DISK MANAGEMENT FOR A HARD REAL-TIME FILE SYSTEM

by

Raymond Man Kit Cheng

B.A.Sc., Electrical Engineering, University of British Columbia, Canada 1993.

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENT FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE**

in

**THE FACULTY OF GRADUATE STUDIES
ELECTRICAL ENGINEERING**

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September 28, 1995

© Raymond Man Kit Cheng, 1995

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of ELECTRICAL ENGINEERING

The University of British Columbia
Vancouver, Canada

Date Oct 12, 1995

Abstract

The problem of scheduling disk requests in a personal hard real-time read/write file system is examined. It is shown that any optimal algorithm for a simplified disk scheduling can be forced to thrash very badly. To avoid thrashing, we propose a *fixed-period scan* (FSCAN) approach for disk scheduling in our file system. The idea is to use the CSCAN policy to pick up the data blocks requested by a periodic preemptive schedule. The approach trades disk block size and memory buffer size for higher performance. We derive the worst-case seek and rotational overhead for the FSCAN algorithm, and we show that the worst-case seek overhead can be measured empirically for a large class of seek functions. Using this approach and utilizing measured seek functions from real disk drives, we show that these policies can transfer data at 40-70% of the maximum transfer rate of modern disk drives, depending on the file system parameters. A configuration program is developed to automatically test and configure the FSCAN algorithm for modern hard disks. The design, implementation and testing of this program are described.

Table of Contents

Abstract	ii
List of Figures	v
List of Tables	vii
Acknowledgement	viii
1. Introduction	1
1.1 Motivation	1
1.2 Objective	2
1.3 Outline	4
2. Background and Related Work	6
2.1 Traditional disk scheduling policies	6
2.2 Variants of SCAN and CSCAN	7
2.3 Deterministic admission control	9
2.4 Storage management of video files	9
2.5 Greedy strategy	10
2.6 Optimal dynamic-programming algorithm	11
3. Disk Model	12
3.1 Modern disk model	12
3.2 Seek time model	15
4. Worst-case Analysis of CSCAN Disk Algorithm	18
4.1 A generalized seek time model	18
4.2 Worst-case CSCAN seek analysis	19
4.3 Verification with an accurate disk simulator	25
4.3.1 Disk simulator	25

4.3.2 Worst-case CSCAN seek	27
5. A FSCAN Heuristic for Periodic Requests	30
5.1 Worst-case analysis of the FSCAN algorithm	30
5.2 Buffering requirement	36
5.3 Schedulability test	36
6. Performance Analysis and Evaluation	41
7. Software Development	50
7.1 Disk scan	50
7.2 Worst CSCAN seek test	56
7.3 FSCAN configuration	62
8. Conclusions and Future Work	68
8.1 Conclusions	68
8.2 Future work	69
Bibliography	71
Appendix: Sample Runs of the Configuration Software	75
A.1 The DISKSCAN program	75
A.1.1 Data file for the Micropolis 4110 drive	77
A.1.2 Data file for the Quantum LPS540S drive	78
A.2 The FSCAN program	80
A.2.1 Sample run of the FSCAN program	81
A.2.2 MATLAB [®] M-file output	82

List of Figures

Figure 3.1: Hard disk mechanical components.	13
Figure 3.2: Seek time function of a typical hard disk.	17
Figure 4.1: A non-decreasing concave seek function model.	19
Figure 4.2: A section of $f(x)$ in the domain $[b_{k-1}, b_k]$.	21
Figure 4.3: Disk access time distribution for HPC2200A and HP97560 disks.	27
Figure 4.4: Disk requests response time excluding rotational latency.	29
Figure 5.1: An illustration of $FSCAN(P,B)$ algorithm.	33
Figure 5.2: Expected waiting time to serve aperiodic requests with different scheduling scheme.	34
Figure 6.1: Effective schedulability factor $\beta(P,B)$ of $FSCAN(P,B)$ with HP97560.	43
Figure 6.2: Maximum buffer requirement in $FSCAN(P,B)$ with HP97560.	43
Figure 6.3: Maximum buffer required for $\beta(P,B)$ with HP97560.	44
Figure 6.4: Overhead components when block size is one half of a track.	45
Figure 6.5: Overhead components when block size is a whole track.	45
Figure 6.6: $\beta(P,B)$ with differing numbers of data streams when block size equals to one track.	46
Figure 6.7: Maximum buffer requirement in $FSCAN(P,B)$ with differing number of data streams when block size equals to one track.	47
Figure 6.8: Effective schedulability for 1-track transfer on different disks.	48
Figure 6.9: Maximum buffer required for 1-track for different disks.	48
Figure 7.1: Elapsed time measurements of 2000 sector jumps for the Micropolis 4110 disk drive.	52

Figure 7.2: Statistics of 2000 sector jump samples for the Micropolis 4110 disk drive.	54
Figure 7.3: Worst-case CSCAN seek curves and their 95% confidence interval upper bounds for the Micropolis 4110 and Quantum LPS540S drives.	58
Figure 7.4: Mean Seek time curve for 50 measurements of the Quantum LPS540S drive in different zones.	59
Figure 7.5: An example showing the rounding-up of a curve.	61
Figure 7.6: The rounded upper bound of worst-case CSCAN seek curves for the Micropolis 4110 and Quantum LPS540S drives.	62
Figure 7.7: Effective schedulability factor with the Micropolis 4110 disk drive.	65
Figure 7.8: Maximum buffer requirement with the Micropolis 4110 disk drive.	66
Figure 7.9: $\beta(P, 1 \text{ track})$ for the Micropolis 4110 drive and the corresponding buffer requirement.	67
Figure B.1: Graphs generated by MATLAB [®] showing the performance of the FSCAN algorithm with chosen scan period and block size.	84
Figure B.2: A plotting generated by MATLAB [®] showing the round-up worst CSCAN seek time function.	85

List of Tables

Table 1.1: The bandwidth requirements of digital media streams	2
Table 6.1: Parameters of some disk drives.	41

Acknowledgement

I would like to express my sincere thanks to my thesis supervisor, Dr. Donald Gillies, for introducing me to this thesis topic and for his continuous guidance in the past year. His remarkable knowledge in real-time system and his insight on this topic have been a great help to me. His patience, perceptiveness and encouragement are deeply appreciated. I would also like thanks my program supervisor, Dr. Mabo Ito, who grants me generous help and freedom in my research. Special thanks to Dr. Mark Greenstreet for his constructive criticisms and precious opinions toward my work. Acknowledgements to Kendra Cooper for her valuable comments on an earlier draft of this work. Many thanks also to Jeffrey Chow, John Jay Tanlimco, Darren Tsang, Steve So, Gary Yam and other colleagues who make my experiences as a graduate student filled with joyful memories. Warmest thanks to my respectable father, my prudent mother, my keen brother Terry, and my lovely sisters Selina and Pinky. Their immeasurable love and support will never be forgotten. My special gratitude go to Winnie Ho, who gives me continuous moral support and immense care throughout my work. Warm thanks to everyone in EP-Cell, who encourage and support me in prayers. I thank God for giving me such a good family and wonderful friends. Thank Him for granting me the opportunity to study and the needed wisdom to finish this work. It is only through Him that all thing are made possible.

1. Introduction

1.1 Motivation

Since the days of the Compatible Time-Sharing System at MIT, the primary service of a computer operating system has always been the electronic file system. Recently, the rise in real-time applications has suggested a need for real-time filing services. Research in this area has focused mainly on large centralized multimedia servers. It seems to us that the trends towards large multimedia servers dedicated to interpreting video streams of one type or another are paralleling the trend in IBM mainframe operating systems in the 1960's where there was one file type for each application. This trend led to a complexity bomb in the operating systems of that period. In the last 15 years industry has moved towards personal computing with loose coupling, not towards centralized systems. Modern personal computers are doubling in speed every 18 months, and disk drive density is advancing at a similarly rapid rate. We believe the trends in personal computers and in disk drives are more compelling than trends in large centralized servers.

1.2 Objective

In this thesis we study the design of a stand-alone *personal hard real-time file system*. An example of the storage and bandwidth requirements of a personal real-time file system is shown in Table 1.1. Consider a television reporter in a digital production studio. This person might want to merge an NTSC quality MPEG-2 video stream and a stereo audio stream into one data stream and store it, while watching another MPEG-2 video with stereo sound. This file system read/write workload contains 6 real-time data streams (3 video streams and 3 audio streams) with a total requested throughput rate of approximately 8.6 Mbps. The challenge for the file system is to insure that all real-time data streams are transferred continuously at their required throughput rates.

Media	Source	Size	Bandwidth	Storage
1000 pages of text		2 KB/page	—	2 MB
100 fax images		64 KB/image	—	6.4 MB
200 JPEG images		100 KB/image	—	20 MB
30 min of compressed voice 8KHz/8-bits, 4:1 compression	[Daig94]	—	16 Kbps	3.6 MB
1 hour of compressed CD music 44.1KHz/16-bits/2-channels, 4:1 compression	[Daig94]	—	353 Kbps	159 MB
30 min of compressed animation 320×240×16Hz/16-bits, 20:1 compression	[Furh94]	—	1 Mbps	225 MB
1 hour NTSC quality MPEG-2 video 720×480×30Hz/24-bits, 100:1 compression (with MPEG recording/playback card)	[Furh94] [Nasi95]	—	2.5 Mbps	1125 MB

Table 1.1: The bandwidth requirements of digital media streams

The design goal of our real-time file system is to handle heterogeneous uninterpreted data at arbitrary throughput rates. The total throughput goal is at

least 10 Mbps, motivated by the example above. The file system must guarantee real-time data delivery to memory and treat all streams equally, independent of bandwidth needs and read or write needs (subject to write verification). Each hard real-time data stream should be characterized by its maximum transfer rate, size and start-up latency. For non-real-time or soft real-time data streams, the file system should minimize their service response time on average. In addition, the file system should be able to store data non-contiguously on the disk drive.

In this thesis, the most important goal is to provide a deterministic timing guarantee for hard real-time data streams, which are assumed to be periodic tasks in our file system. On the other hand, non-real-time or soft real-time data streams are treated as aperiodic requests. Suggestions for handling the aperiodic data streams are briefly discussed in this thesis. We focus on the management of hard real-time periodic requests in this study.

Two approaches to disk scheduling are investigated: optimal scheduling and heuristic scheduling with substantial memory buffering. An optimal scheduling approach is studied in [Cheng95]. The study shows that there are workloads that would cause an optimal policy to intrinsically thrash. This result motivates a heuristic approach to the problem that we call fixed-period SCAN (FSCAN) algorithm for scheduling hard real-time data streams. The key idea is

to use the CSCAN policy to non-preemptively access the data blocks requested by a periodic preemptive schedule. The schedule can be generated using static or dynamic priorities. We derive the worst-case seek and rotational overheads for the FSCAN algorithm, and we show that the seek overhead can be measured empirically for a large class of seek functions. Results show that this policy can transfer data at 40-70% of the maximum disk transfer rate for modern disk drives, depending on the file system parameters and periodic scheduling policy.

A configuration program is developed to test a hard disk and to automatically configure the FSCAN algorithm for modern disk drives. The software runs under DOS and is written in the C++ language. The program performs a series of seek tests to extract the detailed drive information drive such as the zone-bit recording layout of the disk. With this information, the software is able to configure the file system for access by the FSCAN algorithm. The design, implementation and the testing of this software are described in this thesis.

1.3 Outline

This thesis is organized as follows. Chapter 2 surveys and evaluates different disk scheduling policies and real-time file server admission control techniques. The disk model used in our real-time file system is defined in

Chapter 3. Chapter 4 presents a worst-case analysis of the CSCAN. The new FSCAN heuristic approach to the real-time disk scheduling is described in Chapter 5. An evaluation of FSCAN is discussed in Chapter 6. Chapter 7 describes the development of a software package which automatically tests and configures the FSCAN algorithm in modern disks. Chapter 8 summarizes our work and suggests further research. In the appendix, sample runs of the FSCAN configuration software and the instructions for using it are presented. The data files generated by the software are also shown.

2. Background and Related Work

Many studies have been done with regard to disk scheduling policies and admission control techniques of multimedia file servers. In this chapter, we review and evaluate different techniques for disk scheduling and admission control in terms of their capability to handle hard real-time data.

