BLUEPRINT FOR A HARD REAL-TIME FILE SYSTEM

Raymond M. K. Cheng[†], Mark Greenstreet[**], Donald W. Gillies[†]

[†]Department of Electrical Engineering
University of British Columbia
Vancouver, BC, Canada V6T 1Z4
{raymondc,gillies}@ee.ubc.ca

[**]Department of Computer Science
University of British Columbia
Vancouver, BC, Canada V6T 1Z4
mrg@cs.ubc.ca

August 31, 1995

Abstract

We examine the problem of scheduling disk requests in a personal hard real-time read/write file system. For arbitrary aperiodic requests, we model the problem of moving the disk arm 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. We propose an optimal dynamic-programming algorithm to solve this problem, and we also show that any optimal algorithm for this simplified problem 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.

Keywords: *Disk Scheduling*, *Hard Real-Time*, *Travelling Salesperson*, *Worst-Case Analysis*, *File Systems*.

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 video applications 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. We believe the trends in personal computers and in disk drives are more compelling than trends in large centralized servers.

In this paper 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.

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 8.6 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 stream should be characterized by its maximum transfer rate, size and start-up latency. In addition, the file system must be real-time. To do this, we believe it is necessary to store data non-contiguously on the disk drive.

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

Two approaches to disk scheduling are investigated: optimal scheduling and heuristic scheduling with substantial memory buffering. Our analysis of optimal scheduling is based on a one dimensional travelling salesperson problem. We demonstrate that there are workloads that could cause an optimal policy to intrinsically thrash. These results motivate a heuristic approach to the problem known as 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.

1.2 Related Work

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. In fact, SSTF may lead to starvation for the innermost or outermost disk tracks and SCAN has no notion of deadlines for scheduling purpose.

SCAN-EDF is a strategy for real-time disk scheduling [Redd93] 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. The efficiency of SCAN-EDF greatly depends on the fraction of disk requests that have the same deadline and are served with the SCAN 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. 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.

Another approach is the greedy strategy [Abbo84, Vin94]. This strategy tries to minimize both the seek and rotational latency by finding an optimal sequence for retrieving data blocks on disk. 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 problem is solved with a near-optimal greedy disk scheduling algorithm [Vin94]. The greedy strategy is not a dynamic scheduling algorithm as the set of requests are required to be known a priori. Some other policies which take rotational latency into account have been proposed by [Jacob91], [Ng91] and [Selt90]. 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 is used our analysis. We also develop an accurate disk model in order to capture the details of different overhead components of a modern disk.

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 deterministically guaranteed result. In addition, the paper does not provide an explanation of their bound's calculation. This paper, we consider heterogeneous streams with arbitrary data rates. A guaranteed real-time data delivery admission test is also provided.

In the next section, the disk model used in our real-time file system is defined. An analysis of a simplified disk scheduling problem which shows the drawbacks of trying to optimally service requests for randomly stored data is presented in Section 3. Section 4 presents a worst-case analysis of CSCAN and then the FSCAN approach to the real-time disk scheduling. An evaluation of FSCAN is discussed in Section 5. Section 6 summarizes our work and suggests further research.

2. Disk Model

In this section, 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.

2.1 A Modern Disk

A magnetic disk consists a collection of double-sided magnetically coated *platters* which rpm on a common spindle, typically rotate at 3600, 4000, 5400, or 7200. 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 2.1 shows the mechanical components of a hard disk.

