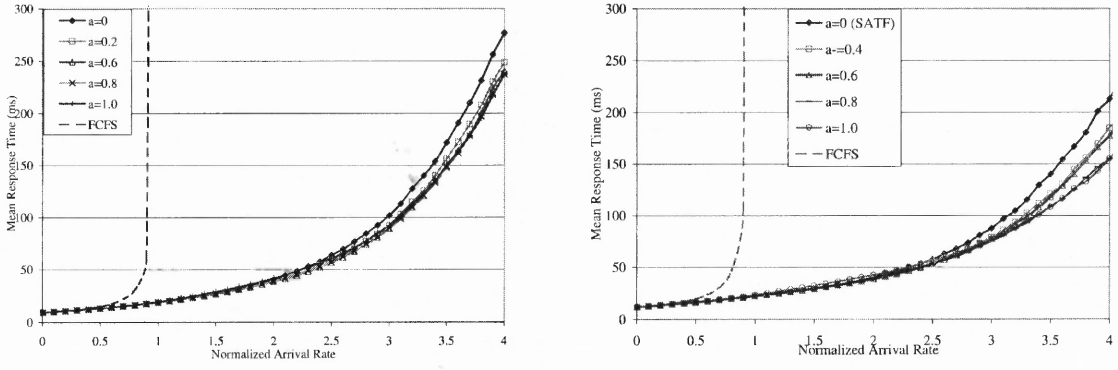
This section presents the results produced by using different simulation techniques and using different workloads. The policies covered in Section 6.1 are evaluated. FCFS and SPTF are the two baseline policies against which all SPTF variations are compared.

6.2.1 Simulation Results Using Synthetic Workload

SPTF-LA2 outperforms SPTF to a small degree where the requests are randomly distributed across the disk [7]. When the effect of locality of accesses is under exam-

ination, another extreme case is assumed: requests are sent to two small regions on the disk, r_1 and r_2, each covering 1% of the entire disk space.

The results are shown in Figure 6.1, where r_1 is located at the outermost disk, while r_2 is placed at the center in one experiment and on the tracks in the other. Experiments showed that the degree of improvement in performance grows as the distance between the two regions increases. In other words, the SPTF-LA2 policy has become more effective as the regions become more clustered but lie farther apart from each other. This enables the scheduler to more accurately predict the next request to be processed, after the current request is completed.



(a) r_2 at center (b) r_2 at innermost tracks

Figure 6.1 SPTF-LA2 performance for different values of α.

To show the sensitivity of W, Figure 6.2 plots the performance for WSTF by varying W according to 1, 2, 3 and 30 seconds. With $W = 1s$, WSTF outperforms SPTF by 21.2% at $\lambda_n = 2.1$, but the disk saturates quickly after that. With $W = 2s$ and $3s$, WSTF outperforms SPTF by about 17% at $\lambda_n = 2.5$ with an acceptable throughput. WSTF with $W = 30s$ yields almost the same performance as SPTF, because the maximum response time of SPTF was approximate $30s$ measured by experiment.

Figure 6.3 shows the results generated by repeating the experiments for ASPTF with various W. Results show that small W (2 in this case) generates comparable R

to SPTF but the improvement in $R_{95\%}$ is limited. ASPTF(12) improves $R_{95\%}$ most significantly but only up to the point when $\lambda_n = 2.0$. ASPTF(6) and ASPTF(6.3) gain significant improvement over SPTF in $R_{95\%}$ while maintaining a rather high throughput, which is consistent with the conclusion drawn in [21].

The similarity between SW-SPTF and WB-SPTF is that they both apply SPTF/SATF over only a portion of outstanding requests, with the intention to schedule the requests more fairly so as to avoid possible request starvation. Figure 6.4 displays the performance data for these two methods against SPTF. It is not surprising that both SW-SPTF and WB-SPTF have little effect at low arrival rates, and their performance deteriorates quickly as the arrival rate increases, because the degree of deviation from SPTF increases as the number of outstanding requests grows up for both algorithms.

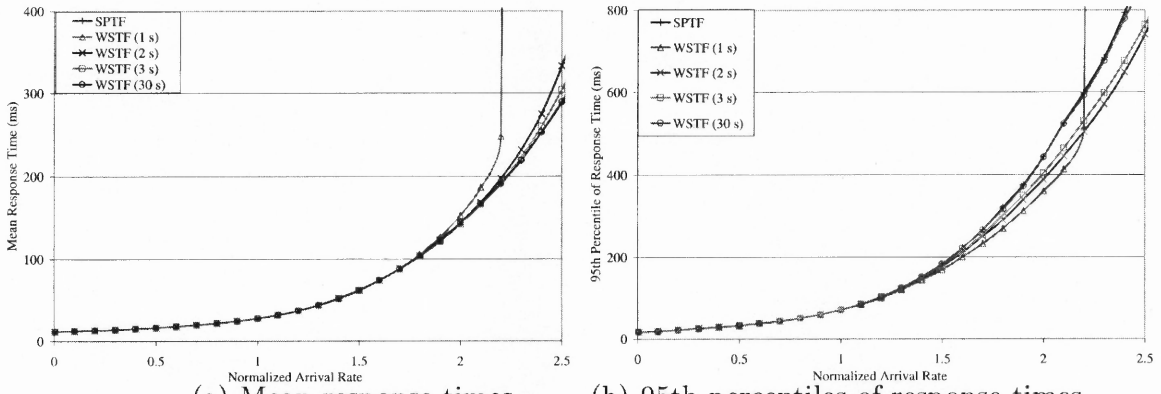
Figure 6.5 presents the performance of SAPTF policy, which is designed to moderate the variation of response time. SAPTF does not degrade R much with respect to SPTF, while maintaining a high throughput. Compared to SPTF and SPTF-LA2, SAPTF improves $R_{95\%}$ by 18% at $\lambda_n = 2.5$. The improvement would be more significant if the parameter t is further optimized.

6.2.2 Simulation Results Using Traced Workload

The performance of selected variations of SPTF is evaluated using the traced workloads described in Section 4.2.2.

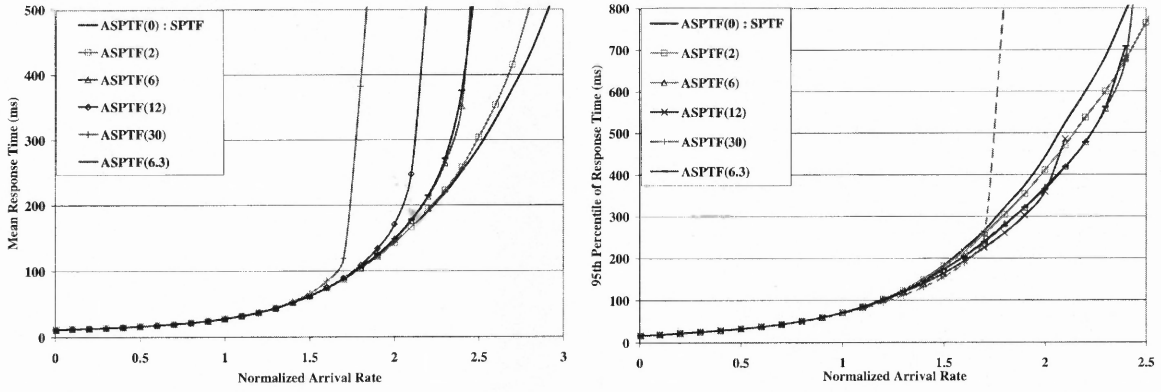
WSTF with trace "WebSearch2"

The performance of WSTF were re-examined with the "WebSearch2" trace, and the results are shown in Figure 6.6. WSTF with $W = 1$ s provides good response time for Reads but a limited throughput; WSTF with $W = 3$ s not only exhibits an improvement over SPTF in $R_{95\%}$, but yields a comparable R to SPTF as shown in



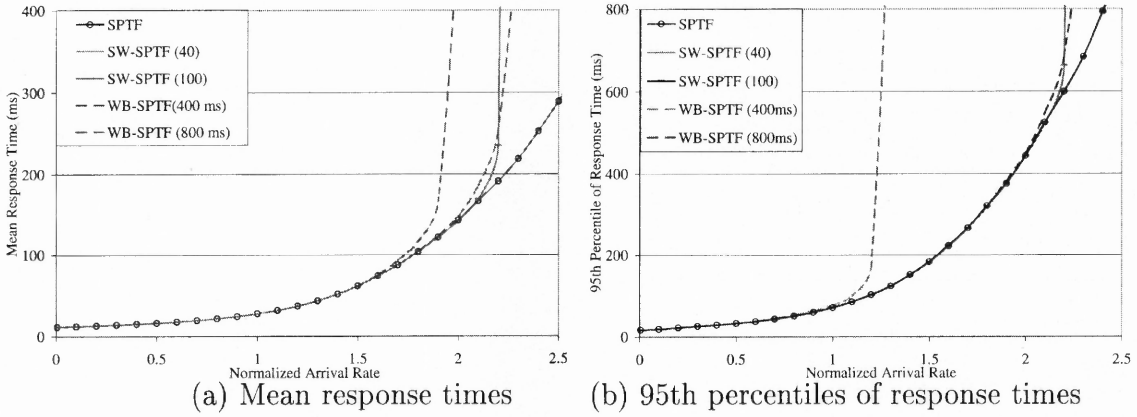
(a) Mean response times (b) 95th percentiles of response times

Figure 6.2 WSTF performance for various W.



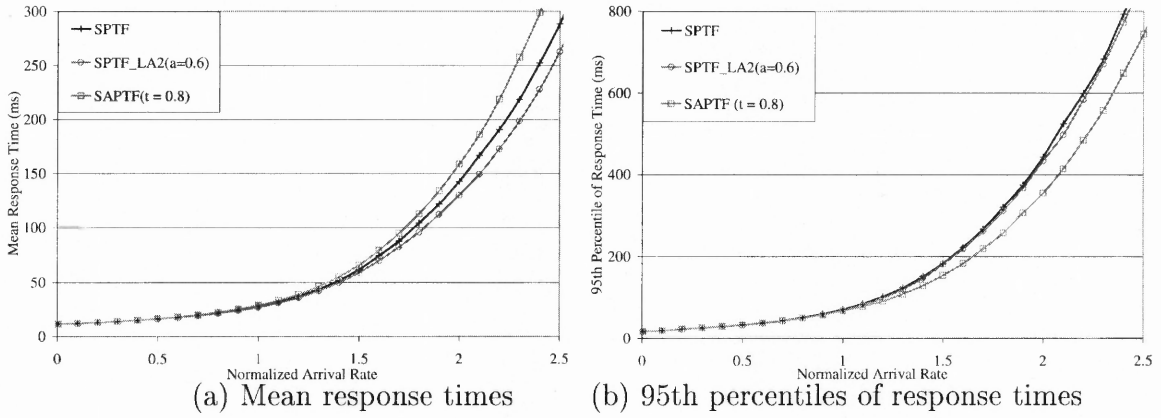
(a) Mean response times (b) 95th percentiles of response times

Figure 6.3 ASPTF performance for various W.



(a) Mean response times (b) 95th percentiles of response times

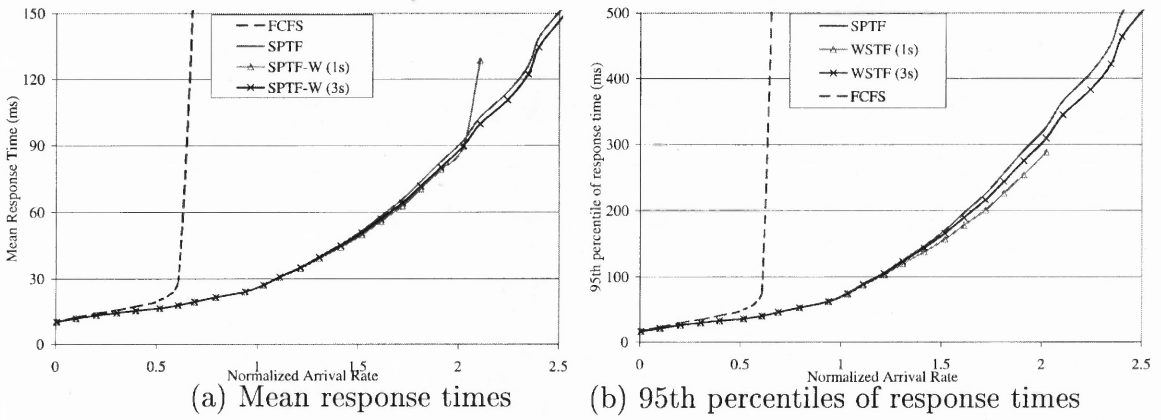
Figure 6.4 SW-SPTF and WB-SPTF performance for various window sizes.



(a) Mean response times (b) 95th percentiles of response times

Figure 6.5 SPTF-LA2 and SAPTF performance.

Figure 6.6(a), which is a more significant improvement than that obtained by the synthetic workload (see Figure 6.2).



(a) Mean response times (b) 95th percentiles of response times

Figure 6.6 WSTF performance for various W for "WebSearch2".

SW-SPTF and WB-SPTF with trace "WebSearch3"

The goal of implementing SW-SPTF and WB-SPTF is to assist the system to sustain short periods of overload. Their performance is compared against SPTF, and the results are shown in Table 6.2, in which R and $R_{95\%}$ are both listed.

Performance data shows that, SW-SPTF and WB-SPTF generate more significant improvements over SPTF during more intensive I/O bursts (period C_W in the table): SW-SPTF slightly outperforms SPTF when the first 10 enqueued requests are scheduled each time (the effect is proved to be more significant at heavier loads); WB-SPTF improves $R_{95\%}$ about 21.7% over SPTF when the window size is 50 ms. The degree of "burstiness" during C_W is higher than that during D_W in that, the "effective" arrival rate, i.e., the arrival rate only over bursts, is 151.3 requests per second for C_W, which is higher than 142.9 requests per second for D_W.

The argument is verified by the following experiment. For the same trace records at C_W and D_W of "WebSearch3", the arrivals are changed to be Poisson while keeping the overall arrival rates of that interval unchanged. The arrival time stamp of each trace record is substituted by a new value generated according to Poisson process, whose arrival rate achieves the same number of arrivals in the given interval.

The comparison results are presented in Table 6.2, which indicates that Poisson arrivals yields a much lower R and even lower $R_{95\%}$ than burst periods, as would have been expected. The disk performance with the policies under consideration is indistinguishable at C_W, because of its low arrival rate over examined time period (270700, 271000). The improvements obtained at D_W are also trivial: compared to SPTF, only SW-SPTF with window size of 4 improves $R_{95\%}$ by 6%; WB-SPTF improves R over SPTF by 3% when the window size is 60 ms, and improves $R_{95\%}$ by 5% when the window size is 40 ms.

A conclusion can be drawn that the assumption of Poisson arrivals in separate intervals is not appropriate for this trace [41].

To further examine the effect of window size on SW-SPTF and WB-SPTF, R and $R_{95\%}$ during the burst period C_W are examined for two policies respectively. The results are plotted in Figures 6.8 to 6.10.

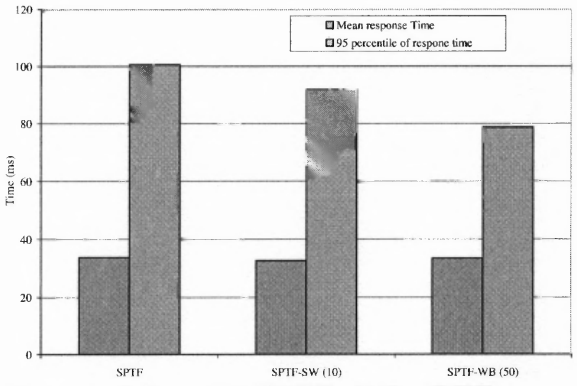


Figure 6.7 SW-SPTF, WB-SPTF performance compared to SPTF during C_W of "WebSearch3".

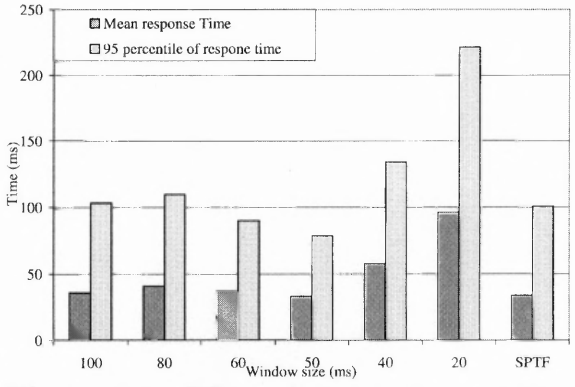


Figure 6.8 Effect of window size on WB-SPTF during C_W of "WebSearch3".