2.1 Traditional disk scheduling policies

Disk arm scheduling algorithms must provide high throughput and deterministic timing control. Although good seek optimization can be achieved by traditional disk policies such as *Shortest Seek Time First* (SSTF), SCAN, and CSCAN, these policies are not appropriate in real-time applications since they do not consider the time constraints of disk requests. With SSTF, disk requests that are closest to the current disk head position will be served first. However, innermost and outermost tracks of the disk may receive poor service compared to the middle range tracks. Hence, starvation may occur and this is not acceptable in real-time applications.

The SCAN algorithm chooses the request to serve that results in the shortest distance in a preferred direction. The disk head moves and serves all requests in one direction until there are no further requests in that direction.

Then the head starts a new sweep in the opposite direction. A variant of SCAN is circular SCAN (CSCAN), which always serves requests in one direction only. The disk arm sweeps from the outermost track to the innermost track serving requests until the requests are exhausted in that direction. Then, the head moves back to the outermost request to start another inward sweep. The requests arriving in the current sweep are served in the next sweep. Both algorithms achieve good seek optimization and small variance in response time of requests. However, the SCAN and CSCAN algorithms have no notion of deadline for scheduling purposes.

Another scheme traditionally used in real-time scheduling is *Earliest Deadline First* (EDF), which is shown to be optimal if the periods and service times of requests are known in advance [Liu73]. However, applying a pure EDF scheme to disk scheduling is not appropriate because of the high costs of preemption and the non-preemptive nature of disk operations.

2.2 Variants of SCAN and CSCAN

Proposed by Reddy and Wyllie [Redd93], SCAN-EDF is a strategy for real-time disk scheduling where disk requests with the earliest deadline are served first. If some disk requests have the same deadline, they are served according to their track positions and the policy reduces to SCAN. Reddy and

Wyllie also consider an aperiodic server proposed by [Lin91] in which aperiodic requests are given higher priority over the periodic real-time requests. When deadlines of requests are deferred, results show that CSCAN has slightly better performance than SCAN-EDF for real-time traffic, and EDF is the worst. However, SCAN-EDF is the best scheme for aperiodic request performance. In fact, the efficiency of SCAN-EDF greatly depends on the fraction of disk requests that have the same deadline and are served with the seek optimized SCAN policy. There is no such restriction in our new scheduling policy.

Other variants of SCAN are *Group Sweeping Scheduling* (GSS) [Chen93] and the *Sorting-Set Algorithm* (SSA) [Gemm93]. Both schemes are functionally equivalent: a set of real-time data streams is divided into several groups and the groups are served in a round-robin fashion. Members within a group are served according to SCAN. If the size of a group is large, the response time for a particular request within the group may vary in the different cycles. Besides, the focus of both studies is on optimizing the disk arm scheduling, not the real-time data admission control. In other words, a deterministic timing guarantee of data delivery is not provided.

Many other hybrid policies based on SCAN or CSCAN exist such as *Feasible Deadline Scan* (FD-SCAN), *Earliest Deadline Scan* (D-SCAN) [Abbo90], *Priority Scan* (P-SCAN) [Care89], and V-SCAN (a variable mixture of SSTF and

SCAN) [Geis87]. All these strategies add the time notion to SCAN or CSCAN in order to increase the schedulability. However, these policies do not provide a hard real-time deterministic schedulability control.

2.3 Deterministic admission control

Some promising approaches to real-time disk scheduling are based on the work by Liu and Layland [Liu73]. Daigle and Strosnider provide a framework to design a multimedia server with *a priori* reasoning about the throughput and the schedulability of a system [Daig94]. They employ a necessary and sufficient schedulability test based on the work by Lehoczky *et al* [Leho87]. Tindell uses a similar approach [Tind93]. He applies the existing fixed priority pre-emptive scheduling theory to the disk scheduling problem, in which the worst-case behaviour of real-time data streams can be predicted. Both policies assume a linear seek function and contiguous file storage. We use a more accurate seek time function and non-contiguous file storage in analysis. We also develop a disk model that captures the details of different overhead components of a modern disk.

2.4 Storage management of video files

Another related work which focuses on the storage management of digital video files is proposed by Tobagi *et al* [Toba93]. Their video server manages an

array of disks and the video data streams are striped among the disks. In their model, only homogenous data with the same requested throughput rates are considered. Thus the main goal of their study is to maximize the number of streams that the server can support for a given memory size and start-up latency requirement. They determine this maximum number by finding a bound on the probability that any one stream fails to be served continuously. This maximum number is not a deterministic guarantee, since there is a non-zero probability of hard failure. In addition, the paper does not provide an explanation of their bounds calculations. In this thesis we consider heterogeneous streams with arbitrary data rates and derive a guaranteed deterministic real-time data delivery admission.

2.5 Greedy strategy

Another approach to disk scheduling is the greedy strategy [Abbo84, Vin94]. This scheme tries to minimize both the seek and rotational latency by finding an optimal sequence for retrieving data blocks on disk¹. This is done by constructing a fully connected directed graph in which edges are weighted according to seek and rotational latencies, and then a travelling salesperson

¹ Some other policies which take rotational latency into account have been proposed by [Jacob91], [Ng91] and [Selt90].

problem is solved with a near-optimal greedy algorithm [Vin94]. The greedy strategy is not a dynamic scheduling algorithm as the set of requests are required to be known *a priori*. For this reason, this scheme is not capable of handling non-predetermined requests, such as write requests.

2.6 Optimal dynamic-programming algorithm

An optimal dynamic scheduling approach to design our hard real-time file system is presented in [Cheng95]. For arbitrary aperiodic requests, the problem of moving the disk arm is modelled simplistically as a travelling salesperson problem on a one dimensional line, where travel time is proportional to the distance and the time spent at each city is zero. This paper proposes an optimal dynamic-programming algorithm to solve this problem. However, even in this simplified model, an optimal algorithm can be forced to thrash very badly if data fragmentation is not managed carefully. These observations motivate the heuristic approach to disk scheduling and block management presented in this thesis.

3. Disk Model

In this chapter, we define the disk model used by our real-time file system. We first discuss the details of a modern magnetic disk drive that we intend to model. Then, we analyze accurate seek time functions for modern disks.

3.1 Modern disk model

A magnetic disk consists a collection of double-sided magnetically coated *platters* which rotate on a common spindle, typically at 3600, 4000, 5400, or 7200 rpm. Each disk *surface* consists of concentric *tracks*, which in turn are divided into *sectors*. A sector is the smallest data storage unit, and typically holds 512 bytes of raw data plus header/trailer information such as error correction codes. A set of tracks at a common distance from the centre of the disk is called a *cylinder*. To access the data stored in a particular sector, the location in terms of cylinder, surface and sector have to be given to the disk mechanism. Figure 3.1 shows the mechanical components of a hard disk.

Figure 3.1: Hard disk mechanical components.

A set of moveable disk arms attached to the same rotational pivot can be positioned to a particular cylinder. This operation is called a *seek* and the time needed to finish a seek is called the *seek time*. The seek operation is typically broken into a high-speed acceleration phase and a *track-following* phase. During the track-following phase, the disk read/write head is activated to find and position the arm precisely on the target track. The time needed for this end-of-seek settling is called the *settling time*. When the correct track is found there is a delay before the desired sector rotates into position under the disk head. The delay due to this rotation is called the *rotational latency*.

There are other mechanical delays and overheads of disk operations. A *track switch* occurs when the disk arm moves from the current cylinder (track) to an adjacent one. A typical value for the track switch time is approximately the same as the settling time. Similarly, when the disk switches its data channel

from one disk surface to another in the same cylinder, a *head switch* occurs. Such a switch typically takes one third to one half of the settling time [Ruem94]. Another delay is the *read/write overhead*, which is incurred when the disk head reads/writes data from/to a disk sector.

Since the disk spins at a constant rotational rate, the horizontal velocity of the recording media at the edge of the platter is higher than in the centre. Disk manufacturers make use of this by *zoning* the disk into sets of concentric cylinders. There are more sectors per track in the outer zones than in the inner zones. Modern disks have 3–9 zones with a greater number of sectors per track in the outer zones.

Given that a delay is incurred during a track switch, the position of the sectors in each track is *skewed* by one or more positions relative to where they were on the previous track. Hence, a sequential read/write from one track to another will not incur a full rotational delay. The track skew factors are different from one zone to another.

Some flawed sectors or *bad sectors* may exist in the disk surfaces during manufacture. When this happens, the bad sector will be re-mapped to a *spare sector*, which is usually located at the end of certain tracks or cylinders. Again, each zone may have a different number of spare sectors.

A disk occasionally needs to recalibrate itself because of thermal expansion and bending of the disk arms. This process is called *thermal calibration* (TCAL). When this occurs, the disk is unavailable to process commands from the disk controller for 500 – 800 ms [Ruem94]. This long delay may cause serious problems for continuous media applications. Certain “AV” disk products specified for digital-video applications have a means to maintain a relatively consistent response time during TCAL. For instance, TCAL is done one head at a time in some AV disks [Holz93]. A simple algorithm to deal with TCAL is to force the disk drive to recalibrate itself periodically at a known time interval.

The problem of hard real-time file system design is greatly complicated by TCAL and zoning. In this thesis we do not address these problems directly. We recommend that someone wishing to use this blueprint purchase AV disk drives and make worst-case assumptions about the number of sectors per track. This is one way to dispense with the problem of zoning and TCAL in disk drives.

3.2 Seek time model

A seek time function $s(d)$ of a disk describes the time required to position the disk head over the desired cylinder where d is the number of cylinders to travel. Many studies regard seek latency as a linear function $s(d) = a_1 + a_2d$

where a_1 and a_2 are mechanical constants [Gemm93, Sale91]. Other studies use $s(d) = a_1 + a_2\sqrt{d}$ where a_1 is the mechanical settling time and a_2 is a constant depending on the acceleration of the disk arm. This function is accurate for seeks less than 1/3 of the total number of cylinders, and is widely used [Abbo90, Bitt89].

A more accurate seek time function incorporates both the linear and non-linear behaviour of the seek latency [Ruem94]. Let N be the total number of cylinders, and let D be the boundary at which both the non-linear and linear function applies. Let a_1 and a_3 be the mechanical settling time constants, and let a_2 and a_4 be constants depending on the acceleration of the disk arm and the top speed of the disk arm respectively. The seek function is defined as

$$(3.1) \quad s(d) = \begin{cases} a_1 + a_2\sqrt{d} & 0 < d < D \\ a_3 + a_4d & D \leq d \leq N. \end{cases}$$

Figure 3.2 shows the seek time versus distance curve for a typical modern hard disk. Notice that the curve is a non-decreasing concave function of d .

Figure 3.2: Seek time function of a typical hard disk.

In this thesis, we assume a non-decreasing concave seek time function $s(x)$ with $s'(x+\delta) \leq s'(x)$ and $s'(x) \geq 0$ for all x and $\delta \geq 0$. This seek-time model contains the three previous models as special cases. Based on this seek-time model, in the next chapter we show that the worst-case seek overhead can be measured empirically for a large class of seek functions.

4. Worst-case Analysis of CSCAN Disk Algorithm

In [Cheng95], it is shown that an optimal seek algorithm for an arbitrary series of requests will cause disk thrashing very badly. It can be shown that the disk transfer rate in the worst case is very low, as little as 1% of the maximum transfer rate if sequential requests are near opposing ends of the disk, and if the data transfers are small. This motivates a heuristic approach to the disk scheduling problem that employs the CSCAN algorithm at the core of our real-time file system. In this chapter, we perform a worst-case analysis of the seek overhead of CSCAN. We propose a non-decreasing concave seek time function model and show that for a large class of seek functions the worst-case CSCAN overhead can be measured empirically without the need to measure the seek function itself.

4.1 A generalized seek time model

We employ a non-decreasing continuous concave seek time function $s(x)$ with $s'(x+\delta) \leq s'(x)$ and $s'(x) \geq 0$ for all x and $\delta \geq 0$, as shown in Figure 4.1. This generalized seek-time model captures all the seek models discussed in the last chapter as special cases. A similar model has been employed in [Toba93], but

only an intuitive result is discussed in that study. A formal and detailed analysis based on this model is presented in the remainder of this chapter.

Figure 4.1: A non-decreasing concave seek function model.