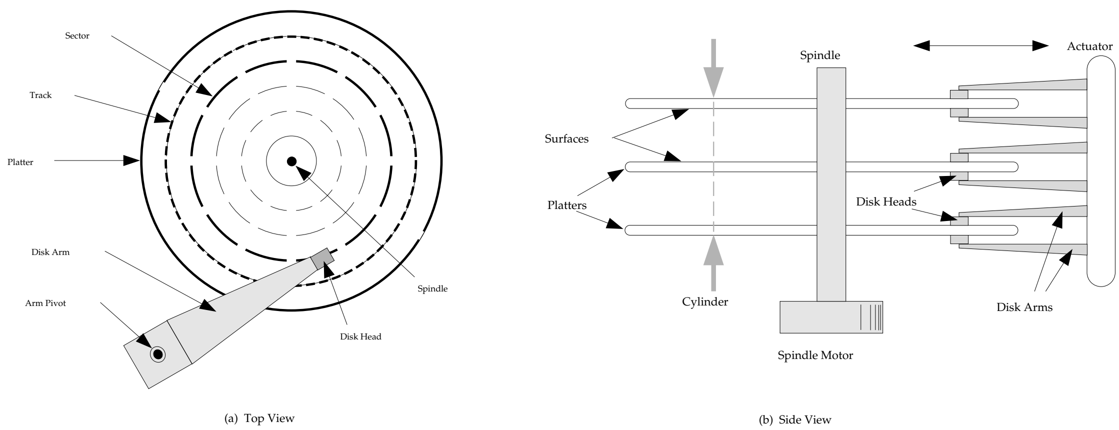

Figure 2.1: Hard disk mechanical components.

A set of moveable *disk arms* attached to the same rotation 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 *settle 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 settle 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 settle 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 logical number of the sectors of 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” disks 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 paper 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.

2.2 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 $s(d) = a_1 + a_2 d$ 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 combines 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 more accurate seek function is defined as

$$s(d) = \begin{cases} a_1 + a_2\sqrt{d} & 0 < d < D \\ a_3 + a_4 d & D \le d \le N. \end{cases} \tag{2.1}$$

Figure 2.2 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 .

In this paper, we assume a non-decreasing concave seek time function $s(x)$, i.e. $s'(x+\delta) \le s'(x)$ and $s'(x) \ge 0$ for all x and $\delta \ge 0$, as shown in Figure 2.3. This seek-time model contains the three previous models as special cases. Based on this seek-time model, we will show that the worst-case seek overhead can be measured empirically for a large class of seek functions (see Section 4.1 and Appendix).

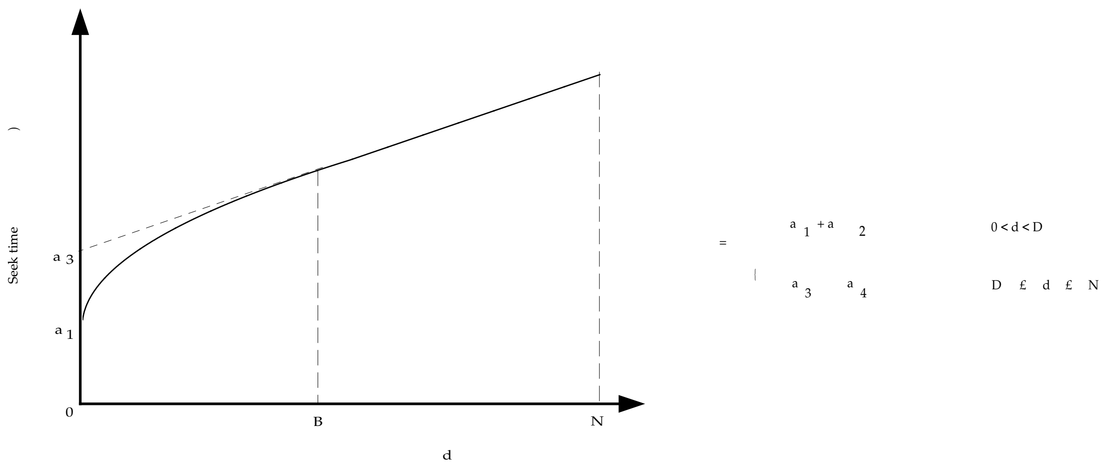

Figure 2.2: Seek time function of a typical hard disk.