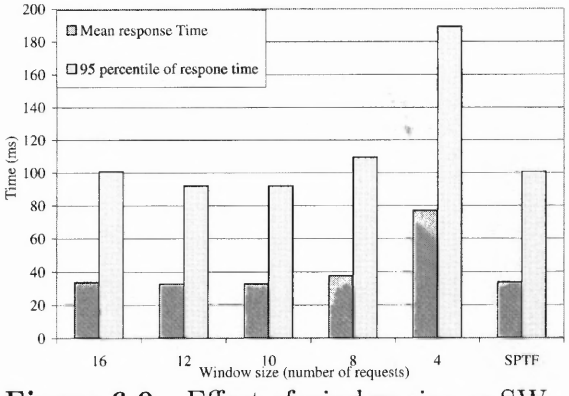


Figure 6.9 Effect of window size on SW-SPTF during C_W of "WebSearch3".

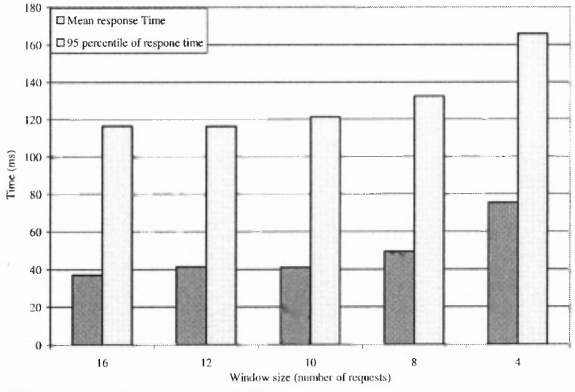
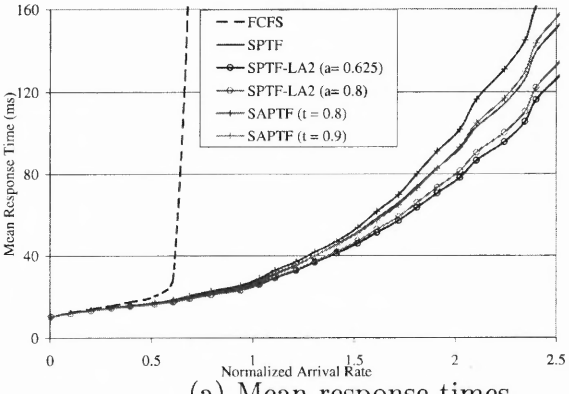
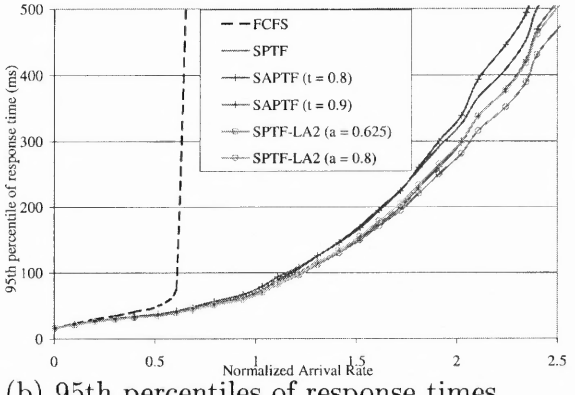


Figure 6.10 Effect of window size on SW-SPTF during D_W of "WebSearch3".



(a) Mean response times



(b) 95th percentiles of response times

Figure 6.11 SPTF-LA2 and SAPTF performance for "WebSearch2".

Table 6.2 SW-SPTF, WB-SPTF and SPTF Performance During C_W and D_W of "WebSearch3"

	C_W(270700, 271000) $ArrivalRate = 26/s$		D_W(272000, 272700) $ArrivalRate = 143/s$	
Policy	Bursty arrivals ($R/R_{95\%}$)	Poisson arrivals ($R/R_{95\%}$)	Bursty arrivals ($R/R_{95\%}$)	Poisson arrivals ($R/R_{95\%}$)
SPTF	33.8/100.57	11.71/22.74	37.1/116.31	27.47/78.81
SW-SPTF	(10)32.6 /91.88	(4 ~ 16)11.71/22.74	(12)41.29 /116.13	(4)30.67/74.13
WB-SPTF	(50) 33.5 /78.71	(40 ~ 100)11.71/22.74	(160)37.29/115.08	(60)26.65/77.09

The window sizes are specified in the parentheses before each corresponding $R/R_{95\%}$ result. The window size is scaled by the number of requests for SW-SPTF, and is scaled by time (ms) for WB-SPTF.

It is interesting to see that in Figure 6.9, the mean response time for SW-SPTF does not decrease monotonically as the window size increases, which differs from the conclusion drawn in [6]. This can be attributed to the fact that they experimented with a closed system, which holds a fixed number of requests. In an open system, however, when the arrival stream is not affected by the number of requests in the system, the best performance are obtained when the window size is 12 at the interval C_W (outperforms SPTF), and 16 at the interval D_W (the same as SPTF), which are shown in Figures 6.9 and 6.10 respectively.

For WB-SPTF, as shown in Figure 6.8, only when the window size is set within a certain region ($\approx$ 50 ms in this case) could it outperform SPTF.

SAPTF and SPTF-LA2 with trace "WebSearch2"

SAPTF is compared against SPTF and SPTF-LA2 by using traced workload "WebSearch2". As shown in Figure 6.11, ASPTF produces a comparable R to SPTF, and shows remarkable improvement over it as far as $R_{95\%}$ is concerned, which agrees with the results obtained from random simulation.

Figures 6.11 (a) also indicates that SPTF-LA2 presents a more significant improvement over SPTF with traced workload, because in real circumstances, the

data is distributed over the disk space in a more "cluster" way, which favors the performance of SPTF-LA2 according to the analysis achieved in Section 6.2.1.

SPTF-CP with trace "Financial2"

The efficiency of SPTF-CP(t) in real circumstances was verified via experimenting with trace "Financial2" at two heavy load periods: A_F(10800, 13000) and B_F(20000, 30000). The results generated by varying threshold t are listed in Table 6.3. The best performance is obtained at around $t = 0.6$, which matches the conclusion drawn by random number generated workload [7].

Table 6.3 SPTF-CP(t) Performance for *Financial2* for Different t

	A_F(10800, 13000)		B_F(20000, 30000)	
t	R_{Read}	$R_{95\%Read}$	R_{Read}	$R_{95\%Read}$
0(SPTF)	14.16	37.91	11.33	28.01
0.5	13.72	34.64	11.11	26.24
0.6	13.71	34.64	11.10	26.24
0.625	13.71	34.68	11.11	26.27
0.7	13.74	34.99	11.12	26.40
0.75	13.79	35.35	11.14	26.54
0.8	13.84	35.69	11.16	26.75

6.3 Summary of Conclusions

This chapter reviews recently proposed variations to the SATF policy, which is known to outperform other more traditional policies. New variations to the SATF policy: SPTF-LA, SPTF-CP and ASPTF are proposed. The last one dynamically varies the (conditional) priority of a request by considering its waiting time (w).

Grouping and batching do not improve performance of SATF. Dynamic Window-Based SPTF improves performance within its effective range, but it cannot sustain overload because of a very limited maximum throughput. WSTF can provide good performance with a certain maximum allowed waiting time (W), but the performance is quite sensitive to W. SAPTF performs as well as SPTF with respect to the mean

response time, but outperforms it as far as $R_{95\%}$ is concerned. The advantage of SAPTF policy is that it seems to be robust to variations in the workload, in that the priority is varied by comparing w with the mean and standard deviation of more recently processed requests.

The proposed scheduling policies are more effective at I/O bursts. However, approximating the arrival process with Poisson requests, as proposed in [41], leads to an underestimation for the mean response time of requests.

CHAPTER 7

MIRRORED DISK SCHEDULING

Disk mirroring (or RAID1) tolerates single disk failures at the high cost of doubling the number of disks. However it also provides twice the access bandwidth of a single disk in processing read requests. Improving the performance of Reads is important because: (i) transaction response times are affected by the processing time of read requests, and the processing of write requests can be deferred; (ii) improvements in disk access time are hindered by its mechanical nature.

To evaluate the performance of mirrored disks, various routing schemes, combined with queue scheduling for dispatching and processing requests, have been studied. The routing policies are carried out based on mirrored disk system configurations. The mirrored disk system can be categorized into independent queue (IQ) and shared queue (SQ) configurations in the queue point of view, or normal mode(NL) and transposed mode(TR) in the view of data allocation. The routing policies applicable to mirrored disks are partitioned into *static* and *dynamic* ones.

This chapter, which contains the joint work with A. Thomasian and C.Han in [13, 12], provides a configuration assortment of mirrored disk system, and presents a comprehensive classification of routing and scheduling policies, including a further discussion on affinity-based (AB) routing. The performance evaluation follows.

7.1 Mirrored Disk Configurations

7.1.1 Independent versus Shared Queues

Two basic queue configurations, which are also regarded as the routing principles for the dispatched requests, are considered in this study.

Independent Queues (*IQ*) or *Source-initiated routing.* The disk array controller acts as a router, which determines the disk to process a read request, while write requests are sent to both disks. There is no queue at the router.

Shared Queue (*SQ*) or *Server-initiated routing.* Requests are held in a shared queue at the router and are processed by one of the disks according to some scheduling policy. Conceptually, there are two copies of write requests, each of which has to be processed at a designated disk. The Writes need not be synchronized, so that there is no need to introduce forced idleness.

7.1.2 Transposed versus Normal Mirroring

In addition to the normal data allocation scheme (*NL*), a transposed (*TR*) data allocation scheme is also considered. In *TR* mirroring, the data on the outer cylinders of one disk is on the inner cylinders of the other disk, and the outer cylinders of both disks are used as the primary storage (marked as shadowed area) to serve read requests, as shown in Figure 7.1.

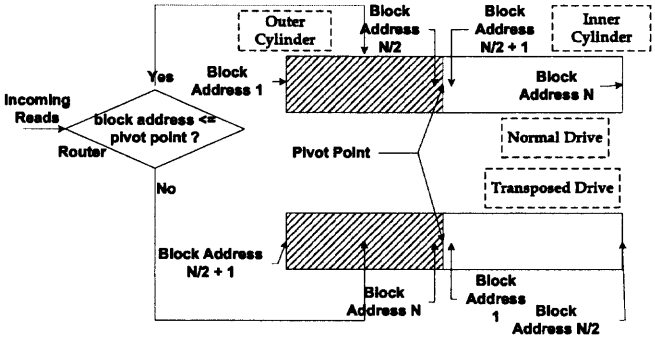


Figure 7.1 Overview of transposed mirroring system.

The intuition behind the *TR* data allocation is that, for a zoned disk drive, the outer cylinders contain the tracks with higher sector-per-track (SPT) than the inner cylinders. This property provides two benefits to the outer disk area. First, fewer cylinders are required to store equal amount of data, which results in less seek

time for reading the primary copy of data. Second, the transfer time for a block is reduced.

The point that separates inner and outer disks is referred to as the *pivot point* hereinafter. The loads for two disks can be balanced by appropriately selecting the pivot point.

7.2 Routing Schemes for Independent Queue

Routing policies are grouped into *static* and *dynamic*. Dynamic policies utilize knowledge of the current system state, such as queue length, to improve performance, while static policies do not. But both of them can use some attributes, such as logic block address (LBA) of the request. Examples of both types of policies for processing read requests are given below.

7.2.1 Static Routing

Requests Not Interpreted: The router checks only the type of requests in some routing schemes, such as *Uniform routing - UR or random routing* and *Cyclic routing - CR or round-robin routing.*

The conclusion that CR outperforms UR when applied to equivalent mirrored disk systems can be proved as follows.

As shown in appendix B.5, the interarrival times to each disk have an exponential distribution with UR and Erlang-2 distribution with CR, given that FCFS is implemented as the local scheduling policy. Assuming that the arrival process is Poisson (with rate 2λ) and service times on two disks are exponentially distributed (with rate μ), both UR and CR result in an arrival rate λ to each disk, so that the utilization factor is $\rho = \lambda/\mu$ in both cases. The mean number of requests at each disk for UR is: $\overline{N} = \rho/(1-\rho)$, and the mean response time is: $R_{M/M/1} = \overline{N}/\lambda = (1/\mu)/(1-\rho)$. The mean waiting time for CR is: $W_{E_2/M/1} = (\sigma/\mu)/(1-\sigma)$, where

$\sigma = (1 + 4\rho - \sqrt{1 + 8\rho})/2$, is the root of $\sigma = A^*(\mu - \mu\sigma)$ with $A^*(s) = [2\lambda/(s + 2\lambda)]^2$ [38]. It can be shown that $R_{E_2/M/1} < R_{M/M/1}$ since $W_{E_2/M/1} < W_{M/M/1}$ for $0 < \rho < 1$.

Requests Interpreted: The addresses of requests are extracted. This information can be utilized for *affinity-based - AB routing.*

Static Affinity Based Routing - *AB Routing*

AB routing has been investigated based on the fact that each disk serves read requests to a disjoint subset of all disk files. The "affinity" has multiple implications: timewise and addresswise. This section describes a simple scheme belonging to the latter category. The timewise AB routing is discussed in dynamic routing (see Section 7.2.2).

With static address-based AB routing scheme, the router sends Reads accessing outer cylinders to one disk, and Reads accessing inner cylinders to the other. In Figure 7.2, these two disks are referred to as the *outer disk* and the *inner disk* respectively.

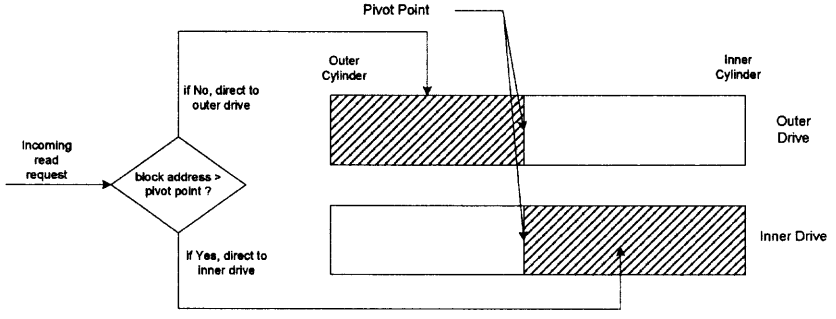


Figure 7.2 Mirrored disks with static routing based on pivot point.

As used in transposed (TR) mirroring, the point of partition of the outer and inner disks is referred to as pivot point P, which also determines the fraction of requests to each disk. Different criteria are used to determine P when requests uniformly access disk space.

i **Equal number of cylinders – EqCyl.** $P = C/2$, where C is the number of cylinders. For zoned disks, this criteria may cause unequal capacities and

request rates, when the accesses to the data blocks are uniformly distributed on the disk.

ii **Equal capacities – EqCap.** The capacities of outer and inner disks are equal, so that uniform requests will result in equal access rates for a given workload.

iii **Equal capacities with TR mirroring -EqCapT.** Referring to Figure 7.1, more general blocks on two disks are stored in "opposite" directions, but this should be done at a sufficiently coarse granularity to ensure efficient sequential accesses.

iv **Equal utilizations – EqU.** Denote the fraction of requests to the outer (resp. inner) disk as f_o (resp. $f_i = 1 - f_o$), such that $\rho_o = \rho_i$, and hence $f_o b_o = f_i b_i$. Note that f_o and P are independent of λ, but this is not the case for the two methods that follow.

v **Minimum overall mean response time – MOR.** Select f_o so that $R = f_o R_o + (1 - f_o) R_i$ is minimized. The disadvantage of this method is that f_o and P vary with λ.

Even when the disk space is accessed uniformly, AB routing can be a possible cause of unnecessary idleness. A *"relaxed" AB* (AB-R) based on a threshold for the queue-length difference is one solution to this problem. In strict AB routing (AB-S), or just AB routing, each disk serves only the read requests destined by the router; while in AB-R, when one disk is idle, it starts serving the requests assigned to the other disk.

7.2.2 Dynamic Routing

Requests Not Interpreted: *Join the Shortest Queue*(JSQ): Read requests are sent to the disk with the shorter queue and ties are broken by tossing a fair coin.

Requests Interpreted: *Shortest Response Time (SRT):* The router sends an incoming request to a disk which will provide it with the smaller response time when

the request is served in FCFS order. SRT is applicable only when the disk scheduling policy is FCFS (or any other policy where the processing order of requests does not vary with new arrivals).

Optimize with respect to Last Request (OLR): The router sends the incoming request to disk which will minimize its service time (according to SATF) with respect to the last requests in the two FCFS queues. Note that a busy disk may provide a lower response time than an idle disk.