4.2 Worst-case CSCAN seek analysis

We employ CSCAN as the core of our file system. With CSCAN, the disk steps across a range of cylinders and services requests in increasing order of cylinders. When the disk arm reaches the cylinder $N-1$ (or the innermost request), it returns to the cylinder 0 with a full-stroke seek and starts another sweep. Consider one sweep of CSCAN. It appears that the worst time to perform a series of n seeks across the disk is bounded by the time taken when all seek locations are equally spaced across the entire range of cylinders. Based on

the generalized non-decreasing continuous concave seek function proposed in the last section, we can prove this as the following theorem.

Theorem: For any function $F(\mathbf{x}) = \sum_{i=1}^n f(x_i)$ where $\mathbf{x} = (x_1, x_2, \dots, x_n)$, $\sum_{i=1}^n x_i = N$, and $f(x)$ is a non-decreasing continuous piecewise-differentiable concave function with $f'(x) \geq 0$ and $f'(x+\delta) \leq f'(x)$ for all x and $\delta \geq 0$, $F(\mathbf{x})$ is maximized when $x_1, x_2, \dots, x_n$ are evenly distributed along a number line from 0 to N (i.e. when $x_1 = x_2 = \dots = x_n = N/n$). Thus $\max(F(\mathbf{x})) = n \cdot f(N/n)$.

Proof: Let $f(x)$ be divided into L maximal differentiable regions with domains $[b_0, b_1], [b_1, b_2], \dots, [b_{L-1}, b_L]$. We divide $f(x)$ into $g_1(x), g_2(x), \dots, g_L(x)$ and extend each $g_k(x)$ linearly as follows.

$$g_k(x) = \begin{cases} f(b_{k-1}) + f'_+(b_{k-1}) \cdot (x - b_{k-1}) & x < b_{k-1} \\ f(x) & b_{k-1} \leq x \leq b_k \\ f(b_k) + f'_-(b_k) \cdot (x - b_k) & b_k < x \end{cases}$$

Each $g_k(x)$ is a non-decreasing differentiable concave function with $g'_k(x+\delta) \leq g'_k(x)$ and $g'_k(x) \geq 0$ for all x and $\delta \geq 0$. Figure 4.2 shows a section of $f(x)$ in the domain $[b_{k-1}, b_k]$. Note that $g_k(x)$ is bounded above by all other $g_j(x)$, $j \neq k$ in the domain $[b_{k-1}, b_k]$. Also observe that $f(x) = g_k(x)$ for $x \in [b_{k-1}, b_k]$ and $f(x) < g_k(x)$ for $x \notin [b_{k-1}, b_k]$.

Figure 4.2: A section of $f(x)$ in the domain $[b_{k-1}, b_k]$.

We first find the maximum value of the sum $G_k(\mathbf{x}) = \sum_{i=1}^n g_k(x_i)$ with the constraint $\sum_{i=1}^n x_i = N$. Let $x_n = N - \sum_{i=1}^{n-1} x_i$. We rewrite the sum as $G_k(\mathbf{x}) = \sum_{i=1}^{n-1} g_k(x_i) + g_k(x_n)$. Taking partial derivatives of $G_k(\mathbf{x})$ with respect to j where $1 \leq j \leq n-1$, we obtain

$$\frac{\partial G_k}{\partial x_j} = g'_k(N - \sum_{i=1}^{n-1} x_i) \cdot (-1) + g'_k(x_j).$$

Setting $\frac{\partial G_k}{\partial x_j} = 0$ and finding the critical point, we have

$$(4.1) \quad g'_k(x_j) = g'_k(N - \sum_{i=1}^{n-1} x_i).$$

To solve (4.1), we consider two types of functions $g'_k(x)$: (I) $g'_k(x+\delta) < g'_k(x)$ and (II) $g'_k(x+\delta) = g'_k(x)$.

CASE I: $g'_k(x+\delta) < g'_k(x)$.

In this case, $g'_k(x+\delta) < g'_k(x)$ for all $\delta > 0$ and $g'_k(x) \geq 0$, hence, $g'_k(x+\delta) = g'_k(x)$ only if $\delta = 0$. Thus, we get the critical point $x_j = x_n$ from (4.1). Since $g_k(x)$ is a concave function with $g''_k(x) \leq 0$ for all x , therefore the critical point $x_j = x_n$ represents a local maxima $G_k(\mathbf{x})$. Hence, $x_1 = x_2 = \dots = x_n = N/n$ gives a maximum value of $G_k(\mathbf{x})$.

CASE II: $g'_k(x+\delta) = g'_k(x)$

The function $g_k(x)$ is a non-decreasing linear function in this case. Hence, $\sum_{i=1}^n g_k(x_i)$ is the same for any vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$ obeying the constraint $\sum_{i=1}^n x_i = N$. Thus, there is no unique solution to (4.1). As a result, $x_i = N/n$ for all i is a non-unique solution to (4.1).

Since we have $f(x) \leq g_k(x)$ for all x and the constraint $\sum_{i=1}^n x_i = N$, we obtain

$$\begin{aligned} \sum_{i=1}^n f(x_i) &\leq \sum_{i=1}^n g_k(x_i) \\ \max \left(\sum_{i=1}^n f(x_i) \right) &\leq \max \left(\sum_{i=1}^n g_k(x_i) \right). \end{aligned}$$

Recall that $f(x) = g_k(x)$ for $x \in [b_{k-1}, b_k]$. Hence,

$$\max \left(\sum_{i=1}^n f(x_i) \right) = \max \left(\sum_{i=1}^n g_k(x_i) \right)$$

when $x_i \in [b_{k-1}, b_k]$ for all x_i , that is, when $x_1 = x_2 = \dots = x_n = N/n$. $\square$

Recall from the last chapter that an accurate and practical seek time function can be defined as (3.1)

$$s(d) = \begin{cases} a_1 + a_2 \sqrt{d} & 0 < d < D \\ a_3 + a_4 d & D \leq d \leq N. \end{cases}$$

Thus the time to perform a series of n seeks across the disk in one direction can be formulated as the problem:

$$(4.2) \quad \begin{array}{ll} \text{maximize} & S(\mathbf{d}) = \sum_{i=1}^n s(d_i) \\ \text{subject to} & \sum_{i=1}^n d_i \leq N \end{array}$$

where $\mathbf{d} = (d_1, d_2, \dots, d_n)$. Since $s(\mathbf{d})$ is a non-decreasing function, we can assume that in any worst-case solution $\sum_{i=1}^n d_i = N$. We observe that (4.2) is composed of a non-linear and a linear portion. For each CSCAN sweep, the disk arm can only have at most $\lfloor N/D \rfloor$ linear seeks (with seek distance $D \leq d \leq N$). So if $n > N/D$ we are in the non-linear range, otherwise we are in the linear range. Thus, we can deduce the following

Corollary 2. Using (3.1) as the seek time function model, the worst time to perform a series of n seeks across the disk in one direction (CSCAN) is given by

$$\begin{aligned}
 W_{seek}(n) &= \begin{cases} n(a_3 + a_4(N/n)) & n \leq N/D \\ n(a_1 + a_2\sqrt{N/n}) & n > N/D \end{cases} \\
 (4.3) \quad &= \begin{cases} a_3n + a_4N & n \leq N/D \\ a_1n + a_2\sqrt{nN} & n > N/D \end{cases} .
 \end{aligned}$$

In fact, Theorem 1 can be applied to any disk which has a non-decreasing continuous concave seek function. We have found in the literature only one disk that has a non-concave function [Henn90]. For this disk, the seek-time function can be closely approximated by an upwardly rounded concave curve. Thus, to find the worst-case seek overhead for a series of n disk requests using CSCAN, we can just do a full CSCAN with n evenly spaced seeks across the entire range of cylinders. Hence, we can measure the worst-case seek overhead of a disk that conforms to our assumptions even if the seek time function of that disk is not known. This can save us from the tedious measurement of the disk constants a_1 , a_2 , a_3 and a_4 . Hence, the significance of Theorem 1 is that it allow us to build a hard real-time file system without having to obtain proprietary and sensitive information from the disk manufacturer.

4.3 Verification with an accurate disk simulator

In the previous section we showed that the worst-case time to perform a series of n seeks across the disk in CSCAN is bounded by the total time taken when all seek locations are equally spaced across the entire range of cylinders, for non-decreasing concave seek function. In this section, we validate Theorem 1 with an accurate disk simulator. We first describe the disk simulator model and its validation, then we show the simulation results illustrated the significance of Theorem 1.

4.3.1 Disk simulator

A validated disk simulator was developed based on the descriptions of modern disks presented in [Ruem94] and [Wort94]. Our disk simulator is event-driven and written in ANSI C. It is configured to model the HPC2200A and HP97560 disk drives. The important characteristics of these two disks, such as seek latency, rotational head positions, head/track switches, and spare regions are accurately modelled. However, the disk caching feature of the HP97560 drive is not modelled, and the HPC2200A drive has no disk cache.

To test the disk simulator, we use a randomly-driven disk request generator to simulate the disk access patterns studied in [Ruem93]. Among the three computer systems they have studied, one system uses HPC2200A disks, and one use the HP97560 disks. Ruemmler and Wilkes use some of the

representative traces to validate their disk simulator described in [Ruem94]. Based on the trace patterns described, we assume 57% of the total disk requests are writes. The size of the request data blocks range from 0.1 to 8 Kbytes. The traces show that the percentage of sequential disk requests is only 9% for the HPC2200A drive and 38% for the HP97560 drive respectively. In our case, we assume 8.33% of the total disk requests are sequential for the HPC2200A drive and 33.3% for the HP97560 drive. We also assume that the request queue is always full. With this simple model, one million disk requests are generated for each disk simulation and the disk response times are recorded. Figure 4.3 shows the disk response time distribution for the HPC2200A and HP97560 disk drives. The mean response time is 24.80 ms for the HPC2200A disk and 17.34 ms for the HP97560 disk (compared to 25.49 ms and 17.51 ms obtained from trace-driven simulations used in [Ruem94]). These disk response time distributions show the validation of our disk simulator.

Figure 4.3: Disk access time distribution for HPC2200A and HP97560 disks.

4.3.2 Worst-case CSCAN seek

We use our validated disk simulator to illustrate the significance of Theorem 1. Given a non-decreasing concave seek function, this theorem guarantees that the worst time for a series of n seeks following CSCAN scheme is bounded by the total seek time in which all the seek distances are equally spaced across the entire range of cylinders. The algorithm to verify this theorem is

simple. For a given n , the test driver makes disk requests according to CSCAN scheme with all different possible combinations of seek distances. For instance, let N denotes the number of cylinders and $(d_1, d_2, \dots, d_n)$ represents the seek distances of n requests. If $n = 3$, then different sets of disk requests such as $(1,1,1)$, $(1,1,2)$, $\dots$, $(1,1,N-3)$, $(1,2,1)$, $\dots$, $(1,2,N-4)$ will be made. The number of different seek patterns is $\binom{N}{n}$, which is a very large number even with a small value of n . Hence, for our purposes, we verify up to 10 requests. In addition, rounding is applied in our algorithm when n is larger than 3. The best and the worst response time and the corresponding seek distances combination are recorded. All disk requests are reads and the size of each requested block is one sector. Rotational latencies are excluded in the tests. Two disk drives are used in the simulations and the resulting response time curves are shown in Figure 4.4. Note that the worst-case seek time curves for both drives are non-decreasing concave functions. Results show that the worst response time occurs when seek distances are equally spaced across N , as stated by Theorem 1. The HP97560 drive has a better worst-case response time because the drive has a relatively fast seek ability (refer to Table 6.1 for the seek time function of these two disks). The best response time occurs when all seek distances are equal to 1 (i.e. track

switches). The HPC2200A drive has a better response time in this case as it takes shorter time to read a sector² and it has a shorter read overhead³.

Figure 4.4: Disk requests response time excluding rotational latency.

² The sector size of HPC2200A is 256 bytes and HP97560 is 512 bytes.

³ Read overhead for HPC2200A is 1.1 ms and HP97560 is 2.2 ms.

5. A FSCAN Heuristic for Periodic Requests

In this chapter, we propose a heuristic approach to scheduling real-time disk requests. We investigate the idea of using main memory to store a series of sectors that have already been read or that are to be written. This allows us to employ the CSCAN algorithm in our real-time file system. We calculate the worst-case timing behaviour of real-time disk operations. We then define a fixed-period scan (FSCAN) algorithm, and compare the FSCAN algorithm to an ideal algorithm which never moves the disk head or experiences rotational latency.