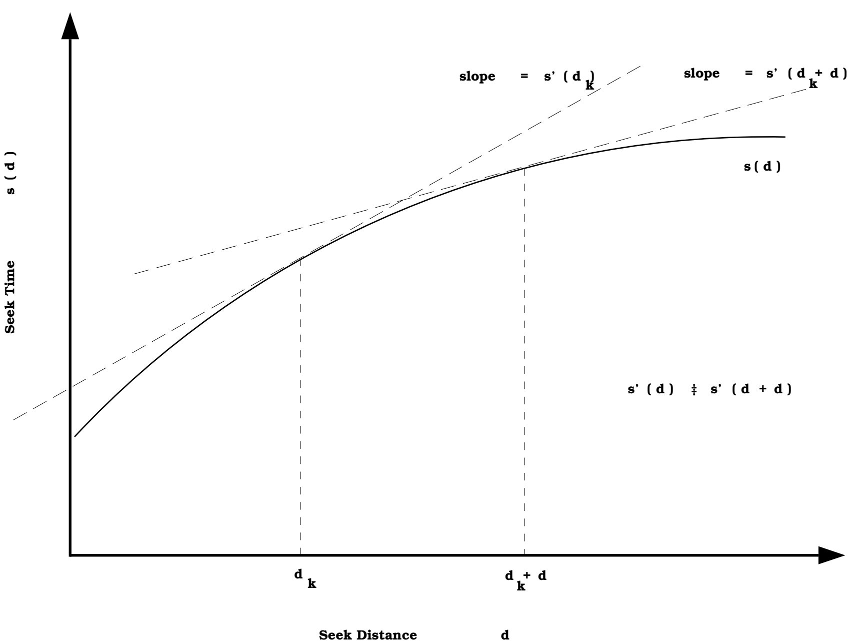

Figure 2.3: A non-decreasing concave seek function model.

3. Optimal Seek Algorithm

In this section a simplified version of the disk scheduling problem and its scheduling algorithm are presented. For arbitrary aperiodic requests the problem of moving the disk arm is modelled as a travelling salesperson problem with deadlines on the cities. Since this is obviously an NP-hard problem we consider a simpler problem of scheduling a travelling salesperson on a one dimensional line, where travel time is proportional to distance and where time spent at each city is zero. Thus we assume that blocks are read in zero time. Even in this extremely simplified framework, thrashing may occur, and finding a trivial schedule requires a non-trivial dynamic programming algorithm. These results motivate the heuristic scheduling method that is presented in the remainder of the paper.

The assumption of zero read-time ignores rotational latency. Although rotational delays may be comparable to or greater than seek times, it is common to absorb them in the block read time because many disks controllers do not provide rotational positioning information. We also assume unbounded buffer capacity: a block may be read arbitrarily early. For simplicity, this presentation uses a model where seek time is proportional to distance.

In the zero read time model, it is sufficient that the heads pass over the required cylinders before the earliest deadline for any block in that cylinder. Let N denote the number of cylinders, where cylinder 0 is the outermost cylinder and cylinder N–1 is the innermost. Since we assume a purely linear seek function and normalized seek times, the time needed to move from cylinder i to cylinder j is $\|i-j\|$. If the heads are at cylinder 0 or N–1 at the beginning of the schedule, then the schedule is feasible if and only if each cylinder is visited on time with a single sweep across the disk. However it is typically necessary to construct a disk schedule in stages, since the schedule may contain thousands or millions of block requests. This is because the schedule or all the disk blocks may not fit into main memory. A second reason to construct a schedule in stages is because a fully constructed schedule may be wasted. Therefore, at some stage we may find that the disk is positioned at the middle of the disk platters, at cylinder $N/2$.

Consider the example in Figure 3.1. Assume that N is even and the head is initially at cylinder $N/2$. For $-N/2 \leq k < N/2$, the series of disk request deadlines require that cylinder $N/2+ k$ be visited by time $2*k^2 - k$. It is easy to see this schedule meets all the deadlines if and only if the heads first move from cylinder $N/2$ to $N/2 + 1$ and then to cylinders $N/2–1$, $N/2+2$, $N/2–2$, … . As this demonstrates, even with zero read times, it may not be possible to schedule with a single sweep across the disk, and thrashing may occur.

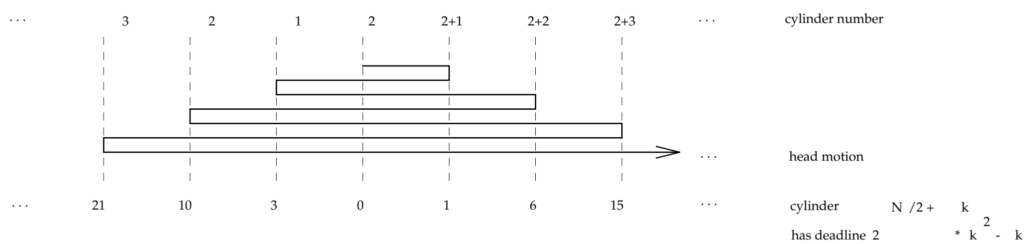

Figure 3.1: Example of a disk schedule with thrashing may occur.