Integrated Routing and Local Scheduling (IRLS): The scheduler predicts and compares the overall mean response (service) times $R_{overall}(S_{overall})$, including the incoming request and currently pending requests, between the two ways the new request will be sent. The incoming request is routed to a disk such that $R_{overall}$ $(S_{overall})$ is minimized. This routing scheme is denoted as *MinR* (or *MinS*), and considered only in the context of SATF, because of its superior performance compared to FCFS.

The former two policies: SRT and OLR are the dynamic address-based AB routing instances, while IRLS is not.

7.3 Routing and Scheduling for Shared Queue

SQ should provide a lower response time than IQ according to the resource-sharing argument [38], i.e., the requests are served more efficiently by reducing disk idling time. The brief proof is shown as follows.

IQ configuration (with UR scheme) corresponds to two independent M/M/1 queues, and the SQ configuration corresponds to an M/M/2 system. Assuming the system has Poisson arrivals with rate 2λ and the service rate of each disk is μ, so the arrival rate to each disk is λ, and the disk utilization for both disks are $\rho = \lambda/\mu$. The mean response times are $R_{M/M/2} = (1/\mu)/(1-\rho^2)$ with an SQ configuration, and $R_{M/M/1} = (1/\mu)/(1-\rho)$ with an IQ configuration. It follows that $R_{M/M/2} < R_{M/M/1}$

for $0 < \rho < 1$. It can also be shown that $R_{M/M/2} \leq R_{E_2/M/1}$, in other words the shared resource system outperforms independent queues with CR.

7.4 Processing Write Requests

The response time of Reads can be improved by using a straightforward head-of-the-line (HOL) nonpreemptive priority policy, given that Reads have a higher priority than Writes. There is no improvement in throughput, since the inherent policy for each queue is FCFS.

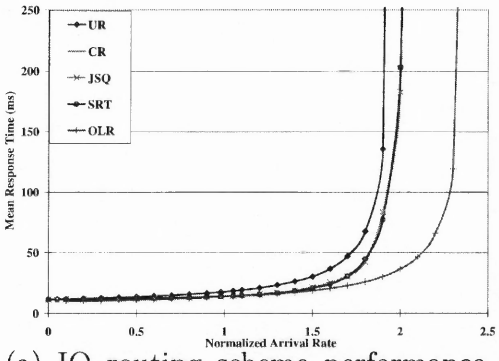
The aforementioned policy for single disk scheduling, SPTF with conditional priorities (SPTF-CP(t)) (see Section 6.1), can be utilized in mirrored disks. Similarly, t ($0 \leq t \leq 1$) is a threshold according to which read requests are prioritized over write requests. Read winners are processed unconditionally, while a write winner is processed only if its predicted service time is much less than that of the current read winner: $x_{write} < x_{read} * t$. Thus, SATF(0) is a read priority policy, where the processing of all the read requests (according to SATF) completes before the start of processing write requests (if any). SATF(1) is a "pure" SATF policy with no priorities for Reads.

There is a trade-off between the mean response time R_r, and the throughput. The "optimal" values for t are determined by experiment.

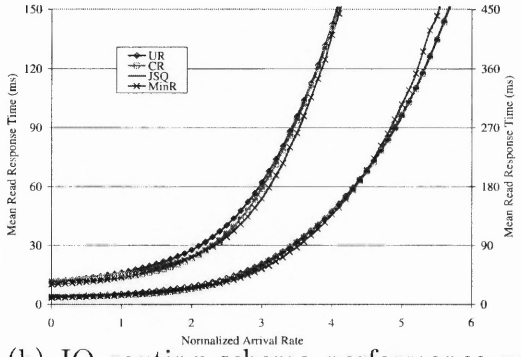
7.5 Simulation Results

A random number-driven simulation (see Section 4.1) is used in this study to evaluate the performance of mirrored disks. In experiments, the IBM Ultrastar 18ES disk drive (see Table 4.1 for characteristic details) is subjected to synthetic workloads comprising Reads and Writes. The ratio of Reads to Writes is varied according to 1:0, 3:1 and 1:1.

A conclusion has been drawn in Section 5.2 that, SATF outperforms other classical scheduling policies, so that FCFS and SATF were implemented as two extreme policies in disk mirroring study.



(a) IQ routing scheme performance with FCFS local scheduling, R:W=1:0



(b) IQ routing scheme performance with SATF local scheduling, R:W=1:0

Figure 7.3 Routing performance with independent queues.

Figure 7.3 displays the mean response times of Reads for various routing schemes with IQ configuration. As shown in 7.3(a), using an FCFS local scheduling method, UR is outperformed by CR since its arrival process is less "random" than Poisson. JSQ improves the performance by balancing the queue-lengths; SRT reduces R_r by dispatching the request to the disk which will minimize the response time of the incoming request. Both of these schemes yield indistinguishable performance with CR, and all of these policies tend to $\lambda_n^{max} = 2$, i.e., twice the maximum throughput of a single disk with FCFS. The OLR scheme achieves the best performance and throughput, which is 10% higher than other policies. OLR is a simple-to-implement dynamic policy which requires the router to remember the block number of the last request routed to a disk.

To compare the routing policies with the SATF local scheduling in IQ, MinR is examined instead of SRT and OLR, because the latter two schemes are designed for FCFS queueing model. Figure 7.3(b) shows little difference between these policies,

which indicates that the effect of implementing routing schemes is diminished by applying SATF scheduling at local disk.

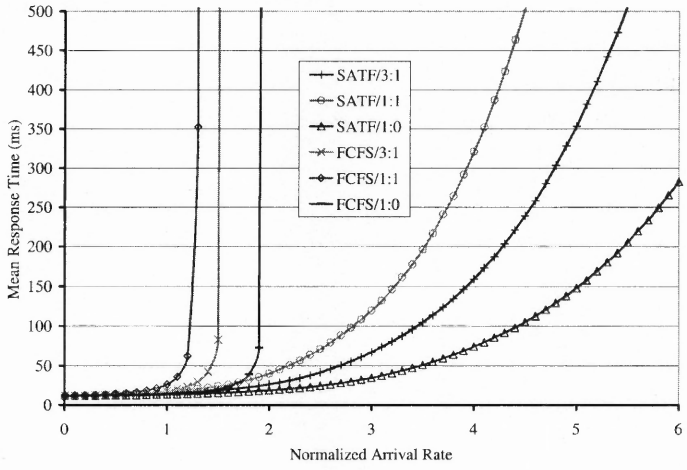


Figure 7.4 SQ performance with FCFS and SATF as local scheduling policies for different fractions of Reads.

Figure 7.4 presents the performance data for routing policies in SQ configuration. Different ratios of Reads to Writes are experimented with. The performance disparities between SATF and FCFS with fractions of Reads are consistent with the results obtained for a single disk, except that each mirrored disk processes 50% of the Reads of a single disk. Increasing the fraction of write requests significantly reduces the maximum throughput for FCFS local scheduling, but this effect is less significant for SATF, which provides a much better performance than FCFS.

The performance of SQ and IQ were then examined by varying the ratio of Reads to Writes. Figure 7.5 shows the results obtained when SATF was implemented as the local scheduling policy for both configurations, and CR was used as a typical routing scheme in IQ. From a ratio viewpoint, the response time of Reads is dominated by the fraction of Reads; from a configuration point of view, SQ configuration shows much better performance than IQ, for example, when Read:Write = 3:1, the $R_r^{SQ} \approx$ 159 ms at $\lambda_n = 4.0$, while $R_r^{IQ} \approx 291$ ms.

The performance of SQ and IQ was also compared when SATF(t) was implemented as the local scheduling policy. A CR scheme was used for IQ as before, and the ratio of Reads to Writes was 3:1. In Figure 7.6, the performance of SATF(t) with SQ in mirrored disks shows consistency with that of SPTF-CP(t) with single disk, i.e., there is a trade-off between throughput and the mean response time of Reads. However when the "pure SATF", i.e. SATF(0), is implemented in both configurations, SQ presents a significant improvement over IQ in R_r.

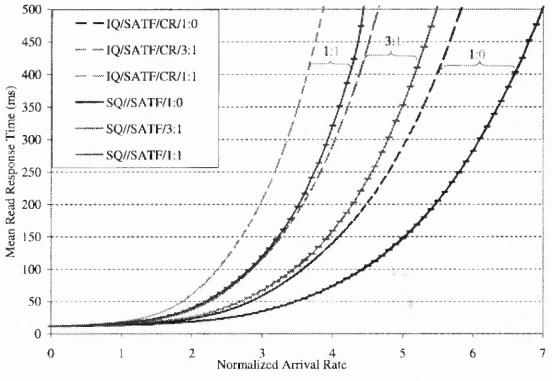


Figure 7.5 Performance comparison between SQ and IQ with CR, with SATF as local scheduling method for different fractions of Reads.

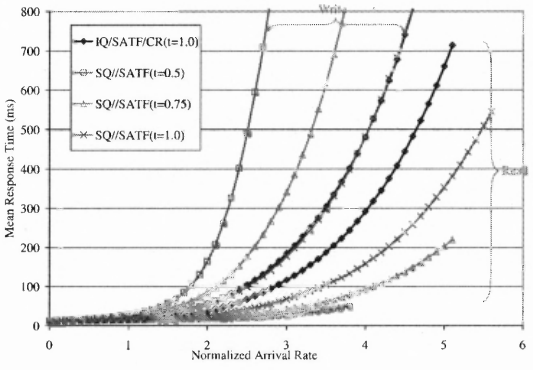


Figure 7.6 Performance comparison between SQ and IQ with CR, with SATF-CP(t) as local scheduling method, R:W=3:1.

Based on SQ configuration, a study of AB routing is carried out and an elaborate report is provided at [12]. This study presents different criteria used to select the pivot point. Results show that under the assumption of uniform distribution of accesses, time-wise selection methods outperform the methods that equalize capacity or number of cylinders.

The effect of transposed data allocation is also investigated in [12], which concludes that the mirrored disk system with TR (transposed) data allocation provides much better performance than that of the system with NL (normal) data allocation when all proposed pivot point methods are implemented. In this dissertation, the

TR data allocation with relaxed (R), strict (S), and "pure" (P) SATF policies are further examined. Figure 7.7 displays the results for different fractions of Reads. Little difference exists between relaxed and strict policies (graphs overlap), while the "pure" SATF policy outperforms both. The gap between the "pure" SATF, which selects request without location limit, and the TR-based SATF policies decreases as the fraction of Reads becomes higher. This is because processing Writes impairs the effect of TR data allocation which aims for reducing the arm movement. However, when all requests are Reads, the inferior performance of TR-based SATF policies to "pure" SATF is incurred by forced idleness and deviation from SATF in selecting the optimal request.

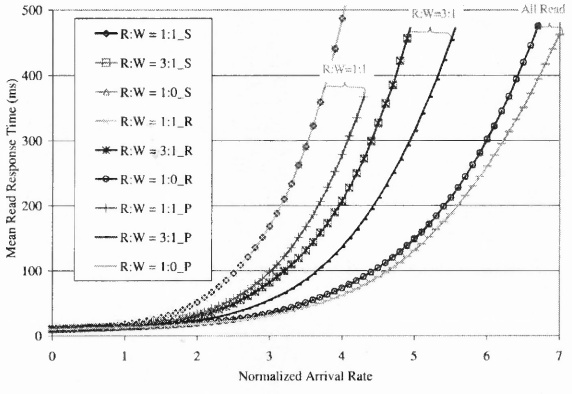


Figure 7.7 SATF performance with transposed data layout and SQ.

7.6 Summary of Conclusions

The local disk scheduling policy dominates the performance of a mirrored disk system, i.e., a much more significant performance improvement is attained with SATF compared to FCFS. The SQ configuration outperforms the IQ configuration in the form of improved response time for FCFS and improved response time and also throughput for SATF. A superlinear increase in throughout is achieved with SATF, because with increasing number of disks there are more opportunities to select requests minimizing disk service time from the shared queue.

In SQ configuration, the performance of CR, JSQ, and SRT are indistinguishable, but all outperform UR due to load balancing. OLR has a limited SATF-like effect in reducing service times, so that a noticeable improvement in throughput is attained.

Affinity based routing can also be used for improving performance when FCFS local scheduling, and transposed data allocation demonstrates further advances. However, the performance of "pure" SATF policy is superior to the aforementioned schemes due to its highest disk utilization.

CHAPTER 8

MIRRORED DISK SCHEDULING WITH AN NVS CACHE

Disk mirroring, also known as RAID1, is a popular paradigm used to attain high data availability and hence improved system performance. Caching the data, which will be finally written on the disks, in non-volatile storage (NVS) at first can be used to further improve the performance, because (i) Writes can now be deemed complete after writing the new data to the cache (fast writes); (ii) the destages, i.e., updating data and parity on the disk from the cache, are performed asynchronously in the background, so that the Reads can be processed in the foreground at a higher priority; (iii) Writes can be destaged in batches and reordered by applying scheduling algorithms, so that the average time for a destage is reduced.

The first study to exploit the caching of Writes in NVS to improve Read performance in mirrored disks was proposed by Polyzois et al. [14]. An *Alternating Deferred Updates (referred to as* ADU *in this chapter)* scheme was presented, which delays Writes and alternatively destages them to the two disks in batches.

This chapter compares three schemes to improve performance of Reads: SATF with conditional priority (SATF-CP), ADU scheme, and an improved Alternating Destages with Conditional Priorities (AD-CP) scheme. Full descriptions of ADU and AD-CP are presented first, followed by performance evaluation. The study is based on an SQ configuration and the assumption that Reads are given higher priority than Writes. Only the Reads (to disk) which result from misses in the caches, and the destaging of dirty blocks or Writes are under examination. The effect of another advantage: reducing the number of writing in the disk via overwriting the dirty blocks cached in NVS, is ignored. This is joint work with A. Thomasian and appears in [15].

8.1 Schemes to Improve Read Performance

The significance of improving the performance of Reads has been addressed in previous chapters. The schemes of utilizing head-of-the-line (HOL) and SPTF-CP(t) have been described in Section 7.4, and compared to each other with an SQ configuration (results shown in Section 7.5). This section specifies two more schemes, both of which utilize processing Writes in batches.

8.1.1 Alternating Deferred Updates (ADU)

A scheme to improve the performance of Reads in a mirrored disk system by deferring the processing of Writes, because they are held in an NVS cache, is presented in [14]. Writes are not processed individually but are only destaged to disks in fixed size batches. The Writes in a batch are reordered according to the physical layout of the disk, more specifically in increasing cylinder number, which is equivalent to the CSCAN method.

Each disk in the mirrored pair alternates between two time periods, during which it is processing Reads or Writes. In an ideal situation, while one disk is processing Writes, the other disk services Reads. In other words, these two time periods sum up to the exact cycle time T, as shown in Figure 8.1, which depicts such an alternation by a disk timing diagram. W represents the processing of a write batch, and $\overline{W}$ represents the time between processing write batches, during which the disk is processing Reads or is idle, and receiving Writes. There is no guarantee that the processing of a batch of write requests, referred to as a *Write batch*, can be started every T time unit, since the number of Writes in a batch, referred to as *batch size*, may not attain the threshold (TW).

If the time periods of processing Write batches do not overlap, there is always at least one disk available to service Reads, as shown in Figure 8.1. However, if the workload is write-intensive, both disks can be involved in processing Write batches

simultaneously as shown in Figure 8.2. The processing of the second batch is started when there are no outstanding read requests. The processing of a Write batch is not interrupted once started.

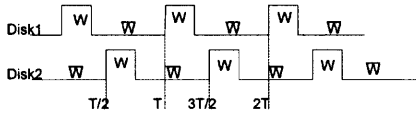


Figure 8.1 No overlapping Writes.

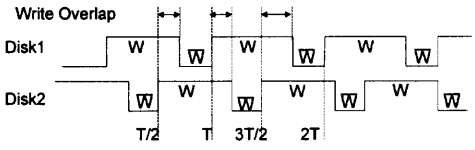


Figure 8.2 Overlapping Writes.

The key points of ADU scheme can be summarized as follows:

1. Each disk alternates in processing Reads and Writes. There is always at least one disk available to service Reads.
2. Writes are processed in batches only when the prerequisite batch size is attained, even if both disks are idle, and thus forced idleness is introduced. The processing of a Write batch cannot be interrupted.
3. Disks alternate in processing Write batches.

A detailed description of implementing ADU scheme in mirrored disk system is given in Appendix D.

8.1.2 Alternating Destages with Conditional Priorities (AD-CP)

Motivated by ADU, a new scheme is developed, tending to improve performance in three aspects: (i) eliminating the forced idleness by processing write requests individually; (ii) using an SATF-based scheduling method or even an exhaustive search for smaller batch sizes; (iii) introducing a threshold (TR) for the number of read requests to further improve performance. Each aspect is specified as follows.