5.1 Worst-case analysis of the FSCAN algorithm

The file system services a stream system of n continuous data stream requests $S = \{S_1, S_2, \dots, S_n\}$. Each data stream S_i is an independent unit of data transfer and has a requested maximum throughput rate of R_i . We ideally want to schedule these data streams using some preemptive scheduling algorithm. The major difficulties in applying an ideal preemptive scheduling algorithm to disk scheduling are (1) disk operations are not preemptable, and (2) the costs of context switches and other overheads such as seek and rotational latency are high. Since pre-emptive scheduling is not possible, we map a preemptive

schedule such as EDF onto a non-preemptive *fixed-period scan* (FSCAN) schedule. So, by analyzing the non-ideal elements of disk operation and finding the worst possible overheads in FSCAN, we may derive an admission test for the stream requests.

Let B denote the logical *block size*, the smallest unit of disk requests in bytes. The block size is typically several physical sectors or a whole track. The time needed to transfer a block of data to or from the disk is defined as $C=B/DTR$, where DTR is the maximum *data transfer rate* of the disk. The quantity C is an analogue to the *computation time* of a task in periodic scheduling theory. In our file system, each data stream S_i is broken into transfers that are multiples of the computation time C . The *request period* of S_i is denoted by T_i , and the *deadline* of S_i is equal to its period. Let R_i be the requested throughput rate of stream S_i , the request period of S_i may be defined as $T_i = B/R_i$.

The file system operates in cycles. We define P as the *scan period* or the time interval for one operating cycle. In each cycle, the file system scheduler determines m_i , the number of data blocks to be read or written in stream i . This quantity is bounded by

$$(5.1) \quad m_i \leq \left\lceil \frac{P}{T_i} \right\rceil = \left\lceil \frac{PR_i}{B} \right\rceil.$$

The scheduler then issues a set of disk requests, which are scheduled according to the CSCAN algorithm. Since CSCAN is applied with periodicity P , we call this method the *fixed-period SCAN* or $FSCAN(P,B)$, where P and B represent the scan period and the block size respectively. In cases when there are aperiodic requests (non-real-time data streams), we may employ an aperiodic server to handle them [Leho87, Spru89]. We focus on the periodic requests in this thesis. The scan period can also be interrupted as the start-up latency of a data stream. It represents the maximum time elapsed for a new data stream to be consumed by the user (or stored to the disk) since the request is initiated.

Figure 5.1(a) depicts an ideal EDF periodic preemptive schedule of a set of three data streams S_1 , S_2 and S_3 in time-line form. The cost of preemption is assumed to be zero. The deadlines of these data streams are equal to their request periods, which are denoted by T_1 , T_2 and T_3 respectively. The length of the computation time of all three streams are the same, i.e. $C_1 = C_2 = C_3$. The *priority* of each the stream is assigned based on the length of its periods: the shorter the period, the higher its priority. Hence S_1 is the highest priority in this example. Also note that the system parameter scan period is arbitrarily assigned in the example.

The FSCAN policy retrieves the blocks in Figure 5.1(a) by following the schedule in Figure 5.1(b). For each stream the number of data blocks to be

served in one period can be determined by (5.1). For example, stream S_1 transfers $m_1 = \lceil P/T_1 \rceil = 5$ blocks in one scan period. The data blocks of the same data stream need not be contiguously stored on the disk, such as C_1 and C_3 in the example. In Figure 5.1(b), t_h and t_{return} denote the disk overheads and the time needed for one full return seek respectively. These are the physical overheads in the $FSCAN(P, B)$ algorithm.

Figure 5.1: An illustration of $FSCAN(P, B)$ algorithm.

Another type of overhead arises when we make the discrete disk system present a continuous access model. This includes overhead due to clock resolution and round-up from (5.1). In some (worst-case) scan periods, the scheduler will read/write one extra data block from each of the streams, such as

S_2 and S_3 in the example. This will be followed in the next period by transferring one less data block from each of the streams.

We employ CSCAN in our FSCAN(P, B) algorithm rather than SCAN because CSCAN performs better for aperiodic requests. Figure 5.2 illustrates this fact. At a particular moment, the disk head is located at cylinder x and is sweeping in the $x+1$ direction, when an aperiodic request targeted at cylinder $x-1$ arrives. As we can see from the Figure 5.2a and 5.2b, if the CSCAN return time is small, the expected waiting time for an aperiodic request for SCAN is almost twice the waiting time for CSCAN. By using CSCAN, any aperiodic server will be able to make better a worst-case latency guarantee. Later, we will see that the full-return seek costs very little in overall throughput. We believe this trade-off is appropriate for most file systems.

Figure 5.2: Expected waiting time to serve aperiodic requests with different scheduling scheme.

To ensure the schedulability of our file system, we must find the worst-case overhead in FSCAN. The two major physical overheads are seek time and rotational latency. Let $m = \sum_{i=1}^n m_i$ be the total number of disk requests issued in

one scan period, the worst case seek time $W_{seek}(m)$ for CSCAN is given by (4.3). The worst case rotational latency is denoted by t_{rot} . Other physical overheads are read/write overhead t_{rw} , time for track switch⁴ t_{ts} , and the time needed for one full return seek t_{return} . The overhead due to the round-up computation of m_i in (5.1) is equal to nC or $n(B/DTR)$. Thus, the overhead function of FSCAN for a given data stream set S is

$$(5.2) \quad H(P,B) = W_{seek}(m) + m \cdot (t_{rot} + t_{rw} + t_{ts}) + t_{return} + n \cdot \frac{B}{DTR}.$$

As we can see from (5.2), the total overhead greatly depends upon m . Also note that if the SCAN algorithm were used rather than CSCAN, the t_{return} term would disappear from the equation. Let $R = \sum_{i=1}^n R_i$ denote the total data throughput rate, the worst-case value of m can be determined from (5.1):

$$(5.3) \quad \begin{aligned} m &= \sum_{i=1}^n m_i = \sum_{i=1}^n \left\lceil \frac{P \cdot R_i}{B} \right\rceil \\ &\leq \sum_{i=1}^n \left(\frac{P \cdot R_i}{B} \right) + n \\ &= \frac{P \cdot R}{B} + n. \end{aligned}$$

⁴ In the worst case track switches occur rather than head switches because a head switch typically takes only 1/3 to 1/2 the time needed for a track switch.

Hence, we find the upper bound of the total number of disk requests in one scan period is $m = (P \cdot R)/B + n$.

5.2 Buffering requirement

We use a double buffering scheme for our file system. The first memory buffer holds data from the previous scan period and it is read-only. The second memory buffer holds data obtained in the current scan period and it is write-only. At the end of each scan period, the roles of the buffers are reversed. Following this scheme, each data stream will require $\lceil 2P/T_i \rceil \cdot DTR$ memory buffer. Hence, the total memory buffering requirement of $FSCAN(P, B)$ is given by:

$$\begin{aligned}
 BUF(P, B) &= \sum_{i=1}^n C \cdot \left\lceil \frac{2P}{T_i} \right\rceil \cdot DTR = C \cdot DTR \cdot \sum_{i=1}^n \left\lceil \frac{2PR_i}{B} \right\rceil \\
 (5.4) \quad &\leq \left(\frac{B}{DTR} \right) \cdot DTR \cdot \left(\frac{2PR}{B} + n \right) = 2PR + nB.
 \end{aligned}$$

5.3 Schedulability test

Using the analysis above, we can perform a simple and efficient schedulability test for a given set of data streams. Since $H(P, B)$ in (5.2) represents the worst-case overhead of $FSCAN$ in one scan period, the percentage of a period that can be used for physical data transfer is then defined by

$(P - H(P, B))/P$. Hence, if $R = \sum_{i=1}^n R_i$ denotes the sum of all individual throughput rates R_i of every stream, a workload is schedulable if

$$(5.5) \quad R \leq \frac{P - H(P, B)}{P} \cdot DTR.$$

We want to find the upper bound on R , that is, the maximum throughput rate that the file system can support. To do this, the worst case value of the overhead function $H(P, B)$ must be found. In the worst case, the maximum number of disk requests is given by (5.3). Since $m > N/D$ in the worst case, thus $W_{seek}(m) = a_1 m + a_2 \sqrt{mN}$ from (4.3). The track switch t_{rw} is incurred if B is greater than the track size and t_{return} is equal to $a_3 + a_4 N$ from (4.1).

Rotational latency t_{rot} is assumed to be the time for the desired disk block to rotate into position under the disk head after the right track is found. If t_{track} denotes the time for one disk rotation, and $track$ and $block$ denote the sectors per track and sectors per block respectively, then the rotational latency can be found by

$$(5.6) \quad t_{rot} = t_{track} * \left(\frac{track - ((block - 1) \bmod track)}{track} \right).$$

For example, if the block size is equal to one whole track, the worst-case rotational latency will be equal to the time of rotation across a single sector. This

function will become more complicated if we take spare sectors into account. Assume that spare sectors are located at the end of certain tracks and *spare* denotes the number of spare sectors per track. Then, if the desired block is located at a track which has a spare region and *block* denotes the sectors per block of a normal track, the rotational latency will become

$$t_{rot} = t_{track} * \left(\frac{track - ((block - spare - 1) \bmod track)}{track} \right).$$

However, since the number of physical sectors dedicated for sparing is relatively small in modern disk drives⁵, we will use (5.6) as the rotational latency in the rest of our analysis.

With all the overheads mentioned above, the total overhead function of FSCAN becomes

$$(5.7) \quad H(P, B) = a_1 m + a_2 \sqrt{mN} + m \cdot (t_{rot} + t_{rw} + t_{ts}) + (a_3 + a_4 N) + n \cdot \frac{B}{DTR}.$$

Using the results from (5.5) to (5.7), for given P and B , the maximum throughput rate R_{max} that the file system can support is given by

⁵ According to the data given in [Ruem94], only 0.88% and 1.3% of physical sectors are reserved for sparing regions in HPC2200A and HP97560 disk drives respectively.

$$\begin{aligned}
R_{max} &= \frac{P - H(P, B)}{P} \cdot DTR \\
(5.8) \quad &= \frac{DTR}{P} \left(P - (a_1 n + a_2 \sqrt{mN} + m(t_{rot} + t_{rw} + t_{ts}) + (a_3 + a_4 N) + n \cdot B/DTR) \right).
\end{aligned}$$

Substituting $m = (P \cdot R_{max})/B + n$ into (5.8), we can find R_{max} by solving a quadratic equation, given below:

$$\begin{aligned}
(5.9) \quad R_{max} & \left(1 + \frac{DTR(a_1 + t_{rot} + t_{rw} + t_{ts})}{B} \right) + \sqrt{R_{max}} \left(a_2 DTR \sqrt{\frac{N}{BP}} \right) \\
& + \frac{DTR}{P} \left(n(a_1 + t_{rot} + t_{rw} + t_{ts}) + a_2 \sqrt{nN} + a_3 + a_4 N + \frac{nB}{DTR} - P \right) = 0.
\end{aligned}$$

Lemma 3 Given a stream system $S = \{S_1, S_2, \dots, S_n\}$ and the system parameters P and B , S is schedulable if the total requested throughput rate $R = \sum_{i=1}^n R_i$ is less than the solution for R_{max} in (5.9).

Proof. Since R_{max} accurately accounts for the overheads the file system can support, therefore if a data stream requests less than R_{max} in throughput then it is schedulable. $\square$

With the above analysis, we find the worst possible disk overhead incurred by FSCAN, which in turn allows us to find the maximum throughput the file system supports. Now, we express R_{max} in terms of a factor $\beta(P, B) = R_{max}/DTR$. We call $\beta(P, B)$ the *effective schedulability factor* of the real-time file system. This factor describes the transfer rate of the file system

supports divided by the maximum disk transfer rate. It can also be interpreted as the maximum disk transfer utilization when FSCAN is performed. With this, let U represent the utilization of the stream system compared to the maximum disk transfer rate. Then we can do the following schedulability test:

Theorem 4 Given the system parameters P and B , a stream system of S is schedulable if $U \leq \beta(P, B)$ and if $BUF(P, B)$ bytes of memory are available.