The zero read time scheduling problem can be solved by dynamic programming. Let the left-most cylinder be the outer cylinder and the right-most be the inner cylinder. We observe that the only important characteristic of a schedule is the left-most and the right-most cylinder visited. When these two cylinders are known, all cylinders between these two have been serviced if the algorithm is optimal. The algorithm determines the entries of an array A. Let $A(i, j)$ be the earliest time that the heads can be at left-most cylinder i given that they have previously been at right-most cylinder j and thus all of the deadlines for cylinders $i…j$ have been satisfied. Likewise, $A(j, i)$ is the earliest time at which the heads can be at rightmost cylinder j given that they have previously been at leftmost cylinder i and that all of the deadlines for cylinders $i…j$ have been satisfied. The algorithm for computing $A(i, j)$ is given in Figure 3.2.

```
/* Assume: n cylinders (numbered 0 ... n−1)
 *              heads start at cylinder m.
 * Recall that A(i, j) gives the earliest time the heads can be
 * at cylinder i given that they have already been to cylinder j.
 */

/ * initialize row m of the tableau */
for(i = 0; i < n; i++)
   A(m, i) = Infinity;
A(m, m) = 0;

/* fill in the rest of the tableau */
for(x = 1; x < n; x++) {     /* x = |m−i| + |m−j| */
   /*  loop invariant:
    *  A(i, j) has been calculated for |m−i| + |m−j| < x
    */
   for(y = 0; y < x; y++) { /* y = |m−i| */
      j = m+y;  /* j >= m */
      i = j−x;  /* i < m, |m−i| + |m−j| = (m−i) + (j−m) = x */
      if((i >= 0) && (j < n)) {
         A(i, j) = min(A(i+1, j) + 1, A(j, i+1) + (j−i));
         if(A(i, j) > deadline(i)) A(i, j) = Infinity;
      }
      j = m−y;  /* j <= m */
      i = j+x;  /* i > m, |m−i| + |m−j| = (i−m) + (m−j) = x */
      if((i < n) && (j >= 0)) {
         A(i, j) = min(A(i−1, j) + 1, A(j, i−1) + (i−j);
         if(A(i, j) > deadline(i)) A(i, j) = Infinity;
      }
   }
}
```

Figure 3.2: Algorithm for computing *A*(*i*, *j*).

When the algorithm terminates, if $A(0, N{-}1)$ or $A(N{-}1, 0)$ is less than $+\infty$ then the schedule is feasible. The entries of A can be determined by evaluating successive diagonals, and entries are only needed for cylinders with deadlines. Let M be the number of deadlines input to the algorithm. This algorithm uses $O(M^2)$ time and $O(M^2)$ space.

In a more realistic model the time to read a block would be considered. If different blocks may have different read times (e.g. due to varying block sizes), then the disk scheduling problem can be shown to be equivalent to the partition problem which is NP-complete. This reduction does not apply in the case where each block takes the same amount of time to be read, as the reduction amounts to a unary encoding of the partition problem. We believe that this problem is NP-complete but do not have a proof of this. These observations motivate the heuristic approach to disk scheduling presented in the next section.

4. A FSCAN Heuristic for Periodic Requests

In the previous section the optimal seek algorithm for an arbitrary series of requests was shown to cause disk thrashing. It can be shown that the disk transfer rate in the worst case is very low if sequential requests are near opposing ends of the disk. In this section, 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 at the core of a real-time file system. 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. With this seek model, we perform a worst-case analysis of the CSCAN seek. Then we 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.

4.1 Worst-case analysis of CSCAN disk algorithm

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 the CSCAN algorithm. Using (2.1) as the seek time function model, the time to perform a series of n seeks across the disk in one direction can be formulated as the problem:

$$\text{(4.1)} \qquad \begin{array}{ll} \text{maximize} & S(\mathbf{d}) = \sum_{i=1}^{n} s(d_i) \\ \text{subject to} & \sum_{i=1}^{n} d_i \le N \end{array}$$

where $\mathbf{d} = (d_1, d_2, \ldots, d_n)$. Notice that the constraint of (4.1) is $\sum_{i=1}^{n} d_i \le N$. However in the worst case, since $s(\mathbf{d})$ is a non-decreasing function, we can assume $\sum_{i=1}^{n} d_i = N$. To solve (4.1), we can use the following theorem.