Eliminating forced idleness: The new method ensures that at least one disk is always available to process Reads, but it differs from ADU method in that Writes are

not necessarily processed simply in batch mode, since this introduces forced idleness when the batch threshold for Writes (TW) is not attained. Processing of Writes one at a time in *individual mode* has the added advantage that the processing of Reads is only delayed by the residual lifetime of a single disk access, rather than a Write batch.

Implementing SATF policy: Reads are processed according to SATF; Writes can be processed according to SATF or FCFS, since FCFS reduces the number of dirty blocks in the cache, so that cache space can be released.

Applying thresholds: Reads and Writes are processed individually, unless the threshold for Writes (TW) is attained. At low arrival rates, it is possible that all Writes are processed individually since TW is never attained. A threshold TR for Reads is also introduced to ensure that Reads truly get higher priority, when there is a backlog of Reads. Theoretically, a smaller TR will improve the mean response time for Reads, since more chance is given by disk pair to process read requests. A Write batch will be processed at a disk provided: (i) the other disk is not processing a Write batch, (ii) the threshold for Reads (TR) has not been attained.

The status of disks and the corresponding conditions are listed in Table 8.1.

Table 8.1 System Condition and Corresponding Disk Status

Conditions	Disk state
TR attained	Both disks process Reads
Neither TR or Write batches attained.	Both disks process Reads and Writes
Write batch attained at both disk, while TR has not been attained	Disks alternate in processing Reads and Write batches; Reads are given higher priority

Similar to the ADU scheme, the processing of a Write batch cannot be interrupted; incoming Writes are disallowed to join into a Write batch which is in progress.

These operations are to ensure that the Writes at both disks are synchronized, so that the number of dirty blocks held in the NVS cache is reduced.

To optimize the mean service time for processing Writes in a batch, instead of CSCAN used in the original, a more flexible scheduling method, which applies exhaustive enumeration when TW is small and SATF otherwise, was used. Experiments show that SATF is efficient and the mean service time is close to that obtained by optimal scheduling (see Appendix C).

The key points of the AD-CP are:

1. At least one disk is always available to service Reads.
2. When the number of Reads exceeds a certain threshold (TR), both disks are dedicated to service Reads, even though TW may have been attained.
3. Writes are processed at both disks when there are no Reads, even though the TW is not attained. Thus, there is no forced idleness.

A detailed description of implementing AD-CP scheme in mirrored disk system is given in Appendix D.

8.2 Simulation Results

Simulation is the major method used in this study; a simplified mathematical analysis utilizing queueing theorem is also carried out for HOL scheme, due to its scheduling policy of FCFS.

The simulator used in this chapter remains unchanged from Chapter 7, including disk model and the setting of parameters. FCFS is the baseline policy against which the above discussed schemes are compared. R_r and its 95 percentile, $R_{95\%}$, are the primary metrics used for comparing various schemes; the maximum throughput, disk utilization, and the time span are also reported for selecting appropriate thresholds.

This section presents the results for HOL, SPTF-CP(t), ADU and AD-CP in two dimensions: first, examining each scheme by varying its own parameters; second, comparing the schemes against each other in an identical context, to draw an overall conclusion.

8.2.1 HOL and SPTF-CP(t) Schemes

HOL is not desirable due to its low throughput. It is easy to prove that the maximum throughput of HOL is equivalent to that of FCFS in both single disk and mirrored disk systems.

Under the assumption of an SQ configuration or an IQ configuration with uniform routing, the maximum throughput for a mirrored disk system, λ_{max}, can be calculated as $\lambda_{max} = [(f_r/2)\bar{x}_{read} + f_w\bar{x}_{write}]^{-1}$, where f_r(resp. f_w) denotes the fraction of Reads (resp. Writes), and $\bar{x}_{read}$ (resp. $\bar{x}_{write}$) denotes the mean service time. Using the approximation $\bar{x}_{write} \approx \bar{x}_{read}$ and $f_r = 1 - f_w$, the λ_{max_n}, normalized by λ_{max}^{FCFS} for a single disk, is calculated as follows:

$$\frac{\lambda_{max}^{mirrored}}{\lambda_{max}^{FCFS}} = \frac{1/(0.5f_r\bar{x}_{read} + (1-f_r)\bar{x}_{write})}{1/\bar{x}_{read}} = \frac{1}{0.5f_r + (1-f_r)}.$$

In the cases when f_r=1.0, 0.75, 0.5, 0.25, the results are 2.0,1.6, 1.33, 1.14 respectively, which exactly match the simulation results listed in Table 8.2.

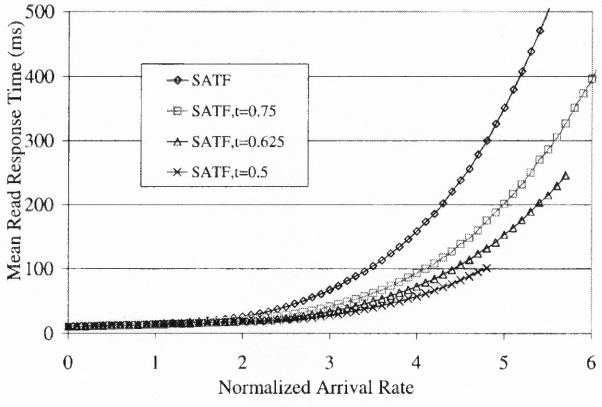


Figure 8.3 SPTF-CP(t) performance with SQ, R:W = 3:1.

Performance of SPTF-CP(t) is displayed in Figure 8.3, where the effect of threshold t is examined. R_r improves as t decreases; the best performance of Reads is achieved when $t = 0$, but the system throughput is very limited because of the quick saturation of Write queues.

8.2.2 ADU Scheme

This section first investigates the effects of Write batch size TW and the Reads:Writes ratio on the performance, and then discusses the requirements of implementing non-overlapped processing for Write batches.

Experiments were repeated for the scheme proposed in [14] by using an IBM Ultrastar 18ES disk model. Reads were routed to the disk whose current read/write head is closer to their location where the process would be performed.

The experiments evaluated the effect of TW by using 0.5 as the default fraction of Reads, which is consistent to the original. Figure 8.4 shows that as far as R_r is concerned, increasing TW results in performance degradation of ADU scheme; it is even outperformed by FCFS at low arrival rates. However, regarding maximum throughput, ADU improves performance with respect to FCFS but not significantly, and both are considerably outperformed by SATF. A quantitative comparison can be drawn from Table 8.2, which shows that the improvement of ADU on FCFS is less than 10 percent.

The degradation in R_r results from the combination of forced idleness and processing Writes in batch. Writes can not be processed individually unless TW is attained, even when the disk is idle; moreover, processing Writes in a batch without interruption may block incoming read requests, and thus hurt the response time of Reads.

The disparity in maximum throughput can be attributed to the number of Writes in buffer. The system is saturated because of the overflow of Writes. Figure

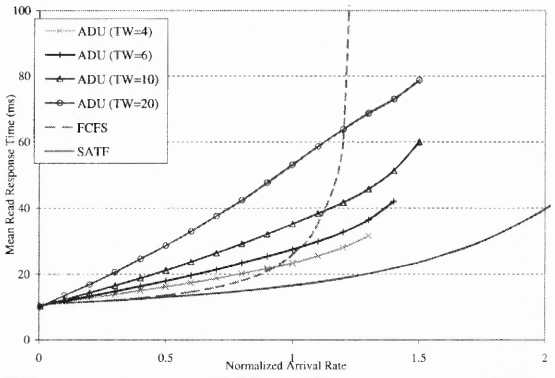


Figure 8.4 ADU performance for different TW, R:W = 1:1.

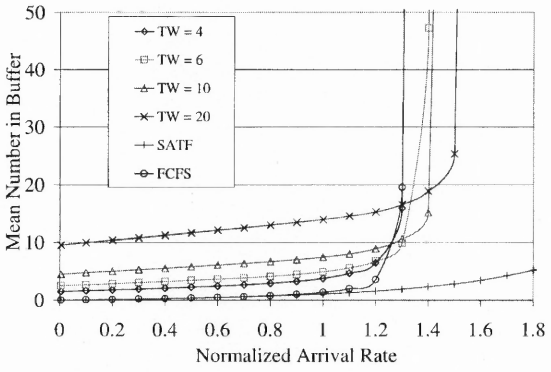


Figure 8.5 Mean number of Writes in system with ADU, R:W = 1:1.

8.5 plots the mean number of Writes in buffer, N_{write}, for ADU scheme with different TW. N_{write} grows along the increasing arrival rates as would be expected, but in a small degree, and remains below TW before system saturates. SATF achieves a much higher throughput by keeping N_{write} at a low level.

Table 8.2 lists the maximum throughputs obtained through experiment for the examined schemes. The accurate λ_{max} is between the two values in some cells due to the displayed unit of 0.1. The table also indicates that λ_{max} degrades as the fraction of Writes increases, which is more specifically shown in Table 8.3. The system saturates faster when disk is more intensively utilized for processing Writes.

The analysis carried out in [14] focuses on "non-overlapping write periods", and two conditions for "non-overlapping write-batch-processing periods" were briefly discussed. This section examines these conditions by using the results obtained via simulation.

The first condition is that there should be enough time to service Reads between processing Write batches. Referring to Figure 8.1, the number of Reads which arrive during a period T is $f_r \lambda T$, and the service time of these Reads is $f_r \lambda T \overline{x}_{Read}$. Thus, for each disk, $f_r \lambda T \overline{x}_{Read}/2 \leq \overline{W}$. The average time of W, $\overline{W}$, the sum $W + \overline{W}$,

Table 8.2 Maximum Throughput with Various Schemes for Different R:W Ratios

Schemes	R:W=3:1	R:W=1:1	R:W=1:3	R:W=1:0
FCFS	1.5-1.6	1.3-1.4	1.1-1.2	2.0
HOL	1.6-1.7	1.3-1.4	1.1-1.2	1.9
SATF	5.4-5.5	4.4-4.5	3.8-3.9	6.9-7.0
ADU (TW=4)	1.6-1.7	1.3-1.4	1.1-1.2	
ADU (TW=6)	1.6-1.7	1.4-1.5	1.2-1.3	
ADU (TW=10)	1.6-1.7	1.5-1.6	1.3-1.4	
ADU (TW=20)	1.7-1.8	1.5-1.6	1.4-1.5	

Table 8.3 Disk Utilization with ADU for Different R:W Ratios, $TW = 10$

TW=10	R:W=3:1			R:W=1:1			R:W=1:3		
Arrival Rate	Read	Write	Sum	Read	Write	Sum	Read	Write	Sum
0.005	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.1	0.032	0.021	0.053	0.019	0.042	0.061	0.007	0.063	0.071
0.2	0.067	0.041	0.108	0.042	0.083	0.125	0.016	0.126	0.143
0.3	0.102	0.062	0.164	0.065	0.125	0.190	0.030	0.188	0.219
0.4	0.139	0.082	0.221	0.094	0.165	0.260	0.037	0.252	0.289
0.5	0.177	0.103	0.279	0.119	0.207	0.326	0.059	0.312	0.371
0.6	0.213	0.123	0.337	0.145	0.248	0.394	0.073	0.374	0.448
0.7	0.252	0.144	0.395	0.171	0.289	0.460	0.090	0.435	0.526
0.8	0.290	0.164	0.454	0.197	0.331	0.528	0.093	0.500	0.593
0.9	0.328	0.185	0.513	0.224	0.372	0.596	0.109	0.561	0.670
1.0	0.367	0.205	0.572	0.250	0.413	0.663	0.123	0.623	0.746
1.1	0.405	0.226	0.631	0.275	0.454	0.730	0.146	0.682	0.828
1.2	0.444	0.246	0.691	0.300	0.496	0.796	0.156	0.746	0.902
1.3	0.486	0.266	0.752	0.324	0.538	0.862	0.171	0.809	0.980
1.4	0.525	0.287	0.812	0.351	0.579	0.930			
1.5	0.564	0.308	0.872	0.383	0.613	0.996			
1.6	0.605	0.329	0.933						

which is also the cycle T, and $f_r \lambda T \overline{x}_{Read}$ for one disk, are listed in Table 8.4. According to experimental data, the first condition is fully satisfied up to the system saturation point.

The second condition is that each disk should be able to process a batch of size $\lambda f_w T$ in a lapse of time less than $T/2$, i.e., $\lambda f_w T \overline{x} \leq T/2$. Table 8.5 lists $\overline{W}$,

Table 8.4 Time Spans for One Disk, R:W = 3:1, $TW = 10$

Arrival Rate	W	$\overline{W}$	$2\overline{W}$	$T = W + \overline{W}$	$f_r \lambda T \overline{x}_{Read}$
0.005	96.52	922418.65	1844837.3	922515.17	3459.43
0.1	95.14	4517.48	9034.96	4612.62	345.95
0.2	94.79	2211.54	4423.08	2306.33	345.95
0.3	94.79	1442.78	2885.56	1537.57	345.95
0.4	94.83	1058.35	2116.7	1153.19	345.96
0.5	94.68	827.88	1655.76	922.56	345.96
0.6	94.80	674.01	1348.02	768.81	345.96
0.7	94.63	564.35	1128.7	658.98	345.97
0.8	94.73	481.89	963.78	576.61	345.97
0.9	94.78	417.77	835.54	512.55	345.97
1	94.75	366.55	733.1	461.30	345.97
1.1	94.67	324.70	649.4	419.37	345.98
1.2	94.75	289.68	579.36	384.42	345.98
1.3	94.59	260.27	520.54	354.86	345.99
1.4	94.70	234.78	469.56	329.48	345.96
1.5	94.76	212.77	425.54	307.53	345.97
1.6	94.72	193.60	387.2	288.32	345.99
1.7	94.79	176.57	353.14	271.36	345.98

i.e., the time between processing Write batches, for one of the mirrored disks with different TW. In experiments, the averages of the time for processing a Write batch are 61.4 ms, 95.5 ms, and 173.8 ms for TW = 6, 10 and 20 respectively. The data exhibited in Table 8.5 indicates that, $\overline{W}$ shortens as the arrival rate increases, and finally becomes much shorter than W, and thus Write queue saturates.

8.2.3 AD-CP Scheme

Two parameters, TR and TW, are introduced by AD-CP, and their effects on performance will be discussed in this section.

Threshold for Reads TR

AD-CP attempts to improve the response time of Reads by introducing a threshold for the number of Reads (TR). When this threshold is reached or exceeded, both disks engage in processing Reads. Figure 8.6 is the performance data for AD-CP scheme, when TR is varied and TW is fixed at 10. R_r can be improved for smaller TR without

Table 8.5 Average Time Span Between Processing Write Batches

Time intervals between batch processing cycles, R:W = 1:1			
Arrival Rate	TW=6	TW=10	TW=20
0.005	275685.366	459477.299	919114.000
0.1	1316.233	2201.488	4421.785
0.2	627.400	1053.156	2124.035
0.3	397.627	670.506	1358.276
0.4	283.261	479.459	975.301
0.5	214.393	364.546	745.464
0.6	168.559	287.927	592.317
0.7	135.567	233.247	482.891
0.8	111.068	192.243	400.744
0.9	92.051	160.309	336.974
1	76.767	134.854	285.834
1.1	64.089	113.992	244.081
1.2	53.679	96.580	209.222
1.3	44.694	81.736	179.734
1.4	37.157	69.098	154.547
1.5		60.010	132.796

sacrificing the maximum throughput, which validates the theory proposed in Section 8.1.2. This experiment was repeated with R:W=1:1 and R:W=1:3, and the results were consistent with the case when Read:Write = 3:1, but the discrepancy in R_r was even less significant than would have been expected.

The AD-CP scheme allows Writes to be individually processed, which considerably enhances flexibility and efficiency, and thus system maximum throughput is also increased. The impact of bringing in individually processing Writes can be perceived from Table 8.6, which lists the utilization up to λ_{max} for one disk in three cases in terms of Reads:Writes ratio; TW is fixed at 10. Write requests are divided according to their current mode when they are processed, i.e., individually or in batch. The time fractions consumed in these two modes are denoted by Write(I) and Write(B), respectively, in Table 8.6.

Two observations can be obtained through comparing Table 8.6 to Table 8.3. First, the disk is more efficiently utilized with AD-CP than with ADU at the same

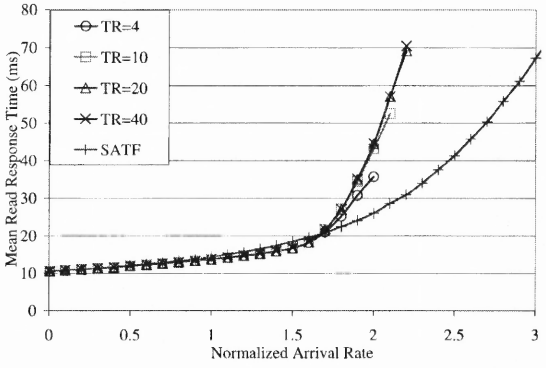


Figure 8.6 AD-CP performance for different TR, TW=10, R:W = 3:1.