Proof. Since the effective schedulability $\beta(P, B)$ represents the maximum disk utilization of the file system, a data stream set S with total utilization smaller or equal to $\beta(P, B)$ would be schedulable. In addition, the memory requirement must also be satisfied for the double buffering technique to work. $\square$

6. Performance Analysis and Evaluation

In this chapter we evaluate the FSCAN algorithm with real disk parameters. For this purpose the file system is configured with the disk parameters listed in Table 6.1. We first choose the HP97560 SCSI disk for our file system and evaluate FSCAN with some surface plots of the effective schedulability factor $\beta(P,B)$. Then, other disks are used to make performance comparisons.

Disk Type	HP97560 [Ruem94]	HPC2200A [Ruem94]	HPC2247 [Wort94]	Fujitsu M2361A [Selt91]
Sector Size (Bytes)	512	256	512	512
Number of Cylinders N	1962	1449	2051	840
Tracks/Cylinder	19	8	13	20
Sectors/Track	72	113	56-96 (assume 56)	67
Revolution Speed (RPM)	4002	4002	5400	3600
R/W Overheads (ms)	2.2	5.1 (write)	1.3 (write)	5.0
Track Switch Time (ms)	1.6	2.5	2.7	3.5
Data Transfer Rate (MB/s)	2.5	1.9	2.6	2.0
Seek Time (ms)	$3.24 + 0.400\sqrt{d} \quad d < D$ $8.00 + 0.008d \quad D \leq d \leq N$ $D=383$	$3.45 + 0.597\sqrt{d} \quad d < D$ $10.8 + 0.012d \quad D \leq d \leq N$ $D=616$	$3.81 + 0.330\sqrt{d} \quad d < D$ $7.75 + 0.0059d \quad D \leq d \leq N$ $D=300$	$4.6 + 0.87\sqrt{d} \quad d < D$ $11.3 + 0.028d \quad D \leq d \leq N$ $D=239$

Table 6.1: Parameters of some disk drives.

Our goal is to plot the effective schedulability factor $\beta(P,B)$, i.e. the maximum file system throughput over the maximum disk transfer rate. For the HP97560 disk, Figures 6.1 to 6.3 show different surface plots of $\beta(P,B)$. The number of data streams is 6 but is varied later. Figure 6.1 plots $\beta(P,B)$ for various values of the scan period P and the block size B . The surface represents the maximum ratio of the total throughput rate to the DTR that the file system

can support. Any set of 6 data streams with a total requested throughput rate larger than $DTR \cdot \beta(P, B)$ for given P and B would not be schedulable. In general, $\beta(P, B)$ increases as P or B increases. However, note that there is a large drop of $\beta(P, B)$ when the block size increases across the boundary of 1 track size ($72 \times 512B \approx 37$ KB in this case). This drop is due to the sudden increase of the rotational latency when the block size is slightly larger than 1 track according to equation 5.6. When the block size is equal to 1 track, $\beta(P, B)$ reaches to 0.6; when B is equal to 2 tracks, $\beta(P, B)$ approaches to 0.75. This represents a throughput of 1.5 to 2.0 MB/s for the HP97560.

Figure 6.2 shows the maximum memory buffer requirement for FSCAN with different P and B . Again, there is a drop of $\beta(P, B)$ when B is slightly larger than 1 track due to the sudden increase of rotational latency according to (5.6). Combining Figure 6.1 and 6.2 gives Figure 6.3, which shows the relationship of $\beta(P, B)$ and the corresponding buffer requirement. For instance, if we want to have an effective schedulability factor β equal to 0.6 with block size equal to 2 tracks, there must be approximately 4 MB memory buffer available for the file system. Some values of P and B are not possible in this figure because there is insufficient buffering to support the scan period.

Figure 6.1: Effective schedulability factor $\beta(P, B)$ of $FSCAN(P, B)$ with HP97560.

Figure 6.2: Maximum buffer requirement in $FSCAN(P, B)$ with HP97560.

Figure 6.3: Maximum buffer required for $\beta(P, B)$ with HP97560.

The breakdown of different overhead components for $FSCAN(P, B)$ is shown in Figures 6.4 and 6.5. With a block size equal to half of a track (Figure 6.4), rotational latency is the most dominant overhead when P is larger than 400 ms. The next dominant overhead is seek, which is around 30% to 40% of the total overhead. However, if we set the block size equal to one track, the rotational latency drops substantially according to (5.6). Hence, as shown in Figure 6.5, rotational latency becomes the least significant component among all the overheads and the seek latency becomes the most dominant factor.

Figure 6.4: Overhead components when block size is one half of a track.

Figure 6.5: Overhead components when block size is a whole track.

Figure 6.6 shows how $\beta(P, B)$ varies according to P and the number of data streams n . From the plot, we note that n is not a dominant factor; the effective

schedulability factor decreases slowly as n increases. From the analysis in the last chapter, we know that schedulability depends substantially on the overhead function $H(P,B)$ given by (5.9), which in turn depends greatly on the total number of disk requests m . Since we found the upper bound of m with R as the dominant factor (rather than n) for a given P and B in (5.3), we obtain a good understanding of $\beta(P,B)$ with a fixed value of n . Similarly, the buffer requirement for FSCAN varies slightly with differing numbers of data streams, as shown in Figure 6.7. Because the curves do not depend very much on n , we used fixed n in Figures 6.1 to 6.5.

Figure 6.6: $\beta(P,B)$ with differing numbers of data streams when block size equals to one track.

Figure 6.7: Maximum buffer requirement in FSCAN(P,B) with differing number of data streams when block size equals to one track.

We now examine the behaviour of FSCAN with the different disk drives given in Table 6.1. We have made worst-case assumptions with the zoned disk HPC2247: the number of sectors per track is assumed to be the lowest possible, i.e. 56, and the data transfer rate is calculated accordingly. Figure 6.8 plots the effective schedulability as a function of scan period with a block size of 1 track and 6 data streams. The maximum buffer required for different configurations is shown in Figure 6.9. As we can see, the performance of FSCAN varies with different disk drives. The major determinant of performance is the worst-case seek time. Overhead due to rotational latencies are minor when a whole track is read. The HP97560 drive has a relatively fast seek ability and therefore it

outperforms the other disks. Although the HPC2247 drive has the fastest seek ability it does not perform as well as HP97560 because of its large cylinder size and because a worst-case zoning assumption is used.

Figure 6.8: Effective schedulability for 1-track transfer on different disks.

Figure 6.9: Maximum buffer required for 1-track for different disks.

As an illustration for how our real-time file systems works in real applications, if we use the HP97560 drive in our file system, transfer 1 track at a

time and set the scan period to 1000 ms, then the effective schedulability is $\beta(1000,1) = 0.52$ from Figure 6.8. Note the scan period also represents the worst-case start-up latency of the data streams. The corresponding buffer requirement is approximately 3 MB according to Figure 6.9. Since *DTR* is 2.5 MB/s (or 20 Mbps) for the HP97560, the file system can support $\beta(1000,1) \cdot DTR = 10.4$ Mbps of throughput. In Chapter 1, we proposed that a useful hard real-time system should support approximately 10 Mbps of concurrent data transfer from 6 separate streams. Thus, the FSCAN algorithm meets the throughput goals of this thesis.

7. Software Development

In this chapter we describe software that tests a hard disk and to automatically configures the FSCAN algorithm performance parameters. The software first performs a series of seek tests to extract detailed information from the hard disk such as the zoning layout. It then finds the worst-case CSCAN seek function by applying Theorem 1. After obtaining this data, the program configures the file system for use by the FSCAN scheme.

The software runs under DOS and is written in the C++ language. Most modules are portable to other platforms except the DOS-specified sub-modules for low-level SCSI disk access and timing utilities. The software consists of approximately 3500 lines of commented C++ code (including some lines of in-line assembler code) and is broken into three main modules: the Disk Scan Module, the Worst CSCAN Seek Module, and the FSCAN Configuration Module. A sample run of the software and instructions for using it are included in the appendix. The three module are described in the following sections.

7.1 Disk scan

The Disk Scan Module is a program which extracts the detailed layout of the zone-bit recording of a modern disk, such as the number of zones, the

number of sectors per track in different zones, the number of cylinders per zone, and the number of spare sectors. It also reports any TCAL occurrence during the test. All this information is stored in a data file for further usage by the other two modules.

The basic methodology of this program is to time the sector jumps (seeks from a sector to the next adjacent one) for the entire logical sector range in consecutive sequence. Three different kinds of sector jumps are possible: sector-to-sector seeks within the same track (*sector seeks* as below), head switches across disk surfaces, and track switches across cylinders. Since a head switch or a track switch takes longer than a sector seek, we can distinguish them by comparing each jump by the threshold value for sector seeks. Figure 7.1(a) shows some TTY debug messages generated by the program running for the Micropolis 4110 drive. By examining the sector jump time measurements shown in the figure, we find that the sector jump from logical sector 1141675 to 1141676 is either a head switch or track switch. Figure 7.1(b) shows the elapsed time distribution of 2000 sector jumps recorded by the program. The type of each sector jump can be identified easily: The three samples at the top are track switches as they have the longest elapsed time; the 18 samples in the middle are head switches; and the rest are sector seeks. Thus, by keeping track of the elapsed time for all sector jumps, the physical layout of the disk such as sectors

per track in different zones can be extracted. We will explain this in more detail in later paragraphs.

...		
S:1141672	Time=2.089383 ms	
S:1141673	Time=2.043288 ms	
S:1141674	Time=2.042450 ms	
S:1141675	Time=2.042450 ms	
S:1141676	Time=2.751482 ms	← Head Switch or Track Switch
S:1141677	Time=2.040774 ms	
S:1141678	Time=2.039935 ms	
S:1141679	Time=2.040774 ms	
S:1141680	Time=2.039935 ms	
S:1141681	Time=2.040774 ms	
...		

(a) Some elapsed time measurements of sector jumps.

(b) Distribution of sector jump elapsed time.

Figure 7.1: Elapsed time measurements of 2000 sector jumps for the Micropolis 4110 disk drive.

To find the threshold for sector seeks, the program measures the elapsed time for 2000 consecutive sector jumps at the middle range tracks and finds the individual jump times and the mean sector jump time. It also finds the time difference for each two successive jumps and calculates the standard deviation ($\sqrt{\sigma}$) of all the jump time differences. Assuming that all the elapsed time of sector jumps are normal distributed and switches are rare compared to sector seeks, then 99.7% of the normal sector jump samples should lie within 3 times of the standard deviation [Devo82]. Real measurements show that $3\sqrt{\sigma}$ is a good threshold for the sector seeks. Hence, the threshold is set to this value for our purpose.

For an illustration, the mean of 2000 sector jump samples is 2.0537 ms and the standard deviation of the time differences of two successive jumps equal to 0.1026 ms, as presented in Figure 7.2(a). These data are generated by the program and stored in an output file (see Appendix). The validation of choosing $3\sqrt{\sigma}$ or 0.3081 ms as the threshold can be shown in Figure 7.2(b), which plots the distribution of the time difference of two successive sector jumps for the Micropolis 4110 drive. From this figure, it is clearly shown that the threshold is assigned appropriately. This threshold will be used in further testing of the program. Similarly, to distinguish between track switches and head switches,

we can exclude all sector seeks and then perform the same statistical technique to find a threshold.

```

Number of Sector Jump samples = 2000
Start Sector = 1140700

Sector Jumps:
Sample Mean          = 2.053685 ms
95% Confidence Interval = 0.003416 ms

Time Differences of Two Successive Jumps:
Sample Mean          = 0.044736 ms
Standard Deviation(SD) = 0.102692 ms

Set Threshold = 3 * SD = 0.308076 ms
  
```

(a) Statistical results of 2000 sector jump samples.

(b) Distribution of the time difference of two successive sector jumps.

Figure 7.2: Statistics of 2000 sector jump samples for the Micropolis 4110 disk drive.

Next, the program performs the sector jump test for the entire sector range in consecutive order and finds the time difference between each jump and the mean value found in the previous test. If the time difference is larger than the threshold, then it must be a head switch or a track switch. Track switches require more time than head switches. If h denotes the number of surfaces (or heads) of the disk, then a track switch will occur every $h-1$ times of head switches. For instance, in Figure 7.1(b), there are eight head switches between every two track switches, since the Micropolis 4110 drive has nine heads. The number of sectors in between two successive head switches is the number of sectors per track in that disk zone. Similarly, the number of track switches represents the number of cylinders of the disk. In some disks, such as the ones we have tested (the Micropolis 4110 and Quantum LPS540S drive), the spare sectors are located at the end of the last track of each cylinders. In other words, this last track will have a smaller number of sectors than other tracks in that zone. The program also records this and outputs to a data file. When the number of sectors per track is different from what has previously been recorded, then the disk head must have reached another disk zone. Thus, we can get the zoning details of the entire disk. In particular, we find that the Micropolis 4110 drive has 10 disk zones and the number of sectors per track ranges from 70 to

114. The zoning details of the Quantum LPS540S drive is also found: it has 15 disk zones and the track size ranges from 62 to 125 sectors.