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

Proof. The proof is contained in the Appendix. ❑

Note that for each CSCAN sweep, the disk arm can only have at most $\lfloor N/D \rfloor$ linear seeks (with seek distance $D \le d \le N$). 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 (2.1) 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

$$W_{seek}(n) = \begin{cases} n\left(a_3 + a_4\left(N/n\right)\right) & n \le N/D \\ n\left(a_1 + a_2\sqrt{N/n}\right) & n > N/D \end{cases}$$

$$\text{(4.2)} \qquad = \begin{cases} a_3 n + a_4 N & n \le N/D \\ a_1 n + a_2\sqrt{nN} & n > N/D \end{cases}.$$

In fact, Theorem 1 can be applied to any disk which has a non-decreasing continuous concave seek function. We have found only one disk that does not fit this model [Henn90]. For this disk, the seek-time function can be rounded upwards. 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 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.

4.2 Worst-case analysis of the FSCAN algorithm

The file system services a stream system of n continuous data stream requests $\mathbf{S} = \{S_1, S_2, ..., S_n\}$. 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 given by

$$m_i = \left\lceil \frac{P}{T_i} \right\rceil = \left\lceil \frac{PR_i}{B} \right\rceil. \tag{4.3}$$

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 only on the periodic requests in this paper. The scan period can 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.

The schedule in Figure 4.1(a) shows an EDF periodic schedule of a given set of streams. The FSCAN policy simulates the EDF algorithm by following the schedule in Figure 4.1(b). For each stream the number of data blocks to be served in one period can be determined by (4.3). 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 4.1(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.

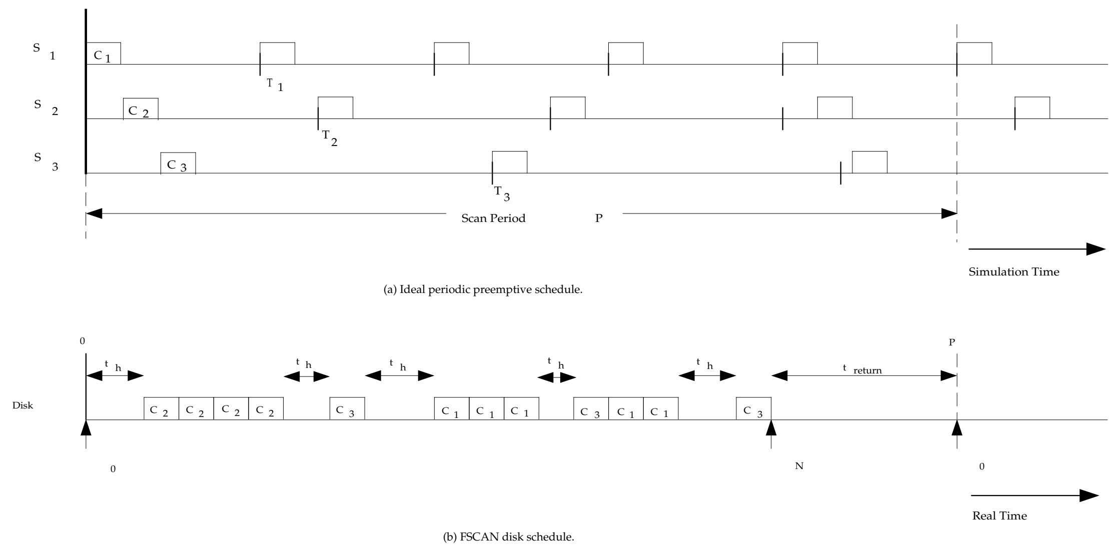

Figure 4.1: An illustration of FSCAN(*P*,*B*) algorithm.

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 (4.3). 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 4.2 illustrates this fact. At a particular moment, the disk head is located at cylinder x and is sweeping towards the x+1 direction, while an aperiodic request targeted at cylinder x–1 arrives. As we can see from the Figure 4.2(a) and 4.2(b), 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 worst-case latency guaranteed. Later, we will see that the full-return seek costs very little in overall throughput. We believe this is a trade-off most file systems would want to make.

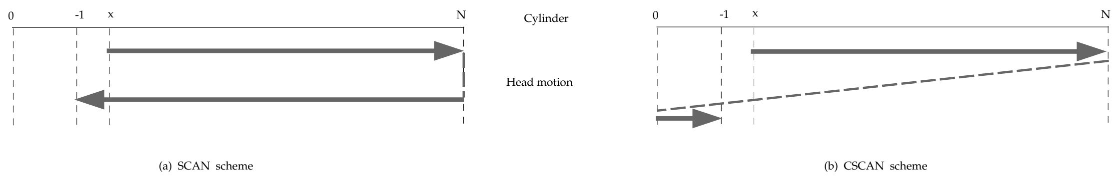

Figure 4.2: Expected waiting time to serve aperiodic requests with different scheduling scheme.