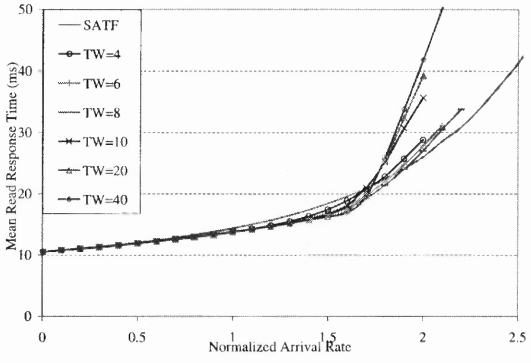


Figure 8.7 AD-CP performance for different TW, TR=4, R:W = 3:1.

arrival rate. For example, when $\lambda_n = 1.0$, the disk utilization with implementing AD-CP and ADU is 0.623 versus 0.572 when Read:Write = 3:1, and 0.782 versus 0.663 when Read:Write = 1:1. AD-CP scheme also improves the maximum throughput in all cases, and the utilization at λ_{max} are all close to 100 percent.

Second, under equal conditions, including Reads:Writes ratio and arrival rate, the time fraction consumed by individually processed Writes for AD-CP, is greater than the time fraction consumed by Writes processed in batch mode for ADU. For example, when R:W=3:1 and λ_n=1.6, the time for processing Writes individually is 0.346 in the AD-CP case, but 0.329 in the ADU case. The reason is that most of the write requests can be processed individually at low arrival rates. Hence few Writes are left to assemble batches. In other word, the chance that the number of pending Writes attains batch size TW is very small.

Table 8.6 Disk Utilization with AD-CP Scheme for Different R:W Ratios, TW=10, TR=10

	R:W=3:1				R:W=1:1				R:W=1:3			
λ_n	Read	W(B)	W(I)	Sum	Read	W(B)	W(I)	Sum	Read	W(B)	W(I)	Sum
0.01	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
0.1	0.018	0.000	0.025	0.044	0.029	0.000	0.051	0.080	0.009	0.000	0.076	0.085
0.2	0.070	0.000	0.051	0.121	0.047	0.000	0.102	0.149	0.020	0.000	0.152	0.172
0.3	0.136	0.000	0.076	0.213	0.029	0.000	0.153	0.182	0.046	0.000	0.228	0.274
0.4	0.089	0.000	0.101	0.190	0.045	0.000	0.203	0.248	0.020	0.000	0.304	0.324
0.5	0.123	0.000	0.126	0.249	0.179	0.000	0.254	0.433	0.055	0.000	0.381	0.436
0.6	0.175	0.000	0.152	0.327	0.096	0.000	0.305	0.401	0.108	0.000	0.456	0.564
0.7	0.212	0.000	0.177	0.389	0.115	0.000	0.355	0.470	0.048	0.000	0.532	0.579
0.8	0.247	0.000	0.203	0.450	0.225	0.000	0.405	0.630	0.088	0.002	0.605	0.695
0.9	0.341	0.000	0.227	0.568	0.254	0.000	0.456	0.710	0.081	0.009	0.674	0.763
1.0	0.370	0.000	0.253	0.623	0.276	0.002	0.504	0.782	0.123	0.039	0.716	0.878
1.1	0.421	0.000	0.278	0.699	0.258	0.011	0.546	0.815	0.135	0.150	0.667	0.952
1.2	0.467	0.000	0.303	0.770	0.293	0.039	0.565	0.897	0.147	0.459	0.390	0.997
1.3	0.489	0.002	0.326	0.817	0.321	0.127	0.514	0.962	0.160	0.504	0.336	1.000
1.4	0.473	0.006	0.348	0.827	0.325	0.333	0.332	0.989				
1.5	0.512	0.018	0.358	0.888	0.339	0.485	0.176	1.000				
1.6	0.542	0.050	0.346	0.939								
1.7	0.566	0.120	0.294	0.980								
1.8	0.567	0.222	0.204	0.993								
1.9	0.561	0.316	0.121	0.998								
2.0	0.548	0.400	0.052	0.999								
2.1	0.534	0.457	0.009	1.000								
2.2	0.539	0.458	0.003	1.000								

Threshold for Writes TW

Figure 8.7 presents the performance data for AD-CP with different TW, when the ratio of Reads to Writes is 3:1. For small Write batch sizes ($TW = 4, 6, 8$ in graph), R_r is improved as TW increases, because more Writes are processed in batch mode, which is optimized in terms of the mean service time. Writes are processed more efficiently, and in turn, the better disk utilization is gained for processing Reads.

However, for large TWs ($TW \geq 10$ in graph), most Writes are processed individually due to the limited chance of forming a big size batch. Since it takes a long time to process a Write batch, there is only one disk available to process Reads, which results in the degradation in R_r.

The maximum throughput is favored by increasing TW. The decrease on λ_{max} from $TW = 8$ to $TW = 10$ is due to the method used for scheduling Writes in a batch, since the exhaustive enumeration was used only when $TW < 10$, while SPTF was used when $TW \geq 10$.

8.2.4 An Overall Comparison

Finally, the overall performance evaluation for the aforementioned schemes is displayed in Figures 8.8 and 8.9, with the FCFS policy used as a baseline. SPTF-CP(t) performance is not reported and is discussed in Section 7.5.

As far as R_r is concerned, the performance of ADU is comparable or even worse than FCFS; both produce poor performance at the higher arrival rates. AD-CP is comparable to HOL, and slightly outperforms HOL when Reads constitute a larger fraction.

As far as the maximum throughput is concerned, ADU is comparable to FCFS and HOL, with unnoticeable improvement. The maximum throughput of AD-CP is much higher than that of ADU, but still inferior to SPTF.

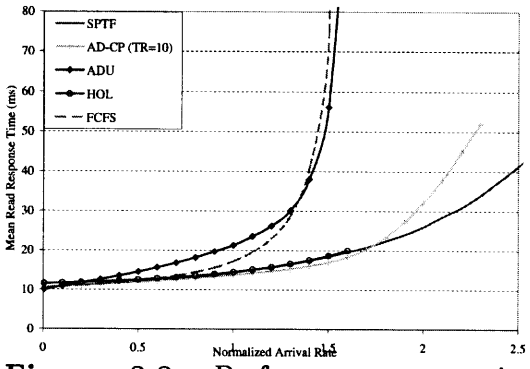


Figure 8.8 Performance comparison, $TW = 6$ with ADU and AD-CP, R:W=3:1.

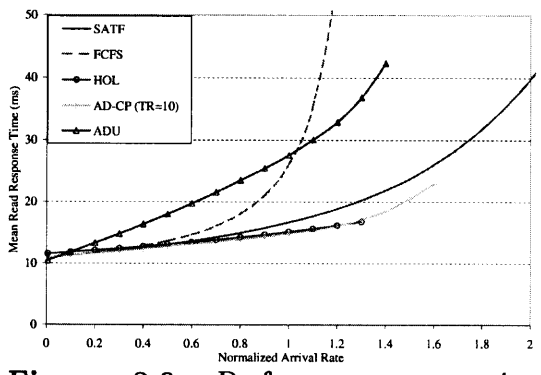


Figure 8.9 Performance comparison, $TW = 6$ with ADU and AD-CP, R:W=1:1.

8.3 Summary of Conclusions

The undertaking of this study was influenced by the marketplace popularity of mirrored disks or RAID1, as opposed to more modern RAID5 disk arrays, which has lower redundancy but poorer performance for OLTP applications. Non-volatile storage allows fast writes, so that the destaging of dirty blocks to disk can be deferred and be carried asynchronously through efficient batch processing. This study is based on the assumption that Reads are held in a shared queue and can be processed on any disk.

The starting point was a scheduling scheme described in [14], which is referred to as ADU scheme in this dissertation. An extension to this method, AD-CP, is then proposed to improve its performance and the improvement has been shown via simulation results. These schemes are compared against FCFS, HOL and SPTF.

ADU shows improvement on FCFS, and HOL to a small degree, in the maximum attained throughput. However, ADU is outperformed by AD-CP and SPTF-CP. It is interesting to note that SPTF attains the highest throughput due to its ability to reduce the mean service time as the queue-length increases. SPTF-CP provides a tuning parameter to prioritize the processing of Reads, so that there is a trade-

off between the response time and the maximum attained throughput. This method, unlike others, has a shortcoming in that it does not pay attention to reducing the space for caching dirty blocks, while this issue is handled automatically by defining a parameter for processing Writes in batch (in ADU and AD-CP).

CHAPTER 9

PERFORMANCE AND RELIABILITY OF RAID1 WITH DECLUSTERING

RAID1, also known as *disk mirroring*, provides high performance and reliability by making data highly available, hence introducing various forms of data redundancy. Traditional RAID1 configuration typically fills each disk with consecutive data before switching to the next pair. As a result, the failure of one disk will cause the Read workload to double on the surviving disk. Therefore, several variations based on the basic configuration have been designed to balance the workload in both normal and degraded modes. Striping is a technique which not only provides better performance, but also enables workload balancing. Combining striping with RAID1 is a common scheme utilized in the variations that are under examination.

This chapter specifies three variations along with the basic RAID1 configuration, and compares their load distribution and reliability in both the normal and the degraded modes[1]. The discussion is based on the assumptions that, the local scheduling policy is FCFS, and each data pair shares a common queue.

9.1 Model and System Descriptions

This section presents the model of the disk mirroring system, specifies the parameters used in analysis, and describes various RAID1 configurations with a fixed disk array framework.

9.1.1 Model Definition

RAID1 system maintains two physical copies of data: *primary* data copy and *secondary* data copy, denoted as X and X', respectively. The *secondary* data also stands

[1]This is an extension of the joint work with A. Thomasian.

for a "backup copy" in this study. A RAID1 system with N disks ($N = 8$ in the following analysis) is under consideration; for the configurations concerning clusters, the cluster size, i.e., the number of disks that constitutes a cluster, is defined as n. Specific system parameters are as follows:

- Total number of disks: N.
- Number of disks in a cluster: n.
- Number of clusters: $c = N/n$.
- Fraction of read requests: f_R.
- Fraction of write requests: $f_W = 1 - f_R$.
- Fraction of read requests processed at disk holding primary data: α.
- Arrival rate of read requests to *each disk pair*: λ_R.
- Arrival rate of write requests to *each disk pair*: λ_W.
- Total load of read requests *at each disk*: Λ_R.
- Total load of write requests *at each disk*: Λ_W.
- Total load of each disk: $\lambda_{disk}(= \Lambda_R + \Lambda_W)$.
- Total load of i_{th} disk: $\lambda_i(= (\Lambda_R)_i + (\Lambda_W)_i)$.
- Mean service time for single read request: $\overline{x}_R$.
- Mean service time for single write request:$\overline{x}_W$.

9.1.2 Various Configurations of RAID1 Declustering

Basic mirroring: Disk $2i$ mirrors disk $2i - 1$, $1 \leq i \leq n$, and vice versa, as shown in Figure 9.1. The disks are identical to each other within a mirrored pair, so the Read load is assumed to be evenly distributed to either disk, i.e., $\alpha = 1/2$.

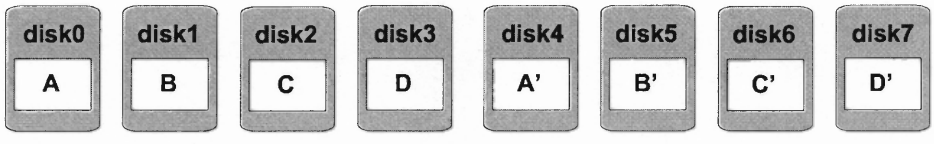


Figure 9.1 Basic mirroring.

Interleaved declustering[42, 43]: The primary and secondary copies of data are stored within an n-disk ($n \geq 2$) cluster. The secondary data copy is evenly striped

across all the remaining disks in the cluster. Figure 9.2 illustrates the case for $n = 4$, and Figure 9.3 for $n = 8$.

α of Reads are processed at the disk which maintains the primary data copy, and $(1 - \alpha)$ of Reads are evenly routed to the other disks in the cluster.

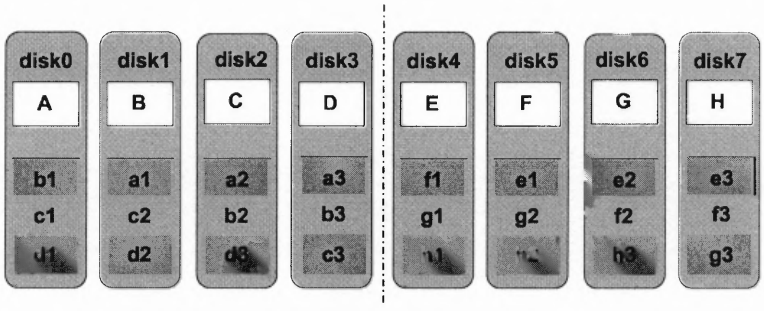


Figure 9.2 Interleaved declustering, $n = 4$.

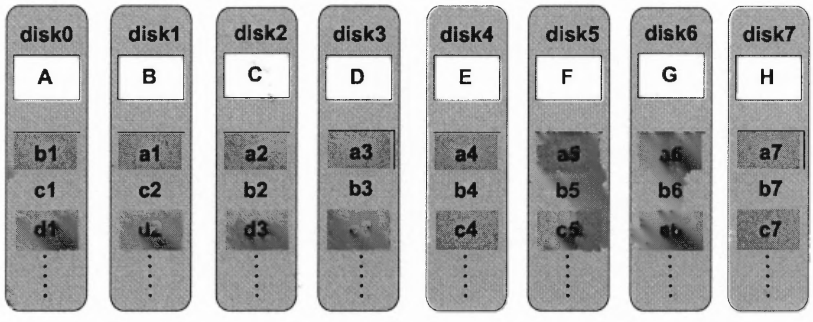


Figure 9.3 Interleaved declustering, $n = 8$.

Chained declustering [44]: As shown in Figure 9.4, the data on each disk is replicated on the next disk (modulo the number of disks). In normal mode, Reads are routed to the primary data copy and write operations update both copies.

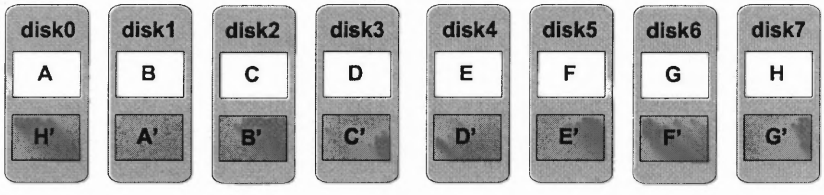


Figure 9.4 Chained declustering.

Group rotate declustering [45]: Group rotate declustering maintains the primary and secondary data copies in separated clusters of disks, as shown in Figure

9.5. The primary data copy is striped across n disk in a cluster, and these striped data are duplicated as the secondary data copy and stored in another n-disk cluster in a rotated manner.

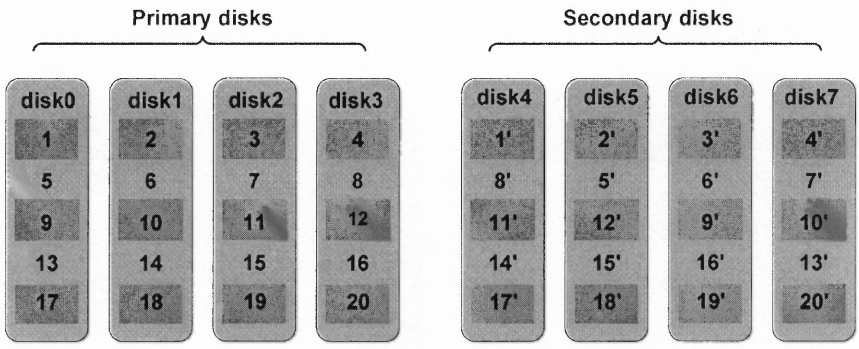


Figure 9.5 Group rotate declustering. X = Primary data, X'= Duplicated data.

9.2 Load Distribution Comparison

Examining the load distribution can effectively deduce both the maximum throughput and the capability of load balancing of the system. The load distribution comparison is performed based on a RAID1 system with the following assumptions: (i) there are $N = 8$ disks under consideration; (ii) each disk maintains both primary and secondary data copies, unless otherwise noted; (iii) the cluster size $n = 4$.

9.2.1 Basic Mirroring

Normal Mode

Referring to Figure 9.1, the load of each disk comprises half of the Read load and the complete Write load:

$$\lambda_{disk} = \Lambda_R + \Lambda_W = (1/2\lambda_R + 1/2\lambda_R) + (\lambda_W + \lambda_W) = \lambda_R + 2\lambda_W. \qquad (9.1)$$

Note that each disk maintains both the primary data and the secondary (back-up) data of the counterpart disk.