The whole test requires approximately one hour for a 1 Gbyte disk. Since the disk is operating during the entire test, TCALs may occur. The program will detect this abnormality and report it. This program assumes a TCAL has occurred if the elapsed time for a sector jump is greater than a pre-defined macro value (*TCAL_TIME*). In our program, this value is set to 100 ms. The scan test for the Micropolis 4110 drive takes around an hour and there are four TCALs occurred during the test. On the other hands, no TCAL is reported during the half-hour scan test for the Quantum LPS540S drive.

After all the tests, the program writes all the relevant information and the TCAL report to a data file, which will be used by the next two modules. The data file generated by the program for the Micropolis 4110 drive and the Quantum LPS540S are shown in the appendix.

7.2 Worst CSCAN seek test

The Worst CSCAN Seek Module finds the worst-case CSCAN seek time of a disk. Based on the disk parameters obtained by the Disk Scan Module, this module performs a series of seek tests to find an upper bound on the worst-case CSCAN seek time for the disk. This bound can be found easily and accurately

by a simple algorithm based on Theorem 1. Recall that the worst-case time to perform a series of n seeks across the disk in CSCAN scheme is bounded by the total seek time with all seek locations equally spaced across the entire range of cylinders for a non-decreasing concave seek function. With this theorem, for n disk requests, the worst-case CSCAN seek time can be found by measuring the total elapsed time of n equal-distance seeks across the disk in one sweeping direction. To reduce random effects, a 95% confidence interval upper bound of the worst-case curve is computed. Figure 7.3 shows these bounds for two modern disks.

Note that in Figure 7.3 there is a sudden decrease in the worst-case seek time for the Quantum LPS540S drive when total number of seeks is above 134. This drop is due to the abnormal behaviour of the seek function of the drive as shown in Figure 7.4. The seek time of the drive in three different disk zones is plotted. Each point on the curve represents the mean elapsed time for 50 samples of seeks. Observe from the figure that there is a sudden increase of seek time when seek distance increases from 20 to 21 cylinders. The number of cylinders of the Quantum LPS540S drive is 2740 (refer to Appendix). Thus the distance travelled by each of the 134 seeks in Figure 7.3 is $2740/134$ or 21 cylinders. Similarly, the number of cylinders travelled by each of the 135 seeks is $2740/135$ or 20. Since the seek time for travelling 21 cylinders is abnormally

large compared to the time required for seeking across 20 cylinders, a sudden decrease in worst-case seek time results. We believe there may be a non-linear control algorithm in the Quantum LPS540S drive controller software. On the other hand, Figure 7.3 implies a good seek time behaviour of the Micropolis 4110 drive because the drive has a smooth non-decreasing concave worst-case seek curve.

Figure 7.3: Worst-case CSCAN seek curves and their 95% confidence interval upper bounds for the Micropolis 4110 and Quantum LPS540S drives.

Figure 7.4: Mean Seek time curve for 50 measurements of the Quantum LPS540S drive in different zones.

The abnormal seek behaviour shown in Figure 7.4 is not uncommon for disk drives. Disk drive manufacturers typically employ two or more seeking algorithms in their drives [Teja95]. These algorithms have different acceleration and deceleration profiles, depending on the seek distance. If two or more algorithms were not used, then in some cases the disk arm might not be able to reach to the desired track efficiently. The problem arises from disk drive actuators (arms). Today, nearly all disk drive actuators use greased bearings

which are meant to spin continuously. Unfortunately, the bearings in an actuator make small moves only. As a consequence, the disk may get a "sticktion" problem with the disk arm, just like the "sticktion" problems on the spindle. For some longer seeks, the arm will actually go past the desired track, and then return a short distance to position the head correctly. For shorter seeks, the arm may not have to overshoot the track [Teja95].

However, this non-perfect seek behaviour is not desirable in our seek function model. According to our model, we claim that all curves in Figure 7.3 should be non-decreasing concave functions, because they are summations of non-decreasing concave functions of seeking. Hence, if there exist any decreasing portions in these curves, they should be rounded upwards. In particular, the program first finds the 95% confidence upper bound for the worst-case seek times, then upward rounding is applied. The algorithm to compute the rounded concave curve is simple: Let m be the number of seeks and $w(m)$ denote the curve. Begin with the $m = 0$, the program tests every other points on $w(m)$ and finds the slope of every chords. The chord with the largest slope is identified. Every point on $w(m)$ which lies within the domain of this chord is rounded upwards to this line. The program then picks the end-point of the chord as a new starting point and repeats the above computation until the largest m is reached. Figure 7.5 illustrates an example. This figure shows a

curve which consists of concave-up and concave-down intervals. Assume A is the starting point. The slope of the chords which connects A to every other points on the curve with m greater than m_A are found. Then the curve is rounded upwards to the chord which has the largest slope (line AF in the example). Next, point F becomes the next starting point and the process repeats. With this algorithm, the rounded curves of the worst CSCAN seek functions for the two disks shown in Figure 7.3 are found. They are presented in Figure 7.6. These curves will be used as the upper bounds of the worst-case CSCAN seeks in the FSCAN Configuration Module.

Figure 7.5: An example showing the rounding-up of a curve.

Figure 7.6: The rounded upper bound of worst-case CSCAN seek curves for the Micropolis 4110 and Quantum LPS540S drives.

7.3 FSCAN configuration

The FSCAN Configuration Module use the information extracted by the previous modules to configure the disk drive for the FSCAN scheme. In

particular, this module configure the disk drive so that the real-time file system can perform the efficient schedulability test of Theorem 4 in Chapter 5.

However, in order to perform the schedulability test, we need to reformulate the theorem slightly to adopt to arbitrary seek time functions. In Chapter 5, the FSCAN overhead function is defined as

$$H(P, B) = (a_1 m + a_2 \sqrt{mN}) + m \cdot (t_{rot} + t_{rw} + t_{ts}) + (a_3 + a_4 N) + n \cdot \frac{B}{DTR}.$$

Since the seek function mechanical constants a_1 , a_2 , a_3 and a_4 are not known, we cannot use this formula to find the total overhead of the FSCAN algorithm. Instead, we use the rounded-up worst-case CSCAN curve obtained in the Worst CSCAN Seek Module.

Recall that m denotes the number of disk requests, let $W_{seek}(m)$ represent the worst-case CSCAN curve found by measurement, we rewrite (5.7) as

$$(7.1) \quad H(P, B) = W_{seek}(m) + m \cdot (t_{rot} + t_{rw} + t_{ts}) + W_{seek}(1) + n \cdot \frac{B}{DTR}.$$

Then, for given P and B , the maximum throughput rate R_{max} that the file system can support is given by

$$(7.2) \quad \begin{aligned} R_{max} &= \frac{P - H(P, B)}{P} \cdot DTR \\ &= \frac{DTR}{P} \left(P - (W_{seek}(m) + m(t_{rot} + t_{rw} + t_{ts}) + W_{seek}(1) + n \cdot B/DTR) \right). \end{aligned}$$

Recall that the worst-case value of m is denoted by (5.3), given as $m = (P \cdot R_{\max})/B + n$. However, since the domain of $W_{\text{seek}}(m)$ are integers, we cannot use (5.3). Instead, we round m upwardly to $\lceil P \cdot R_{\max}/B + n \rceil$ to determine $W_{\text{seek}}(m)$. On the other hand, the second m in (7.2) can be substituted by (5.3) to represent the worst-case value. After these substitutions, equation (7.2) is simplified to

$$R_{\max} = \left(\frac{B \cdot DTR}{P(B + DTR \cdot (t_{\text{rot}} + t_{\text{rw}} + t_{\text{ts}}))} \right) \times \left[\left(P - n(t_{\text{rot}} + t_{\text{rw}} + t_{\text{ts}}) - W_{\text{seek}}(1) - \frac{n \cdot B}{DTR} \right) - W_{\text{seek}}(\lceil P \cdot R_{\max}/B + n \rceil) \right].$$

For given values of P and B , we can solve the above equation for $R_{\max}$ by bisection algorithm because the right side decreases monotonically with R . Finally, we are able to find the effective schedulability factor $\beta(P, B)$, which represents the maximum file system throughput over the maximum disk transfer rate, given as $\beta(P, B) = R_{\max}/DTR$.

There is one last consideration. Recall that since the track size of the Micropolis 4110 drive varies in different disk zones (70 to 114 sectors), the data transfer rate of the disk also varies with the zone position. For simplicity, the program assumes the worst-case data transfer rate of the disk, just as we did the in Chapter 6 for the zoned disk HPC2247. The program asks the user to input

the revolution speed of the disk drive and the sector size. For the Micropolis 4110 drive, the former is 5400 RPM (or 90 Hz) and latter is 512 bytes. Thus the maximum data transfer rate of the Micropolis 4110 drive is assumed to be $90 \times 512 \times 70 = 3.23$ MB/s.

As an illustration, Figure 7.7 plots $\beta(P, B)$ of the file system using the Micropolis 4110 disk drive with 6 data streams. The maximum buffer requirement for this file system is shown in Figure 7.8. Both surface plots are similar to those shown in Chapter 6 in which the HP97560 drive is used in the file system (see Figure 6.1 and 6.2).

Figure 7.7: Effective schedulability factor with the Micropolis 4110 disk drive.

Figure 7.8: Maximum buffer requirement with the Micropolis 4110 disk drive.

For better representation, Figure 7.9 plots $\beta(P,B)$ against scan period with block size equals to 1 track. The corresponding buffer requirement of the system is also plotted. The user can decide the block size and the scan period (or the maximum start-up latency) for the FSCAN. Suppose the block size is set to be 1 track and the scan period is chosen as 850 ms. Then, for a file system with 6 data streams, the effective schedulability factor is computed to be 0.50, as shown in Figure 7.9. The corresponding buffer memory required is found to be 3 MB. In other words, the file system can support a maximum throughput of $\beta(850,1) \cdot DTR$ or $0.50 \times 3.23 = 1.62$ MB/s (or 12.92 Mbps). Hence, with this configuration the real-time file system meets the throughput goals of this thesis, using an off-the-shelf disk drive.

Figure 7.9: $\beta(P, 1 \text{ track})$ for the Micropolis 4110 drive and the corresponding buffer requirement.

8. Conclusions and Future Work

8.1 Conclusions

We have examined the problem of scheduling disk requests in a stand-alone personal hard real-time read/write file system. Previous study in [Cheng95] showed that it can be very inefficient to optimally service disk requests for data laid down randomly on the disk and any optimal algorithm in that environment can be forced to thrash very badly. These results motivate a heuristic approach to the problem. We proposed an integrated fixed-period SCAN (FSCAN) algorithm for scheduling hard real-time data streams. The approach was parameterized by disk block size and memory buffer size, and traded disk fragmentation and memory for higher performance. We derived the worst-case seek and rotational overhead for the FSCAN algorithm, and we showed that the seek overhead can be measured empirically for a large class of seek functions. Results showed that the FSCAN policy can efficiently transfer data at 40-70% of the maximum data throughput for modern disk drives, depending on the file system configuration.

Our analysis allowed us to build a hard real-time file system without having to obtain proprietary and sensitive information from the disk drive

manufacturer. Software was developed to automatically test and configure the FSCAN algorithm using the techniques suggested by Theorem 1. As a result of this work, we could build a file system using the Micropolis 4110 drive and 3 MB of memory that supports approximately 13 Mbps of throughput.

8.2 Future work

The use of an aperiodic server to serve non-real-time requests is also a logical extension of this work [Leho87, Spru89]. The challenge is how to efficiently co-operate an aperiodic server with the present periodic FSCAN policy. In particular, the important goal is to retain the deterministic schedulability and high throughput rate for hard real-time data streams and, at the same time, to provide a fast service response for non-real-time or soft real-time data requests. In addition, in practice, we can improve the performance for serving aperiodic requests by picking disk blocks of aperiodic streams during the CSCAN returning sweep. In other words, an outer-to-inner sweep will serve periodic requests and an inner-to-outer returning sweep will serve aperiodic requests. The amount of time available for serving non-real-time disk blocks during the returning sweep can be maximized by reclaiming the time resource left unused by the periodic requests. Resource reclaiming is possible since real-

time requests usually takes less time than their worst-case computation time in practice [Shen90].