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 (4.2). The worst case rotational latency is denoted by t_{rot}. Other physical overheads are read/write overhead t_{rw}, time for track switch[1] 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 (4.3) is equal to nC or $n(B/DTR)$. Thus, the overhead function of FSCAN for a given data stream set $\boldsymbol{S}$ is

$$H(P,B) = W_{seek}(m) + m \cdot (t_{rot} + t_{rw} + t_{ts}) + t_{return} + n \cdot \frac{B}{DTR}. \tag{4.4}$$

As we can see from (4.4), 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 (4.3):

$$\begin{aligned} m = \sum_{i=1}^{n} m_i &= \sum_{i=1}^{n} \left\lceil \frac{P \cdot R_i}{B} \right\rceil \\ &\leq \sum_{i=1}^{n} \left(\frac{P \cdot R_i}{B} \right) + n \\ &= \frac{P \cdot R}{B} + n. \end{aligned} \tag{4.5}$$

---

[1] In the worst case track switches occur rather than head switches because a head switch typically takes only one third to one half 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 $m = (P \cdot R)/B + n$.

4.3 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:

$$BUF(P,B) = \sum_{i=1}^{n} C \cdot \left\lceil \frac{2P}{T_i} \right\rceil \cdot DTR = C \cdot DTR \cdot \sum_{i=1}^{n} \left\lceil \frac{2PR_i}{B} \right\rceil$$

$$\leq \left(\frac{B}{DTR} \right) \cdot DTR \cdot \left(\frac{2PR}{B} + n \right) = 2PR + nB \;. \qquad (4.6)$$

4.4 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 (4.4) represents the worst-case overhead of FSCAN in one scan period, the percentage a period that can be spent transferring data at the maximum transfer rate of the disk 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

$$R \leq \frac{P - H(P,B)}{P} \cdot DTR \;. \qquad (4.7)$$

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 (4.5). Since $m > N/D$ in the worst case, thus $W_{seek}(m) = a_1 m + a_2 \sqrt{mN}$ from (4.2).

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 (2.1).

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

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

For example, if the block size is equal to the size of a track, the worst-case rotational latency will 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 the 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

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

However, since the number of physical sectors dedicated for sparing is relatively small in modern disk drives[2], we will use (4.8) as the rotational latency in the rest of our analysis.

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

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

---

[2] According to the data given in [Ruem94], only 0.88% and 1.3% of physical sectors are reserved for sparing regions in HP C2200A and HP 97560 disk drives respectively.

Using the results from (4.7) to (4.9), for given P and B, the maximum throughput rate R_{max} that the file system can support is given by

$$R_{max} = \frac{P - H(P,B)}{P} \cdot DTR$$

$$= \frac{DTR}{P}\left(P - \left(a_1 n + a_2\sqrt{mN} + m(t_{rot} + t_{rw} + t_{ts}) + (a_3 + a_4 N) + n \cdot B/DTR\right)\right). \quad (4.10)$$

Substituting $m = (P \cdot R_{max})/B + n$ into (4.10), we can find R_{max} by solving a quadratic equation. The equation to solve for R_{max} is given below:

$$\begin{aligned} &R_{\max}\left(1 + \frac{DTR(a_1 + t_{rot} + t_{rw} + t_{ts})}{B}\right) + \sqrt{R_{\max}}\left(a_2 DTR\sqrt{\frac{N}{BP}}\right) \\ &+ \frac{DTR}{P}\left(n(a_1 + t_{rot} + t_{rw} + t_{ts}) + a_2\sqrt{nN} + a_3 + a_4 N + \frac{nB}{DTR} - P\right) = 0. \end{aligned} \quad (4.11)$$

Lemma 3 Given a stream system $\boldsymbol{S} = \{S_1, S_2, ..., S_n\}$ and the system parameters P and B, $\boldsymbol{S}$ is schedulable if the total requested throughput rate $R = \sum_{i=1}^{n} R_i$ is less than the solution for R_{max} in (4.11).