Degraded Mode

The failure of one disk in a mirrored pair does not affect other pairs. For the surviving disk corresponding to the failed one, the Read load is doubled and Write load remains:

$$\lambda_{disk} = 2\lambda_R + 2\lambda_W. \tag{9.2}$$

The load of the disks in other mirrored pairs remains unchanged, and hence results in unbalanced workloads across all of the remaining $N - 1$ disks.

9.2.2 Interleaved Declustering

Normal mode

On the assumption that α of Reads access primary data copy (see Figure 9.2), the Read load of each disk comprises two parts: the accesses to the primary data; and the accesses to the secondary data that corresponds to the primary data stored on the other disks within the same cluster:

$$\Lambda_R = \alpha\lambda_R + (n-1)\frac{(1-\alpha)\lambda_R}{n-1} = \lambda_R.$$

Similarly, the Write load of each disk is:

$$\Lambda_W = \lambda_W + (n-1)\frac{\lambda_W}{n-1}.$$

The total load of each disk is the sum of the above two parts:

$$\lambda_{disk} = \Lambda_R + \Lambda_W = \lambda_R + 2\lambda_W. \tag{9.3}$$

Degraded mode

Interleaved declustering allows only one disk failure within each cluster. For each of the surviving disks, the Read load comes from four sources:

i accesses to the secondary data corresponding to the failed disk: $\lambda_R/(n-1)$

ii accesses to the primary data: $\alpha\lambda_R$

iii accesses that cannot be routed to the secondary data copy on the failed disk: $(1-\alpha)\lambda_R/(n-1)$

iv accesses routed from other surviving disks: $(n-2)\frac{(1-\alpha)\lambda_R}{n-1}$

The total Read load of each disk is the sum of the above four:

$$\Lambda_R = \frac{\lambda_R}{n-1} + \alpha\lambda_R + \frac{(1-\alpha)\lambda_R}{n-1} + (n-2)\frac{(1-\alpha)\lambda_R}{n-1} = \frac{n}{n-1}\lambda_R.$$

The Write load of each disk is:

$$\Lambda_W = \lambda_W + (n-2)\frac{\lambda_W}{n-2} = 2\lambda_W.$$

The total load of each surviving disk within a cluster is:

$$\lambda_{disk} = \Lambda_R + \Lambda_W = \frac{n}{n-1}\lambda_R + 2\lambda_W. \tag{9.4}$$

9.2.3 Chained Declustering

Normal mode

In normal mode, Reads are routed to the disk that holds primary data, and Writes are required to update both data copies. The load of each disk is:

$$\lambda_{disk} = \Lambda_R + \Lambda_W = \lambda_R + 2\lambda_W. \tag{9.5}$$

Degraded mode

As shown in Figure 9.4, chained declustering simply backups the data and stores it on the disk next to which the primary data is stored, rather than having the secondary data spread over multiple disks. Reads access only the primary data copy in the normal mode. Therefore, in the degraded mode, a disk failure does not cause the change in Write load of surviving disks. For simplification, a read-only workload is assumed in the analysis. Denote λ_i to be the load of $disk_i$ in normal mode, and λ_i' in degraded mode. Assume the load of each disk is identical: $\lambda_0 = \lambda_1 = \lambda_2 = \cdots = \lambda_{n-1}$.

Chained declustering does not tolerate contiguous disk failures, and the maximum number of failed disks, which will not cause data loss, is $\lfloor n/2 \rfloor$. Load balancing is not achievable in all the cases when system sustains in degraded mode. The case-by-case discussions are as follows:

Single disk failure: Load balancing is easily achieved across all the surviving disks. Each of the disks serves $N/(N-1)(N = n$ in this case) of the total load. For example, if $disk_0$ fails, the load distribution over all the surviving disks is: $\lambda_1' = \lambda_0 + \frac{1}{n-1}\lambda_1$, $\lambda_2' = \frac{n-2}{n-1}\lambda_1 + \frac{2}{n-1}\lambda_2, \ldots$; so that the load of $disk_i$ is

$$\lambda_i = (\Lambda_R)_i = \frac{n-i}{n-1}\lambda_{i-1} + \frac{i}{n-1}\lambda_i. \tag{9.6}$$

Double disk failures:

Due to the fact that two consecutive disk failures will cause data loss, the analysis assumes $disk_0$ and $disk_k$ fail, where $k \neq 1$ and $n-1$.

However, load balancing cannot be achieved in all the cases without data loss. For example, if $disk_0$ and $disk_2$ fail, the load of $disk_1$ will be doubled while not necessary for other surviving disks. This is because $disk_1$ maintains the data copies B and A', both of whose counterparts are on failed disks.

Load balancing with two disk failures occurs only when each of the remaining disks serve $n/(n-2)$ of the load: $\lambda_1' = \lambda_0 + \frac{2}{n-2}\lambda_1, \lambda_2' = \frac{n-4}{n-2}\lambda_1 + \frac{4}{n-2}\lambda_2, \ldots$; however, because $disk_k$ fails and consequently can not take over the load of $disk_{k-1}$, $disk_{k-1}$ services all the requests accessing its primary data: $\lambda_{k-1} = \frac{2}{n-2}\lambda_{k-2} + \frac{n-2}{n-2}\lambda_{k-1}$. It other words, load balancing is achieved only when the surviving disks between $disk_0$ and $disk_k$ can accomplish the load of k disks: $k\lambda = (k-1)\frac{n}{n-2}$, and thus $k = n/2$.

In such a load-balancing case, the load of each surviving disk is $\lambda_{disk} = \Lambda_R = \frac{n}{n-2}\lambda_0$; the load of $disk_i$ $(0 \leq i \leq k)$ is:

$$\lambda_i = (\Lambda_R)_i = \frac{n-2i}{n-2}\lambda_{i-1} + \frac{2i}{n-2}\lambda_i. \tag{9.7}$$

Multiple disk failures:

For multiple k disk failures, $k \geq 2$, a conclusion can be easily derived from the above analysis that, load balancing is achieved only when $k = 2^m, (m \geq 1)$. For the example shown in Figure 9.4, the extreme load-balancing case is four disk failures, in which the load of each surviving disk doubled.

9.2.4 Group Rotate Declustering

Normal mode

Assume α of the read requests are routed to the primary disks. Similar to interleaved declustering, the Read load of each disk consists of the accesses to the primary data and the accesses to the secondary data corresponding to its counterpart stored in the other cluster (of disks):

$$\Lambda_R = \alpha\lambda_R + (1-\alpha)\lambda_R = \lambda_R.$$

The Writes to each disk update both the primary and secondary data copies:

$$\Lambda_W = \lambda_W + \lambda_W = 2\lambda_W.$$

The total load of each disk is the sum of the above two:

$$\lambda_{disk} = \Lambda_R + \Lambda_W = \lambda_R + 2\lambda_W. \tag{9.8}$$

Degraded mode

Group rotate declustering sustains multiple disk failures only when the failed disks are in one cluster, i.e., either primary disks or secondary disks, but not both. In the analysis, disk failure is assumed to occur at the primary disks.

In order to achieve load balancing across the surviving disks, the load proportion between primary disks and secondary disks, α, is varied along with the number of failed disks. To simplify the calculation, assume each cluster maintains only one type of data copy, i.e., either primary data copy or secondary data copy. A cluster of disks that maintain the primary/secondary data copy are denoted as *primary/secondary disks*, as shown in Figure 9.5. The correctness of the assumption is accomplished by the fact that the two clusters are symmetric to each other.

The analyses, based on the system with $N = 8$ disks and cluster size $n = 4$, are as follows:

Single disk failure

The load of failed disk is evenly distributed to all the secondary disks. For each secondary disk, the Read load includes the accesses corresponding to the surviving $n - 1$ primary disks, and $1/n$ of the accesses from failed disk:

$$\Lambda_R = (n-1)\frac{(1-\alpha)\lambda_R}{n} + \frac{\alpha}{n}\lambda_R = \lambda_R(1-\alpha+\frac{\alpha}{n}).$$

To achieve load balancing, the load between the primary and secondary disks must be equal: $\alpha\lambda_R + \lambda_W = \lambda_R(1-\alpha+\frac{\alpha}{n}) + \lambda_W$, and thus $\alpha = n/(2n-1)$. For $n = 4$ and $N = 8$, $\alpha = 4/7$.

The load of each surviving disk is:

$$\lambda_{disk} = \frac{n}{2n-1}\lambda_R + \lambda_W. \tag{9.9}$$

Multiple disk failures

Similarly, to balance the load between the surviving primary disks and secondary disks, α is adjusted according to

$$\alpha\lambda_R + \lambda_W = (n-k)\frac{(1-\alpha)\lambda_R}{n} + \frac{k\lambda_R}{n} + \lambda_W, (k \leq n)$$

and thus $\alpha = n/(2n-k)$.

The load of each surviving disk is:

$$\lambda_{disk} = \frac{n}{2n-k}\lambda_R + \lambda_W. \tag{9.10}$$

In an extreme case in which $k = n$, the Read load of each secondary disk doubled.

9.3 Reliability Comparison

This section estimates the reliability of various RAID1 configurations by using the *probability of data loss* for each level in degraded mode.The estimates will be given based on the system described in Section 9.2. Let P_k denote the probability that k disk failures will not cause data loss.

9.3.1 Basic Mirroring

Basic mirroring can tolerate up to $N/2$ disk failures, as long as the failed disks are in different mirrored pairs. Denote m to be the number of pairs in an N-disk basic mirroring system, so that $m = N/2$.

In general, there are $\binom{N}{k}$ ways for k disk failures; the number of ways that k disk failures do not cause data loss is $\binom{m}{k}2^k$. Thus the reliability of basic mirroring is:

$$P_k = \frac{\binom{m}{k}2^k}{\binom{N}{k}} \tag{9.11}$$

9.3.2 Interleaved Declustering

Interleaved declustering distributes the load of a failed disk evenly to the remaining disks within a cluster. So the probability that one disk failure does not incur data loss $P_1 = 1$.

Next, consider the case of two disk failures, given that there are n disks per cluster, so that the number of clusters $c = N/n$.

There are $\binom{N}{2}$ ways for two disks to fail. After the first disk failure, which has N possibilities, the number of ways that the second disk can fail in other clusters is $N - n$. So the total number of ways that two disk failures are in different clusters is $N(N-n)/2$. Thus the probability that two disk failures do not lead to data loss is

$$\frac{N(N-n)/2}{\binom{N}{2}} = \frac{N-n}{N-1}.$$

More generally, when $k \leq c$ disks fail, data loss does not occur as long as the failed disks are in different clusters. There are $\binom{N}{k}$ ways for k disk failures; the number of possibilities of no data loss is $(N-n)\cdot(N-2n)\cdots(N-(k-1)n)$. Thus the probability of system survival after k disk failures is:

$$\begin{aligned} P_k &= \frac{N(N-n)\cdot(N-2n)\cdots(N-(k-1)n)/k!}{\binom{N}{k}} \\ &= \frac{n^k\frac{N}{n}\cdot(\frac{N}{n}-1)\cdot(\frac{N}{n}-2)\cdots(\frac{N}{n}-(k-1))/k!}{\binom{N}{k}} \\ = \frac{n^k c\cdot(\frac{N}{n}-1)\cdot(c-2)\cdots(c-(k-1))/k!}{\binom{N}{k}} &= \frac{\binom{c}{k}n^k}{\binom{N}{k}}. \end{aligned} \tag{9.12}$$

Note that this is smaller than basic disk mirroring.

9.3.3 Chained Declustering

As described in Section 9.2.3, chained declustering can still tolerate one disk failure, so the probability of no data loss $P_1 = 1$; chained declustering system can sustain up to $\lfloor 2/n \rfloor$ ($n = N$ in this case) failures.

The probabilitiy of k disk failures without data loss will be calculated for each case.

Single disk failure:

$$P_1 = 1. \tag{9.13}$$

Double disk failures:

There are $\binom{n}{2}$ ways for two disk failures; the number of ways that two contiguous disks can fail is n. Thus

$$P_2 = 1 - \frac{n}{\binom{n}{2}}. \tag{9.14}$$

Three disk failures:

There are $\binom{n}{3}$ possible ways for three disk failures, Any two contiguous disk failures would result in data loss, which includes two possible cases: three failed disks are consecutive; two failed disks are consecutive, but the third one is one or more surviving disks apart.

The number of ways that three contiguous disks fail is n; the number of ways that two contiguous disks fail excluding three consecutive failures is $n(n-2-2)$, which is elaborated as follows: the number of two contiguous disk failures is n; the third failed disk is not possible adjacent to those two, so that the third failure is only possible to occur on the remaining $n-2-2$ disks. Thus

$$P_3 = \frac{\binom{n}{3} - n - n(n-4)}{\binom{n}{3}}. \tag{9.15}$$

Four disk failures:

Similar to the above analysis, there are $\binom{n}{4}$ ways for four disks to fail; the number of ways that four failed disks are consecutive is n; the number of ways that three consecutive disks fail excluding four consecutive failures is $n(n-3-2)$; the number of ways that two consecutive disks fail excluding three or four consecutive failures is $n\binom{n-2-2}{2} - \frac{n(n-4-1)}{2}$ (two pairs of consecutive failed disks are counted twice). Thus

$$P_4 = \frac{\binom{n}{4} - n - n(n-3-2) - n\binom{n-2-2}{2} + \frac{n(n-4-1)}{2}}{\binom{n}{4}}. \tag{9.16}$$

9.3.4 Group Rotate Declustering

As explained in Section 9.2.4, group rotate declustering tolerates multiple disk failures, as long as the failed disks are within one cluster, i.e. either primary disks or secondary disks. Given the assumption of disk failure(s) occurring on primary disk(s), the reliability analysis is performed as follows.

The probability that single disk failure does not incur data loss $P_1 = 1$. When $k(k \geq 2)$ disks fail, there are $\binom{N}{k}$ ways for this to happen; while there are only $\binom{n}{k}$ out of them that k disks fail within one cluster, which does not incur data loss. There are $c(c = N/n)$ clusters in total, so the number of ways that k disks can fail within a cluster and hence without data loss is $c\binom{n}{k}$. Thus

$$P_k = \begin{cases} 1 & k = 1 \\ \frac{\frac{N}{n}\binom{n}{k}}{\binom{N}{k}} = \frac{c\binom{n}{k}}{\binom{N}{k}} & 1 < k \leq n. \end{cases} \tag{9.17}$$

9.4 Summary of Conclusions

Two architectural techniques are applied in redundant disk arrays: data striping, for improved performance, and redundancy, for improved reliability. Between the two most popular RAID organizations, trends in disk technology has made RAID1, also known as disk mirroring, more preferable than RAID5, because the high data availability in RAID1 enables an improved performance of randomly accessing small data blocks, which is typical in OLTP applications.

A drawback of basic disk mirroring is the possibility of a data access skew in both normal and degraded modes. This can be solved by including data striping. Several variations with data striping have been proposed in the past years; this chapter has reviewed three of them: interleaved declustering, chained declustering and group rotate declustering, and compares their capability of load balancing and reliability to the basic RAID1 configuration. The results are summarized as in Table 9.1.

Table 9.1 Capability of Load Balancing and Reliability Comparison

Configuration type	Basic	Interleaved	Chained	Group rotate
Load after one disk failure	Not balanced	Balanced	Balanced	Balanced
Ease of balance	Not applicable	Easy	Limited	Easy
Maximum number of failed disks	N/2	one per cluster	N/2	N/2
Prob[k disk failures without data loss]	equation 9.11	equation 9.12	equations 9.13 to 9.16	equation 9.17

CHAPTER 10

CONCLUSION

This dissertation presents the impact of disk scheduling on the secondary storage systems. Heavy workloads create long queues of pending requests on the disk side; the disk arm is a valuable resource for improving disk or system performance, by reordering the processing of the pending requests according to certain criteria.

A detailed description of the disk structure was demonstrated, together with increasingly complex data layouts resulting from updated technologies. A rather sophisticated disk simulator, which has taken into account the above information, was developed for both analytic and simulation studies. Experiments in this study utilized both synthetic and traced workloads. When priority becomes one of the considerations, the requests are simply classified into two levels: read requests, which have a higher priority than write requests.

A survey of existing scheduling algorithms for single disk and mirrored disks was conducted. Extensions of single disk scheduling were proposed after a close study of the existing algorithms.

A mathematical queueing model, based on some favorable assumptions, was established for the disk system, in order to theoretically verify the previous work and obtain a starting point for later studies. Two analytic methods, used to evaluate SATF performance by calculating the mean request response time, were specified. A valuable by-product was produced in the experiments: the normalized mean disk access time is inversely proportional to the fifth root of the number of requests on the disk.