At present, the performance of the FSCAN scheme is promising when a large block size is chosen. Since the real-time file system may be used to store small data files as well, disk fragmentation may result. Therefore, further investigations of this study might use two or more block sizes to reduce the problem of disk fragmentation.

Better approaches to buffer management can also be investigated. For instance, instead of using the double buffering scheme, management techniques such as memory sharing may be employed in order to reduce the buffer requirement of the FSCAN algorithm.

Another research direction is to investigate the use of multiple disks in the file system in order to support future multimedia applications which may require higher data throughput. Problems such as data placement, buffer management and data synchronization will be the key issues to address.

Finally, we believe the optimal disk scheduling problem with a non-zero read time model can be shown to be equivalent to the partition problem which is NP-complete. We are working on a proof of this conjecture.

Bibliography

- [Abbo90] Robert K. Abbott and Hector Garcia-Molina, "Scheduling I/O Requests with Deadlines: a Performance Evaluation," *Proceedings of the 11th IEEE Symposium on Real-time Systems*, Dec. 1990, pp. 113-124.
- [Bitt89] Dina Bitton, "Arm Scheduling in Shadowed Disks," *COMPCOM Spring '89 IEEE Computer Society Int'l Conference*, 1989, pp. 132-136.
- [Burn89] A. Burns, A. Wellings, *Real-time systems and their programming languages*, Workingham England: Addison-Wesley, 1989.
- [Care89] M. J. Carey, R. Jauhari and M. Livny, "Priority in DBMS Resource Scheduling," *Proceedings of the 15th VLDB Conference*, 1989, pp. 397-410.
- [Cheng95] Raymond M. K. Cheng, Mark Greenstreet, Donald W. Gillies, "Blueprint For A Hard Real-Time File System," Technical Report CICS-TR95-05, Centre For Integrated Computer System Design, U. of British Columbia, Aug. 1995.
- [Daig94] Steve J. Daigle and Jay K. Strosnider, "Disk Scheduling For Multimedia Data Streams," *Proceedings of the Conference on High-Speed Networking and Multimedia Computing*, Feb. 1994, pp. 212-223.
- [Denn67] P.J. Denning, "Effects of scheduling on file memory operations," *AFIPS Spring Computer Conference*, Apr. 1967, pp. 9-21.

-
- [Devo82] Jay L. Devore, *Probability & Statistics for Engineering and the Sciences*, Belmont, Calif: Brooks/Cole, 1982, p. 147.
- [Gemm93] D. James Gemmell, "Multimedia network file servers: Multi-channel delay sensitive data retrieval," *Proceedings of 1st ACM International Conference on Multimedia*, Aug. 1993, pp. 243-250.
- [Geis87] R. Geist, S. Daniel, "A Continuum of Disk Scheduling Algorithms," *ACM Transactions on Computer Systems*, Feb. 1987, pp. 77-92.
- [Furh94] Borko Furht, "Multimedia Systems: An Overview," *IEEE Multimedia*, Vol. 1, No. 1, Spring 1994.
- [Jaco91] D. Jacobson, J. Wilkes, "Disk Scheduling Algorithms Based on Rotational Position," Hewlett-Packard Technical Report, HPL-CSP-91-7, Feb 26., 1991.
- [Jeff91] Kevin Jeffay, Donald F. Stanat and Charles U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks," *Proceedings of the 12th IEEE Real-Time Systems Symposium*, Dec. 1991, pp. 129-139.
- [Juli95] Egil Juliussen, "Small Computers," *IEEE Spectrum*, Jan. 1995, pp. 44-47.
- [Henn90] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, San Mateo, Calif: 1990, p.558.
- [Holz93] Helen Holzbaur and Jim Hurd, "Lab Report: Hard Drives," *BYTE*, vol. 18, no. 10, Sept. 1993, pp. 176-193.

-
- [Leho87] J. P. Lehoczky, L. Sha and J. K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments," *Proceedings of the 8th IEEE Real-Time Systems Symposium*, Dec. 1987, pp. 261-270.
- [Lin91] T. H. Lin and W. Tarn, "Scheduling Periodic and Aperiodic Tasks in Hard Real-Time Computing Systems," *Proceedings of SIGMETRICS*, May 1991, pp. 31-38.
- [Liu73] C. L. Liu and James W. Layland, "Scheduling Algorithms for Multiprogrammings in a Hard Real-time Environment," *Journal of the ACM*, Vol. 20, No. 1, Jan. 1973, pp. 46-61.
- [Nasi95] Panos Nasiopolous, personal communication, Apr. 1995.
- [Ng91] Spencer W. Ng, "Improving Disk Performance Via Latency Reduction," *IEEE Transactions on Computers*, Vol. 40, No. 1, Jan. 1991.
- [Redd93] A. L. Narasimha Reddy and Jim Wyllie, "Disk scheduling in a multimedia I/O system," *Proceedings of 1st ACM International Conference on Multimedia*, Aug. 1993, pp. 225-233.
- [Ruem93] Chris Ruemmler and John Wilkes, "UNIX Disk Access Patterns", *Winter USENIX '93*, Jan. 1993, pp. 405-420.
- [Ruem94] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modelling," *IEEE Computer*, Mar. 1994, pp. 17-28.
- [Sale91] Mofreh M. Salem, "Computers Operating Systems: V-SSTF Dynamic Disk Scheduling Technique," *Third International Conference on Software Engineering for Real Time Systems*, Sept. 1991, pp. 254-259.

-
- [Selt90] Margo Seltzer, Peter Chen and John Ousterhout, "Disk Scheduling Revisited," *USENIX Winter Conference*, 1990, pp. 313-324.
- [Sha90] Lui Sha, Ragunathan Rajkumar, John P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, Vol. 39, No. 9, Sept. 1990, pp. 1175-1185.
- [Shen90] Chia Shen, Krithi Ramamritham, John A. Stankovic, "Resource Reclaiming in Real-Time," *Proceedings of the 11th IEEE Real-Time Systems Symposium*, Dec. 1990, pp. 41-50.
- [Spru89] B. Sprunt, L. Sha and J. P. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems," *Journal of Real-Time Systems*, Vol.1, No.1, 1989, pp. 27-60.
- [Teja95] Fred Tejata, personal communication, Sept. 1995.
- [Tind93] K. Tindell, A. Burns, "Scheduling Hard Real-Time Multi-Media Disk Traffic," Technical Report YCS204, Dept. of Computer Science, U. of York, Jul. 1993.
- [Toba93] F. A. Tobagi, J. Pang, R. Baird, M. Gang, "Streaming RAIDTM -- A Disk Array Management System For Video Files," *ACM Multimedia*, Aug. 1993, pp. 393-400.
- [Wort94] Bruce L. Worthington, Gregory R. Ganger, Yale N. Patt, "Scheduling for Modern Disk Drives and Non-Random Workloads," Technical Report CSE-TR-194-94, Dept. of Electrical Engineering and Computer Science, U. of Michigan, Mar. 1994.

Appendix: Sample Runs of the Configuration Software

In this appendix, sample runs of the FSCAN configuration software and the instructions for using it are presented. The data files generated by the software are also shown.

The software consists of two executable programs: DISKSCAN and FSCAN. They run under DOS and are written in the C++ language. They are designed for testing SCSI disk drives only. The two programs and their outputs are described below.

A.1 The DISKSCAN program

DISKSCAN is a program which implements the Disk Scan Module and the Worst CSCAN Seek Module. It first executes a series of sector jump tests for the entire sector range, extracts the detailed layout of the disk, performs a series of seek tests, and then finds the round-up bound on the worst-case CSCAN seek time for the disk. The program takes three parameters at the DOS prompt: the optional flag, the drive letter of the disk to be tested, and the name of the output file. A error message is returned if the user has entered an incorrect list of parameters. The optional flag `-d` specifies the program to output some debug messages onto the screen. For example, at DOS prompt, a command

```
diskscan -d D config_d.ini
```

will execute DISKSCAN to test drive D, output the data to file config_d.ini and output debug messages to the screen. The whole program takes approximately one hour for a 1-GB disk. Part of the TTY output of this program is shown as follows:

```
C:\FSCAN>diskscan
usage: c:\fscan\diskscan.exe [-d] DriveId Outfile
flag : -d = debug message
DriveId : 'C' or 'D'
Outfile : output file to store disk information
```

```
C:\FSCAN>diskscan -d D config_d.ini
```

```
+-----+
| DISKSCAN v1.0                               |
|                                             |
| Extract the disk layout details.           |
|                                             |
| All Right Reserved.                       |
+-----+
```

```
CAUTION: THIS TEST TAKES ABOUT 1 HOUR FOR A 1 MB DISK.
THIS PROGRAM ONLY WORKS FOR SCSI DISKS.
```

```
Preparing for Disk Scanning Test ...
```

```
[Debug] Mean Seek = 2.066860, Threshold = 0.314229
[Debug] Number of Heads = 9
```

```
Start Disk Scanning ...
```

```
[Debug Messages]
L:1073 Offset= 114; Time= 2.614034 ms, Diff= 0.547174 ms
L:1187 Offset= 114; Time= 2.694492 ms, Diff= 0.627632 ms
L:1301 Offset= 114; Time= 2.613196 ms, Diff= 0.546336 ms
L:1415 Offset= 114; Time= 2.652587 ms, Diff= 0.585727 ms
L:1529 Offset= 114; Time= 2.694491 ms, Diff= 0.627632 ms
L:1643 Offset= 114; Time= 2.695330 ms, Diff= 0.628470 ms
L:1757 Offset= 114; Time= 2.694491 ms, Diff= 0.627632 ms
L:1871 Offset= 114; Time= 2.733044 ms, Diff= 0.666184 ms
L:1979 Offset= 108; Time= 3.049846 ms, Diff= 0.982986 ms
L:2093 Offset= 114; Time= 2.613196 ms, Diff= 0.546336 ms
L:2207 Offset= 114; Time= 2.653425 ms, Diff= 0.586565 ms
L:2321 Offset= 114; Time= 2.614034 ms, Diff= 0.547174 ms
L:2435 Offset= 114; Time= 2.695330 ms, Diff= 0.628470 ms
L:2549 Offset= 114; Time= 2.614034 ms, Diff= 0.547174 ms
L:2663 Offset= 114; Time= 2.692815 ms, Diff= 0.625956 ms
L:2777 Offset= 114; Time= 2.652587 ms, Diff= 0.585727 ms
L:2891 Offset= 114; Time= 2.693653 ms, Diff= 0.626794 ms
L:2999 Offset= 108; Time= 3.012131 ms, Diff= 0.945272 ms
L:3113 Offset= 114; Time= 2.655101 ms, Diff= 0.588241 ms
L:3227 Offset= 114; Time= 2.614034 ms, Diff= 0.547174 ms
```

```
<Truncated>
```

Two SCSI disks are tested by the DISKSCAN: the Micropolis 4110 drive and the Quantum LPS540S drive. The data files generated by the program are shown below. The first data file is for the Micropolis 4110 disk drive with a disk ID of 0x81 and the second one is for the Quantum LPS540S drive with a disk ID of 0x80.