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. ❑

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 percentage of throughput the file system supports compared to the maximum disk throughput 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 $\boldsymbol{S}$ 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 $\boldsymbol{S}$ 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. ❑

5. Performance Analysis and Evaluation

In this section we evaluate the FSCAN algorithm with real disk parameters. For this purpose the file system is configured with the disk parameters listed in Table 5.1. We first choose the HP 97560 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	HP 97560 [Ruem94]	HP C2200A [Ruem94]	HP C2247 [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 (worst)	1.3 (worst)	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)	$3.24 + 0.400\sqrt{d}$ $d<D$ $8.00 + 0.008d$ $D \leq d \leq N$ $D=383$	$3.45 + 0.597\sqrt{d}$ $d<D$ $10.8 + 0.012d$ $D \leq d \leq N$ $D=616$	$3.81 + 0.330\sqrt{d}$ $d<D$ $7.75 + 0.0059d$ $D \leq d \leq N$ $D=300$	$4.6 + 0.87\sqrt{d}$ $d<D$ $11.3 + 0.028d$ $D \leq d \leq N$ $D=239$

Table 5.1: 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 HP 97560 disk, Figures 5.1 to 5.3 show different surface plots of $\beta(P,B)$. The number of data streams is 6 but is varied later. Figure 5.1 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 ($72 \times 512\text{B} \approx 37$ 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 (equation (4.8)). 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 HP 97560.

Figure 5.2 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 (4.8). Combining Figure 5.1 and 5.2 gives Figure 5.3, 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 0.4 with block size equal to 2 tracks, there must be an 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.

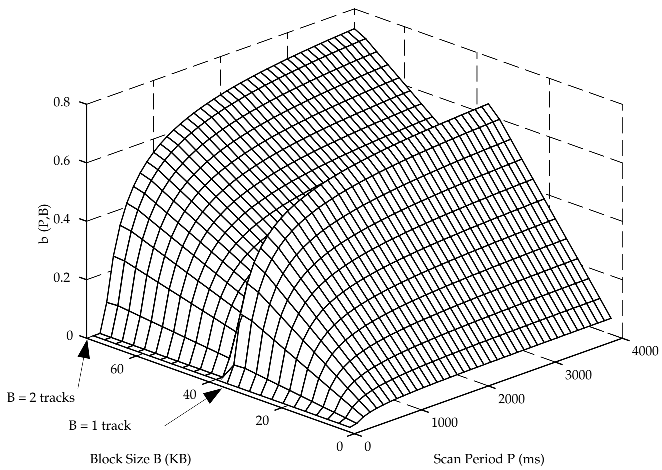

Figure 5.1: Effective schedulability factor $\beta(P,B)$ of *FSCAN*(P,B) with HP 97560.

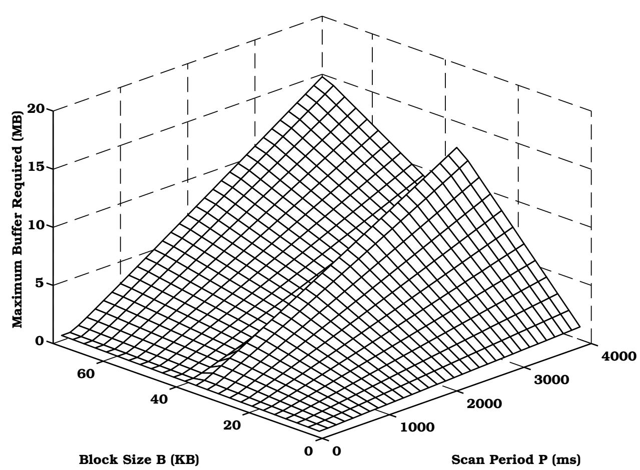

Figure 5.2: Maximum buffer requirement in *FSCAN*(P,B) with HP 97560.

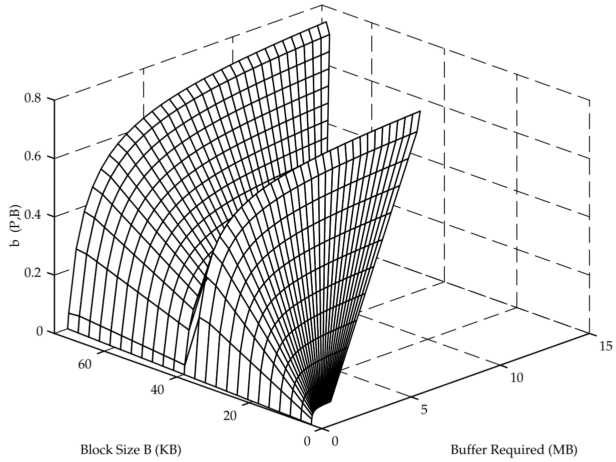

Figure 5.3: Maximum buffer required for $\beta(P,B)$ with HP 97560.