SATF has been proved to outperform other classical scheduling policies in many studies; this dissertation validates the conclusion by simulation. Several vari-

ations of SATF were reviewed and re-evaluated by experiments with both synthetic and traced workloads. Results have shown that these variations tend to outperform SATF in certain aspects, e.g. throughput, under particular circumstances, such as intensive workload bursts. Three new variations were then defined: SATF-LAi, designed to reduce the variance of the request response times, also shows significant improvement in mean request response time when the workload accesses data in a more clustered manner; SAPTF adjusts the priority of each request by considering its waiting time, and shows remarkable improvement in the 95^{th} percentile of response times; SPTF-CP prioritizes the low-priority requests with long waiting time by multiplying a reducing factor $t(0 < t < 1)$, and the results exhibit an exact match between synthetic and real workloads in selecting the optimum value for t.

The study in mirrored disk system was started with a discussion on the system configuration. A comprehensive classification of routing schemes and disk scheduling policies was then presented, in order to facilitate performance evaluation. A simulation with synthetic workloads was conducted to assess various combinations of scheduling and routing. SQ configuration displays advantage over IQ in both request response time and system throughput, which is consistent to the theoretical queueing analysis. Under the same configuration, the routing schemes which exploit the information of current system state, demonstrates better performance than the schemes that route requests in a mechanical way. Affinity-based routing improves performance only when the FCFS is applied as local scheduling policy.

The study of a mirrored disk system with NVS cache was driven by the paper by Polyzois et al. [14]. Write requests are destaged and processed in batch, while there is always at least one disk available to process read requests. Their scheme is improved by refining the scheduling policy for processing Writes in a batch, and setting a threshold for prioritizing the processing of Reads.

Traditional disk mirroring enables the improvements in both performance and system reliability via high data redundancy. However, load balancing is hard to achieve in basic mirroring, so that many variations were developed to address this problem. This dissertation describes three representative variations, and compares their capability of load balancing and reliability. Analysis has shown that "group rotate declustering" demonstrates an ease of implementation, a good behavior of balancing load and robustness to tolerate multiple disk failures.

APPENDIX A

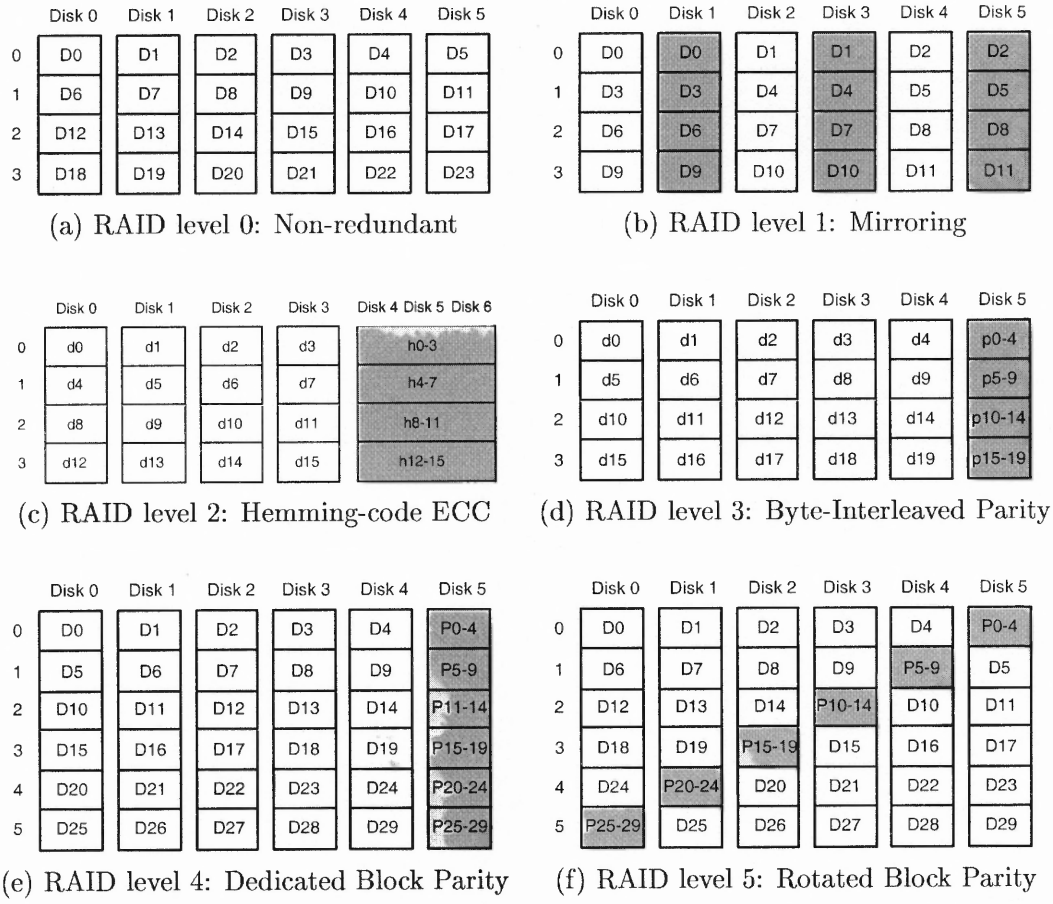
RAID ORGANIZATION LEVELS

RAID level 0 is often used to indicate a non-redundant disk array with striping. Briefly, RAID1 is block interleaved dedicated mirroring. RAID2 is bit or byte-interleaved and uses Hamming error correcting code [46]. RAID3 is byte-interleave parity with one disk dedicated to parity. RAID4 is block-interleave parity with one disk dedicated to parity. RAID5 is rotated block-interleaved parity with the parity blocks distributed over all disks. In Figure A.1, the data and redundancy information organizations for RAID levels 0 through 5 are illustrated. The RAID5 design shown uses *left-symmetric* organization [47], which is first placing the parity stripe units on the diagonal and then placing consecutive data stripe units on consecutive disks.

Among those RAID levels, RAID 2 and 4 are of less interest. RAID2 uses Hamming code, which introduces higher redundancy than necessary. RAID4 differs from RAID3 only in that it is block-interleaved. There is a load imbalance problem for RAID4, since the disk that is dedicated to parity can be overloaded if a large fraction of requests are Writes. RAID5 offers a better solution by distributing the parity stripe units over all disks, such that the load is balanced and all disks can contribute to the read throughput. RAID3 is suitable for the special scenario when the disk array is dedicated to a single application and the process demands large amount of data at high bandwidth.

The concept of disk array offers a solution for highly reliable parallel data storage. For single disk tolerant disk arrays, the reliability can be measured in the form of *mean time to data loss(MTTDL)*. A simple expression for the MTTDL for a redundant disk array that can tolerate one disk failure is given in [10]:

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



(a) RAID level 0: Non-redundant

(b) RAID level 1: Mirroring

(c) RAID level 2: Hemming-code ECC

(d) RAID level 3: Byte-Interleaved Parity

(e) RAID level 4: Dedicated Block Parity

(f) RAID level 5: Rotated Block Parity

Figure A.1 Data layout in RAID levels 0 through 5. The shaded blocks are parities. d_i means data are bit or byte interleaved over disks. D_i means data are block interleaved. p_{i-j} means the parity is computed over d_i through d_j, P_{i-j} is defined similarly.

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

APPENDIX B

QUEUEING MODEL FACILITIES

B.1 Little's Theorem

Little's Theorem is one of the most important theorems in Queuing Theory. The statement of Little's Theorem is:

$$N = \lambda T. \tag{B.1}$$

where N is the average number of customers in the system, λ is the average arrival rate into the system, and T is the average amount of time a customer spends in the system.

Little's theorem can be applied in almost any system or part of it.

B.2 Poisson Process

One of the most frequently used model for the arrival processes is the Poisson process. It has been mathematically shown that the merging of a large number of statistically independent arrival processes gives a Poisson process asymptotically. Poisson process is used to study the most basic queueing system: M/M/1 queues.

The probability density distribution for a Poisson process is shown as Equation B.2, which describes the probability of seeing n arrivals in a period from 0 to t.

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t} \tag{B.2}$$

where t is used to define the interval 0 to t; n is the total number of arrivals in the interval 0 to t; λ is the average arrival rate.

Therefore, the distribution of the interarrival times follows exponential distribution:

$$P\{X > t\} = P_0(t) = e^{-\lambda t} \tag{B.3}$$

where X is the time interval between two consecutive arrivals.

B.3 M/G/1 Queuing Formulas

$$\begin{cases} M: & \text{Poisson arrival process, intensity } \lambda \\ G: & \text{general holding time distribution, mean } 1/\mu \\ 1: & \text{single server, load } \rho = \lambda/\mu \end{cases}$$

An M/G/1 queueing system can be used to derived the Pollaczek-Khinchin (P-K) formula, which determines the average time a customer spends in the system. Most of the equations in this section can be found in [48]. First define b_i to be the ith moment of service time, λ to be the arrival rate, and $\rho = \lambda b_1$ to be the utilization factor. The P-K formula is:

$$\overline{W} = \frac{\lambda b_2}{2(1-\rho)}, \tag{B.4}$$

where $\overline{W}$ is the mean waiting time of a customer in a queue.

The second moment of the waiting time is:

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

The moments of response time:

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

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

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

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

B.4 Non-Preemptive Priority Queuing

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

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

P queues for each priority level,

Jobs at each level are served in FCFS order,

Jobs in class p have an arrival rate λ_p,

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

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

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

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

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

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

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

The overall waiting time and response time are

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

let

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

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

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

B.5 Some Important Distributions

Erlang Distribution

A random variable X has an Erlang-k distribution if X is the sum of k independent random variables having a common exponential distribution with mean $1/\lambda$. The probability density function is:

$$f(t) = \frac{\lambda^k t^{k-1} e^{-\lambda t}}{(k-1)!} \quad t > 0, \quad \lambda > 0, \quad k = 1, 2, \ldots \tag{B.5}$$

The mean, variance and squared coefficient of variation are as follows:

$$E[X] = \frac{k}{\lambda}, \quad \sigma^2[X] = \frac{k}{\lambda^2}, \quad c_X^2 = \frac{1}{k}$$

When the CR (cyclical routing) scheme is implemented in a mirrored disk system with independent queues, the request interarrival times to *each disk* have an Erlang-2 distribution.

Hyperexponential Distribution

A random variable X is hyperexponentially distributed if X is chosen with probability $p_i, i = 1 \ldots k$ from a mixture of k exponential random variables. The probability density function is:

$$f(t) = \sum_{i=1}^{k} p_k \lambda_i e^{-\lambda_i t} \quad t > 0, \quad \lambda > 0, \quad k = 1, 2, \ldots \tag{B.6}$$

The mean is equal to

$$E[X] = \sum_{i=1}^{k} \frac{p_k}{\lambda_k},$$

and the coefficient of variation c_X is always greater than or equal to 1.

When the UR (uniform routing) scheme is implemented in a mirrored disk system with independent queues, *each disk* has an equal probability to be sent the requests whose interarrival times have an exponential distribution. In other words, each

disk chooses requests from two exponentially distributed queues with equal probabilities, and the arrival rate to each disk is half of the total arrival rate to the system.

APPENDIX C

SCHEDULING SCHEMES FOR BATCH PROCESSING

An exhaustive enumeration is desirable to determine an optimal schedule for the processing of Writes. This is computationally feasible for smaller values of TW, where the minimum schedule (makespan) among all possible is selected.

For larger values of TW a heuristic algorithm should be used to obtain a near optimal schedule. Rather than CSCAN utilized in [14], the SATF algorithm was employed in the AD-CP scheme to minimize the schedule. There have been several investigations on this topic such as [49], [50], but this study aims to find a quick and easy approach to be implemented. Of course if the best heuristic provides a poor performance compared to the optimum, which is unfortunately only known for smaller values of TW, then smaller values of TW should be selected. On the other hand it is intuitively clear that more efficient scheduling is possible for larger values of TW.

Table C.1 Mean Service Time for Write Batches with SATF Scheduling and An Optimal Ordering

TW	$T_{batch,SATF}$	$T_{batch,Optimal}$	ratio	$\overline{X}_{service,SATF}$	$\overline{X}_{service,Optimal}$
5	47.49	45.33	1.05	9.50	9.07
6	55.21	52.07	1.06	9.20	8.68
7	61.65	58.10	1.06	8.81	8.30
8	69.53	64.85	1.07	8.69	8.11
9	76.13	71.13	1.07	8.46	7.90
10	83.80	77.36	1.08	8.38	7.74

An experiment for a preliminary investigation was initialized by generating n requests randomly placed on the disk. Assuming that the disk arm is initially at block number zero, makespan is determined for all $n!$ permutations; the optimal one and the ordering generated according to SATF were reported. The experiment was

repeated for 100 times and the averages are listed in Table C.1. The averages of the mean service times, $\overline{X}_{service} = T_{batch}/TW$, with two scheduling methods are also presented. The $\overline{X}_{service}$ is reduced as the batch size increases in both cases. The ratio $T_{batch,SATF}/T_{batch,Optimal}$ demonstrates that SATF provides schedules quite close to the optimal ordering.

Figure C.1 displays the effect of TW on $\overline{X}_{service}$ with the SATF policy in an extended range. The drop in $\overline{X}_{service}$ slows down beyond $TW = 50$ and approximates to one half of its value.

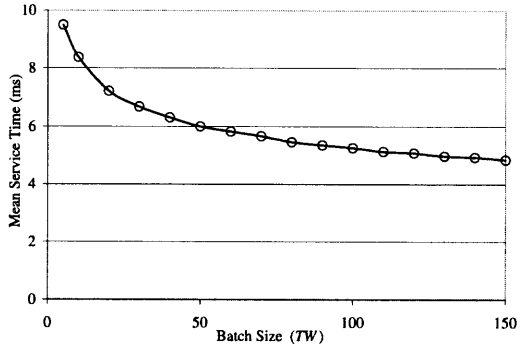


Figure C.1 Mean service time with SATF scheduling policy

APPENDIX D

SCHEDULING SCHEME FLOWCHARTS

D.1 ADU Flowcharts

The flowcharts in Figure D.1 depict the logical steps to perform the Alternating Deferred Updates (ADU) [14] scheme in a mirrored disk system with an NVS cache (see Section 8.1.1). The specifications for the flowcharts are listed in Table D.1.