A.1.1 Data file for the Micropolis 4110 drive

```
# DiskScan data file
#
# Created at Fri Sep 15 23:43:01 1995

# Pre-scanning test statistics:
# Number of sample = 2000
# Start Sector = 1140700
#
# Sector Jumps:
# Sample Mean = 2.066860
# Confidence Interval = 0.003314
#
# Time Difference of Two Successive Jumps:
# Sample Mean = 0.022418
# Standard Deviation = 0.104743
# Set Threshold = 3 * SD = 0.314229

# Disk Scanning :
# Total time taken = 4235.778692 sec

# Disk Information (all time unit in millisecond unless specified)
DiskId 81 H
Cylinders 2413
Tracks/Cylinder 9
Sectors/Track 70 -- 114

HeadSwitch(outer_zone) 2.677136
HeadSwitch(inner_zone) 2.915826

TrackSwitch(outer_zone) 3.019884
TrackSwitch(inner_zone) 3.256717

Number_of_Zones 10
Maximum_LSector 2053818
```

Zoning					
#	Zone	Logical Sector Range	Sectors/track	Last Track	Cylinder Range
#					
0	0	52	53	53	0 0
1	53	231478	114	108	1 227
2	231479	522058	112	106	228 517
3	522059	824308	104	98	518 842
4	824309	1099723	101	95	843 1147
5	1099724	1331383	96	90	1148 1417
6	1331384	1547093	91	86	1418 1682
7	1547094	1721893	85	80	1683 1912
8	1721894	1861293	78	73	1913 2112
9	1861294	1975568	73	69	2113 2287

```

10 1975569 2053818 70 66 2288 2412
/Zoning

```

```

TCAL
# TCAL lsector Time Stamp (sec) Elapsed Time (ms)
# -----
0 61292 122.246862 788.147564
1 527678 1072.419320 303.910984
2 990911 2023.106888 780.173880
3 1449514 2973.902503 787.451103
4 1906470 3924.736636 794.759335
/TCAL

```

```

Worst_CSCAN
# Worst Seek Time with
# No of Seek 95% Confidence Interval Rounded Time
# -----
0 0.000000 0.000000
1 20.437636 20.437636
2 26.717942 26.717942
3 32.936090 33.078556
4 39.409237 39.439170
5 45.799784 45.799784
6 51.990684 51.990684
7 58.001615 58.001615
8 63.255427 63.802982
9 68.985648 69.604350
10 73.999459 75.405717

```

<not shown>

```

391 1660.753320 1660.753320
392 1660.360008 1664.166552
393 1665.783292 1667.579784
394 1668.164942 1670.993016
395 1670.435115 1674.406249
396 1671.636391 1677.819481
397 1681.118150 1681.232713
398 1683.565298 1684.645945
399 1688.059177 1688.059177
400 1687.376076 1687.376076
/Worst_CSCAN

```

A.1.2 Data file for the Quantum LPS540S drive

```

# DiskScan data file
#
# Created at Fri Sep 15 23:43:15 1995

# Pre-scanning test statistics:
# Number of sample = 2000
# Start Sector = 587056
#
# Sector Jumps:
# Sample Mean = 1.151236
# Confidence Interval = 0.007604
#
# Time Difference of Two Successive Jumps:
# Sample Mean = 0.058075
# Standard Deviation = 0.233979
# Set Threshold = 3 * SD = 0.701937

# Disk Scanning :
# Total time taken = 1239.465693 sec

```

Disk Information (all time unit in millisecond unless specified)

DiskId 80 H
Cylinders 2740
Tracks/Cylinder 4
Sectors/Track 62 -- 125

HeadSwitch(outer_zone) 2.326566
HeadSwitch(inner_zone) 2.533309

TrackSwitch(outer_zone) 3.268695
TrackSwitch(inner_zone) 3.452396

Number_of_Zones 15
Maximum_LSector 1057554

Zoning

#	Zone	Logical Sector Range	Sectors/track	Last Track	Cylinder Range
0	0	63	64	64	0 0
1	64	212086	125	123	1 426
2	212087	292762	122	120	427 592
3	292763	370118	117	115	593 758
4	370119	444818	113	111	759 924
5	444819	516198	108	106	925 1090
6	516199	585586	105	103	1091 1256
7	585587	650990	99	97	1257 1422
8	650991	713074	94	92	1423 1588
9	713075	772502	90	88	1589 1754
10	772503	827946	84	82	1755 1919
11	827947	880734	80	78	1920 2085
12	880735	930202	75	73	2086 2251
13	930203	976350	70	68	2252 2417
14	976351	1019178	65	63	2418 2583
15	1019179	1057554	62	60	2584 2739

/Zoning

TCAL

NO TCAL OCCURRED.

/TCAL

Worst_CSCAN

#	No of Seek	Worst Seek Time with 95% Confidence Interval	Rounded Time
0	0.0	0.0	0.0
1	21.789137	21.789137	21.789137
2	31.398306	31.398306	31.398306
3	39.977726	39.977726	39.977726
4	47.665861	47.665861	47.665861
5	55.185590	55.634211	55.634211
6	62.579248	63.602560	63.602560
7	70.846273	71.570909	71.570909
8	78.778130	79.539259	79.539259
9	86.130487	87.507608	87.507608
10	94.017604	95.475958	95.475958

<truncated>

A.2 The FSCAN program

The FSCAN program implements the FSCAN Configuration Module. It first obtains the disk information from the data file generated by DISKSCAN and configures the FSCAN algorithm for the file system with the extracted disk. This program takes the data file name as a parameter at the DOS command prompt. After the program is executed, it first asks the user to enter the revolution speed of the disk in RPM. Next, it asks for the two system parameters: the block size in sectors and the scan period (or start-up latency) in millisecond. The sector size of the disk is assumed to be 512 KB. The program then computes the corresponding effective schedulability factor (denoted by *Beta* in the output) and the buffer requirement as a function of the number of data streams. Part of the results are returned to the screen. Then it asks the user whether he/she would like to output the results to a file in MATLAB® M-file format. The TTY output of a sample run and the corresponding M-file are shown in the following sections.

A.2.1 Sample run of the FSCAN program

The TTY output of a sample run of the FSCAN program is shown in this section. Note that the input entries of the user is highlighted.

```
C:\FSCAN>fscan
usage: C:\FSCAN\FSCAN.EXE datafile
```

```
C:\FSCAN>fscan config_d.ini
```

```
+-----+
| FSCAN v1.0                               |
|                                           |
| Configure Real-Time File System.         |
|                                           |
| All Right Reserved.                     |
+-----+
```

```
Enter the revolution speed of the disk in RPM: 5400
Enter Block Size in sectors (1 track = 70 sectors): 70
Enter Startup Latency (ms): 850
```

FSCAN results:

```
Disk Id           = 81 H
Revolution Speed  = 5400 RPM
Track Size assumed = 70 Sectors
Data Transfer Rate assumed = 3225600 MB/s
Block Size (B)    = 70 Sectors (35840 Bytes)
Start-up Latency (P) = 850.00 ms
```

No of Stream	Max Throughput Rate (MB/s)	Beta	Buffer Required (MB)
1	1813.082353	0.562092	3.118080
2	1770.917647	0.549020	3.082240
3	1728.752941	0.535948	3.046400
4	1686.588235	0.522876	3.010560
5	1644.423529	0.509804	2.974720
6	1602.258824	0.496732	2.938880
7	1560.094118	0.483660	2.903040
8	1517.929412	0.470588	2.867200
9	1475.764706	0.457516	2.831360
10	1433.600000	0.444444	2.795520

```
Want to change Block Size or Start-up Latency? (Y/N) n
```

```
Want to output data to MATLAB format? (Y/N) y
```

```
Enter the file name: mydisk.m
```

```
<Program exit>
```

```
C:\FSCAN>
```

A.2.2 MATLAB® M-file output

The MATLAB® M-file of the sample run shown in the section A.2.2 is shown here. Note that it plots two graphs in MATLAB®, as shown in Figure B.1 and Figure B.2. Figure B.1 contains four plots which show the performance of the FSCAN algorithm with the chosen system parameters. The second graph shows the round-up worst-case CSCAN seek function of the tested disk. The M-file is shown as follows:

```
% +-----+
% | MATLAB file generated by FSCAN v1.0 |
% |                                     |
% | Define disk parameters.           |
% |                                     |
% | All Right Reserved.               |
% +-----+

% Created at Sun Sep 17 18:57:59 1995.

clear ;
% Disk Parameter for Disk 81 H
SECTOR_SIZE      = 512;
CYLINDER         = 2413;
TRACK_PER_CYL   = 9;
SECTOR_PER_TRACK = 70;          % worst-case assumption (70 - 114)
SPEED_RPM       = 5400;
READ_OVERHEAD   = 1.500000;    % ms
WRITE_OVERHEAD  = 2.500000;    % ms
TRACK_SWITCH    = 3.019884;    % ms
HEAD_SWITCH     = 2.677136;    % ms
N=CYLINDER;

% Rounded up worst-case seek function (for Matlab)
Wseek(1) = 20.437636; Wseek(2) = 26.717942; Wseek(3) = 33.078556;
Wseek(4) = 39.439170; Wseek(5) = 45.799784; Wseek(6) = 51.990684;
Wseek(7) = 58.001615; Wseek(8) = 63.802982; Wseek(9) = 69.604350;
.
.
<not shown>
.
.
Wseek(397) = 1681.232713; Wseek(398) = 1684.645945; Wseek(399) = 1688.059177;
Wseek(400) = 1687.376076;

% FSCAN results:
% Track Size assumed = 70 Sectors

% Data Transfer Rate assumed
DTR = 3225600 ; % (MB/s)

% Block Size chosen
B = 35840 ; % (Bytes) (70 Sectors)
```

```

% Start-up Latency chosen
P = 0.850 ; % sec

R_w(1) = 1813082.352941; R_w(2) = 1770917.647059; R_w(3) = 1728752.941176;
R_w(4) = 1686588.235294; R_w(5) = 1644423.529412; R_w(6) = 1602258.823529;
R_w(7) = 1560094.117647; R_w(8) = 1517929.411765; R_w(9) = 1475764.705882;
R_w(10) = 1433600.000000; R_w(11) = 1391435.294118; R_w(12) = 1349270.588235;
R_w(13) = 1307105.882353; R_w(14) = 1264941.176471; R_w(15) = 1222776.470588;
R_w(16) = 1180611.764706; R_w(17) = 1138447.058824; R_w(18) = 1096282.352941;
R_w(19) = 1054117.647059; R_w(20) = 1011952.941176;

Beta(1) = 0.562092; Beta(2) = 0.549020; Beta(3) = 0.535948;
Beta(4) = 0.522876; Beta(5) = 0.509804; Beta(6) = 0.496732;
Beta(7) = 0.483660; Beta(8) = 0.470588; Beta(9) = 0.457516;
Beta(10) = 0.444444; Beta(11) = 0.431373; Beta(12) = 0.418301;
Beta(13) = 0.405229; Beta(14) = 0.392157; Beta(15) = 0.379085;
Beta(16) = 0.366013; Beta(17) = 0.352941; Beta(18) = 0.339869;
Beta(19) = 0.326797; Beta(20) = 0.313725;

BUF(1) = 3118080.000000; BUF(2) = 3082240.000000; BUF(3) = 3046400.000000;
BUF(4) = 3010560.000000; BUF(5) = 2974720.000000; BUF(6) = 2938880.000000;
BUF(7) = 2903040.000000; BUF(8) = 2867200.000000; BUF(9) = 2831360.000000;
BUF(10) = 2795520.000000; BUF(11) = 2759680.000000; BUF(12) = 2723840.000000;
BUF(13) = 2688000.000000; BUF(14) = 2652160.000000; BUF(15) = 2616320.000000;
BUF(16) = 2580480.000000; BUF(17) = 2544640.000000; BUF(18) = 2508800.000000;
BUF(19) = 2472960.000000; BUF(20) = 2437120.000000;

BUF = BUF / 1000000;
R_w = R_w / 1000000;

figure(1); clf;
subplot(2,2,1),plot(Beta), grid, xlabel('Number of streams'),
ylabel('Beta(P=0.850 sec, B=35840 Bytes)'),title('Beta (Disk 81H)');

subplot(2,2,2),plot(BUF), grid, xlabel('Number of streams'),
ylabel('Buffer Requirement (MB)'),title('Buffer Requirement (Disk 81H)');

subplot(2,2,3),plot(R_w), grid, xlabel('Number of streams'),
ylabel('Max. Throughput (MB/s)'),title('Max. System Throughput (Disk 81H)');

subplot(2,2,4),plot(BUF,Beta), grid, xlabel('Buffer Requirement (MB)'),
ylabel('Beta(P=0.850 sec, B=35840 Bytes)'),title('Beta vs Buffer (Disk 81H)');

figure(2); clf;
plot(Wseek),grid, ylabel('Worst Seek Time (ms)'), xlabel('Number of Seeks'),
title('Roundup Worst-case CSCAN Seek Function (Disk 81H)');

% end.

```

Figure B.1: Graphs generated by MATLAB[®] showing the performance of the FSCAN algorithm with chosen scan period and block size.

Figure B.2: A plotting generated by MATLAB® showing the round-up worst CSCAN seek time function.