Table D.1 Specification for ADU Scheme Flow Charts

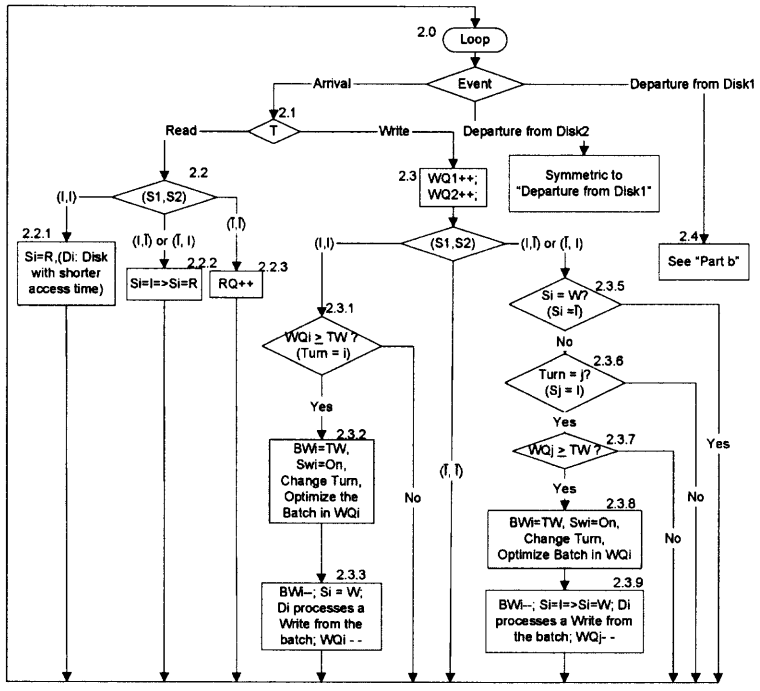
Simulation related entities
Event$\in$ {Arrival, Departure form Disk1, Departure from Disk2} T$\in$ {Read, Write} – request type D$\in$ {D1, D2} – disk number
Scheduling Parameters:
NC - Number of completed requests TW - Write batch size
System state and its components:
State $(S1, S2, RQ, WQ1, WQ2, BW1, BW2, Sw1, Sw2)$ $Si \in$ {R, W, I} R-disk being read, W-disk being written, I-idle disk (disk states) RQ - Number of read requests in the shared queue WQi - Number of write requests enqueued for Di BWi - Number of write requests left in the batch being processed by Di $Swi \in$ {On, Off} - Switches for Write batches: On-disk is processing a batch; Off-otherwise $Turn \in \{1, 2\}$ - Switch to assign a disk to process a batch

D.2 AD-CP Flowcharts

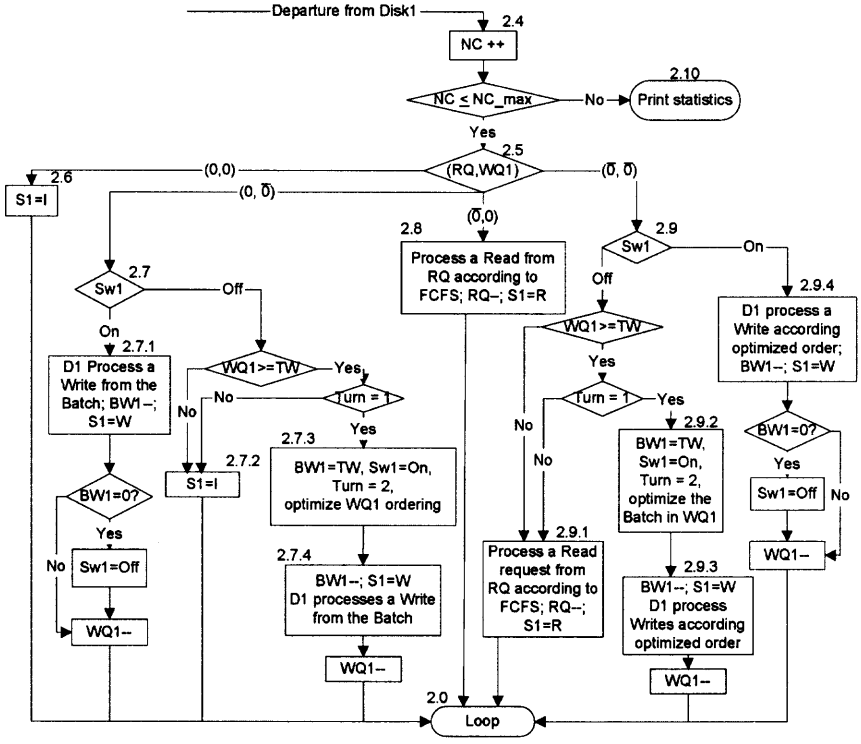
The flowcharts in Figure D.2 depict the logical steps to perform the Alternating Destages with Conditional Priorities (AD-CP) scheme in a mirrored disk system with an NVS cache (see Section 8.1.2). The specifications for the flowcharts are listed in Table D.2.

Table D.2 Specification for AD-CP Scheme Flow Charts.

Input:
Event$\in$ {Arrival, Departure form Disk1, Departure from Disk2} T$\in$ {Read, Write} – request type D$\in$ {D1, D2} – disk number
Scheduling Parameters:
TW - Write batch size TR - Threshold for processing Reads
System state and its components:
System state: $(S1, S2, RQ, WQ1, WQ2, BW1, BW2, Sw1, Sw2, Sr)$ $Si \in$ {R, W, I} R-disk being read, W-disk being written, I-idle disk (disk states) RQ – Number in Read queue WQi – Number in Write queue for Di BWi – Number of Writes left in the batch being processed by Di $Swi \in$ {On, Off}– Switches for Write batches: On-disk is processing a batch; Off-otherwise. (Swi is set by considering both the Write queue length and the status of the other Sw; it is checked right before processing Writes) $Sr \in$ {On, Off } – Switch for read batch: On-when $RQ \geq TR$; Off-otherwise (Sr is checked when a Read arrives)



(a)



(b)

Figure D.1 Flowcharts for ADU scheme (a) Arrivals (b) Departures ($\bar{0}$ denotes greater than 0).

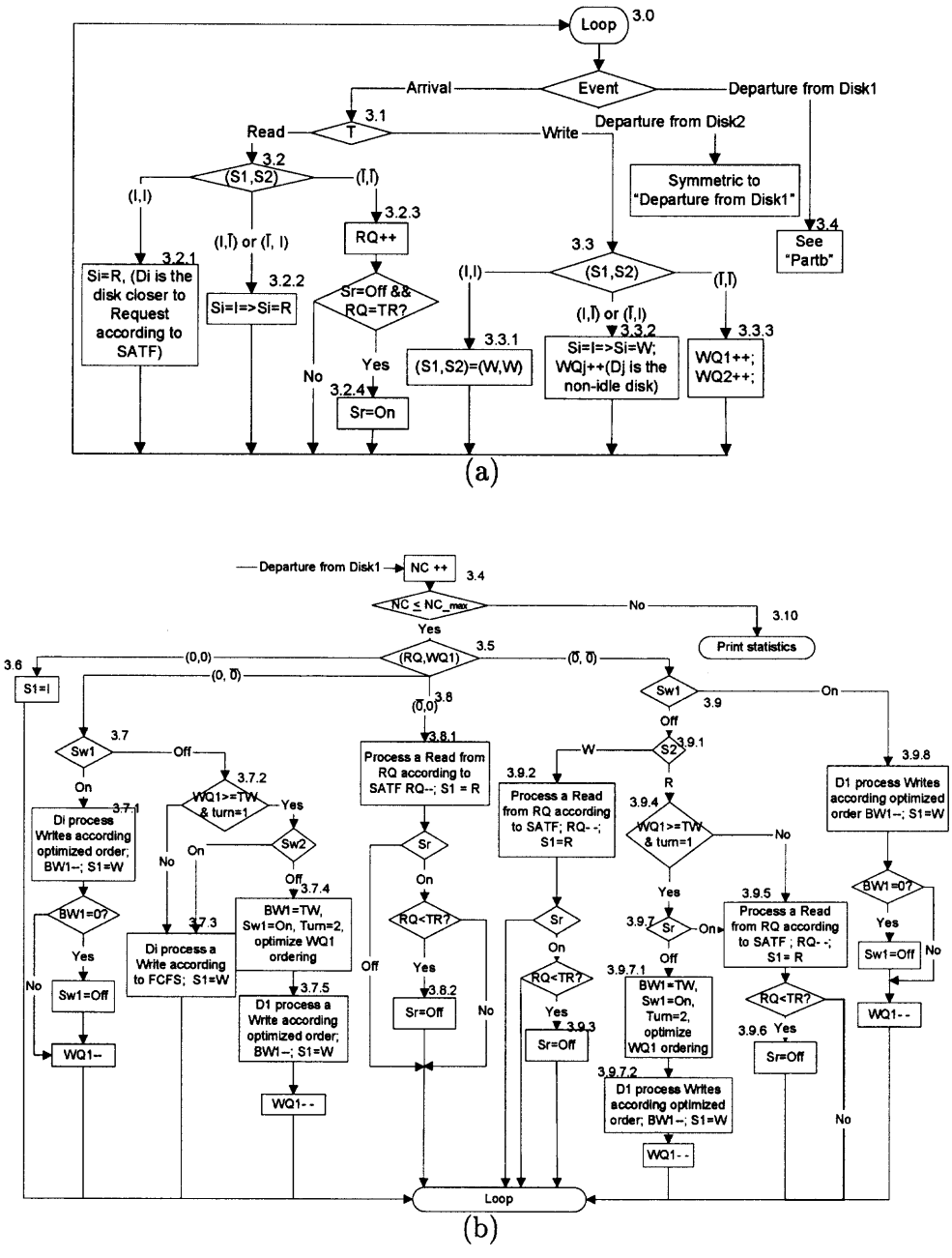


Figure D.2 Flowchart for AD-CP scheme (a)Arrivals (b)Departures ($\bar{0}$ denotes greater than 0).

REFERENCES

[1] K. K. Ramakrishnan, P. Biswas, and R. Karedla, "Analysis of file I/O traces in commercial computing environments," in *Proceedings of the 1992 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pp. 78-90, ACM Press, 1992.

[2] P. J. Denning, "Effects of scheduling in file memory operations," in *Proc. AFIPS Spring Joint Computer Conf. - SJCC*, vol. 30, pp. 9-21, 1967.

[3] M. Seltzer, P. Chen, and J. Ousterhout, "Disk scheduling revisited," in *Proceedings of the USENIX Winter 1990 Technical Conference*, (Berkeley, CA), pp. 313-324, USENIX Association, 1990.

[4] D. M. Jacobson and J. Wilkes, "Disk scheduling algorithms based on rotational position," Tech. Rep. HPL-CSP-91-7rev1, HP Laboratories, 1991.

[5] A. Riska and E. Riedel, "It's not fair - evaluating efficient disk scheduling.," in *MASCOTS*, pp. 288-295, Oct 2003.

[6] A. Riska, E. Riedel, and S. Iren, "Managing overload via adaptive scheduling," in *Proceedings of the 1st Workshop on Algorithms and Architecture for Self-Managing Systems*, pp. 23-24, June 2003.

[7] A. Thomasian and C. Liu, "Some new disk scheduling policies and their performance," in *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 266-267, ACM Press, 2002.

[8] A. Thomasian and C. Liu, "Disk scheduling policies with lookahead," *SIGMETRICS Performance Evaluation Rev.*, vol. 30, no. 2, pp. 31-40, 2002.

[9] A. Thomasian and C. Liu, "Performance evaluation of variations of the SATF scheduling policy," in *Proceedings of the 2004 International Symposium on Performance Evaluation of Computer and Telecommunication System*, 2004.

[10] D. A. Patterson, G. A. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, June 1-3, 1988* (H. Boral and P.-Å. Larson, eds.), pp. 109-116, ACM Press, 1988.

[11] P. M. Chen and D. A. Patterson, "Maximizing performance in a striped disk array," in *Proc. 17th Annual Int'l Symp. on Computer Architecture, ACM SIGARCH Computer Architecture News*, pp. 322-331, 1990.

[12] C. Han, A. Thomasian, and C. Liu, "Affinity based routing in zoned mirrored disks." Submitted to *The Computer Journal*, August 2004.

[13] A. Thomasian, J. Spirollari, C. Liu, C. Han, and G. Fu, "Mirrored disk scheduling," in *Proceedings of the international symposium on Performance Evaluation of Computer and Telecommunication Systems*, 2003.

[14] C. A. Polyzois, A. Bhide, and D. Dias, "Disk mirroring with alternating deferred updates... (prize paper)," in *Very large data bases, VLDB '93: proceedings of the 19th International Conference on Very Large Data Bases, August 24-27, 1993, Dublin, Ireland* (R. Agrawal, S. Baker, and D. Bell, eds.), (Los Altos, CA 94022, USA), pp. 604-617, Morgan Kaufmann Publishers, 1993.

[15] A. Thomasian and C. Liu, "Mirrored disk scheduling methods with a non-nolatile shared cache," technical memo, Integrated Systems Laboratory, Department of Computer Science, NJIT, 2003.

[16] C. Han, *Studies of disk arrays tolerating two disk failures and a proposal for a heterogeneous disk array.* PhD thesis, NJIT, Newark, NJ, May 2004.

[17] C. Ruemmler and J. Wilkes, "An introduction to disk drive modeling," *IEEE Computer*, vol. 27, no. 3, pp. 17-28, 1994.

[18] B. L. Worthington, *Aggressive centralized and distributed scheduling of disk requests.* PhD thesis, The University of Michigan, Michigan, June 1995.

[19] R. Geist and S. Daniel, "A continuum of disk scheduling algorithms," *ACM Transactions on Computer Systems (TOCS)*, vol. 5, no. 1, pp. 77-92, 1987.

[20] R. Geist and R. B. Ross, "Disk scheduling revisited: Can $o(n^2)$ algorithms compete," in *Proceedings of the 35th Annual ACM Sounthest Conference*, (Murfreesboro, TN), pp. 51 - 56, April 1997.

[21] B. L. Worthington, G. R. Ganger, and Y. N. Patt, "Scheduling algorithms for modern disk drives," in *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pp. 241-252, 1994.

[22] S. W. Ng, "Handling seek time variabilities in shortest access time first disk scheduling," Tech. Rep. RJ-10206, IBM Research, Almaden, CA, 2001.

[23] E. G. Coffman and M. Hofri, "Queueing models of secondary storage devices.," *Queueing Systems*, vol. 1, no. 2, pp. 129-168, 1986.

[24] M. Hofri, "Disk scheduling: FCFS vs. SSTF revisited," *Communications of the ACM*, vol. 23, no. 11, pp. 645-653, 1980.

[25] T. J. Teorey and T. B. Pinkerton, "A comparative analysis of disk scheduling policies.," *Commun. ACM*, vol. 15, no. 3, pp. 177-184, 1972.

[26] D. Bitton and J. Gray, "Disk shadowing," in *Fourteenth International Conference on Very Large Data Bases, August 29 - September 1, 1988, Los Angeles, California, USA, Proceedings* (F. Bancilhon and D. J. DeWitt, eds.), pp. 331-338, Morgan Kaufmann, 1988.

[27] R. P. King, "Disk arm movement in anticipation of future requests," *ACM Transactions on Computer Systems (TOCS)*, vol. 8, August 1990.

[28] R. Geist, R. G. Reynolds, and D. Suggs, "Minimizing mean seek distance in mirrored disk systems by cylinder remapping," *Performance Evaluation*, vol. 20, no. 1-3, pp. 97-114, 1994.

[29] S. W. Ng, "Improving disk performance via latency reduction," *IEEE Transaction on Computers*, vol. 40, no. 1, pp. 22-30, 1991.

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

[31] J. A. Solworth and C. U. Orji, "Distorted mirrors," in *Proceedings of the first international conference on Parallel and distributed information systems*, pp. 10-17, IEEE Computer Society Press, 1991.

[32] D. F. Towsley, S. Chen, and S.-P. Yu, "Performance analysis of a fault tolerant mirrored disk system," in *Performance '90, Proceedings of the 14th IFIP WG 7.3 International Symposium on Computer Performance Modelling, Measurement and Evaluation, Edinburgh, Scotland, 12-14 September 1990* (P. J. B. King, I. Mitrani, and R. Pooley, eds.), pp. 239-253, North-Holland, 1990.

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

[34] A. Thomasian and J. Menon, "RAID5 performance with distributed sparing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 6, pp. 640-657, 1997.

[35] "Storage performance council." `http://www.storageperformance.org`, (Retrieved on August 2003).

[36] A. Thomasian and C. Liu, "Empirical performance analysis of the SATF policy," in *Proceedings of the 2004 International Symposium on Performance Evaluation of Computer and Telecommunication System*, 2004.

[37] A. Thomasian and C. Liu, "Empirical performance evaluation of the SPTF disk scheduling policy." submitted to *The Computer Journal*, August 2004.

[38] L. Kleinrock, *Queueing Systems Volume 1: Theory.* Wiley Interscience, 1975.

[39] S. S. Lavenberg, *Computer Performance Modeling Handbook.* Academic Press, Inc., 1983.

[40] A. Thomasian, C. Han, G. Fu, and C. Liu, "A performance evaluation tool for RAID disk arrays," in *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems*, pp. 8-17, September 2004.

[41] P. Zabback, J. Menon, and J. Riegel, "The RAID configuration tool," in *Proceedings of the Third International Conference on High-Performance Computing (HiPC '96)*, p. 55, IEEE Computer Society, 1996.

[42] *DBC/1012 Database Computer System Manual.* Release 2, Teradata Corporation, Nov 1985.

[43] G. P. Copeland and T. Keller, "A comparison of high-availability media recovery techniques," in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 - June 2, 1989* (J. Clifford, B. G. Lindsay, and D. Maier, eds.), pp. 98-109, ACM Press, 1989.

[44] H.-I. Hsiao and D. J. DeWitt, "Chained Declustering: A new availability strategy for multiprocessor database machines," in *Proceedings of 6th International Data Engineering Conference*, pp. 456-465, 1990.

[45] S. Chen and D. Towsley, "A performance evaluation of RAID architectures," *IEEE Transactions on Computers*, vol. 45, no. 10, pp. 1116-1130, 1996.

[46] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes.* Cambridge, MA, USA: The M.I.T. Press, 2nd ed., 1972.

[47] E. K. Lee and R. H. Katz, "Performance consequences of parity placement in disk arrays," in *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 190-199, 1991.

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

[49] G. Gallo, F. Malucelli, and M. Marre, "Hamiltonian paths algorithms for disk scheduling," Tech. Rep. HPL-95-71, Hewlett Packard Laboratories, July 19 1995.

[50] M. Andrews, M. A. Bender, and L. Zhang, "New algorithms for the disk scheduling problem," in *37th Annual Symposium on Foundations of Computer Science - FOCS : October 14-16, 1996, Burlington, Vermont* (IEEE, ed.), (1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA), pp. 550-559, IEEE Computer Society Press, 1996.