The breakdown of different overhead components for FSCAN(P,B) is shown in Figures 5.4 and 5.5. With block size equal to half of a track (Figure 5.4), rotational latency is the most dominant overhead when P 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 (4.8). Hence, as shown in Figure 5.5, rotational latency becomes the least significant component among all the overheads and the seek latency becomes the most dominant factor.

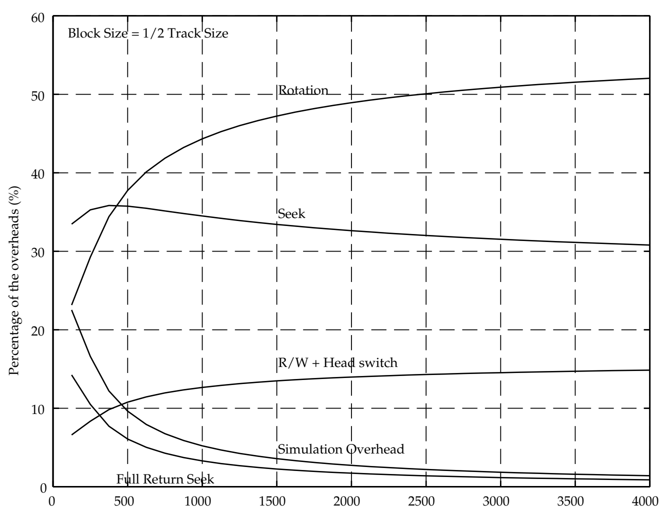

Figure 5.4: Overhead components when block size is one half of a track.

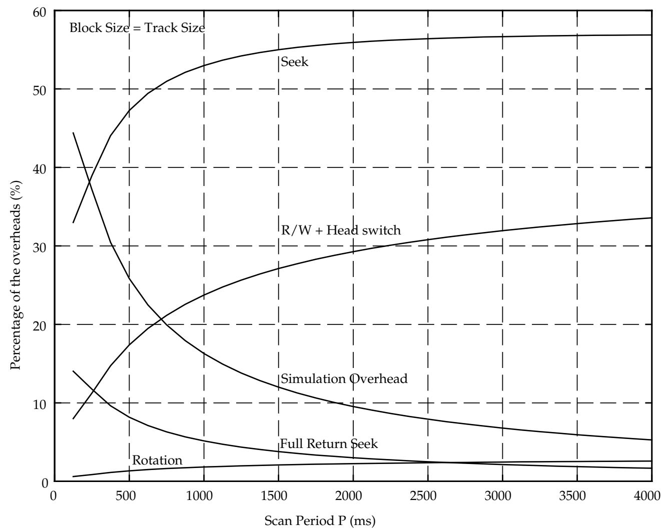

Figure 5.5: Overhead components when block size is a whole track.

Figure 5.6 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 section, we know that schedulability depends substantially on the overhead function $H(P,B)$ given by (4.9), 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 (4.5), we obtain a good understanding of $\beta(P,B)$ with a fixed value of n. Similarly, the buffer requirement for FSCAN varies slightly with different number of data streams, as shown in Figure 6.7. That is the reason for choosing a fixed n in the previous plots.

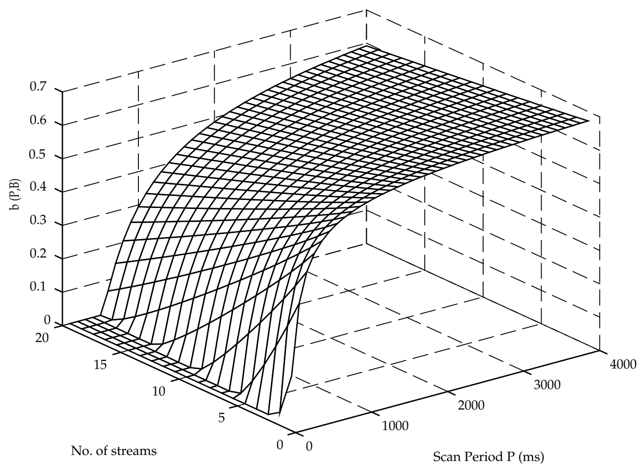

Figure 5.6: β(*P*,*B*) with differing numbers of data streams when block size equals to one track.

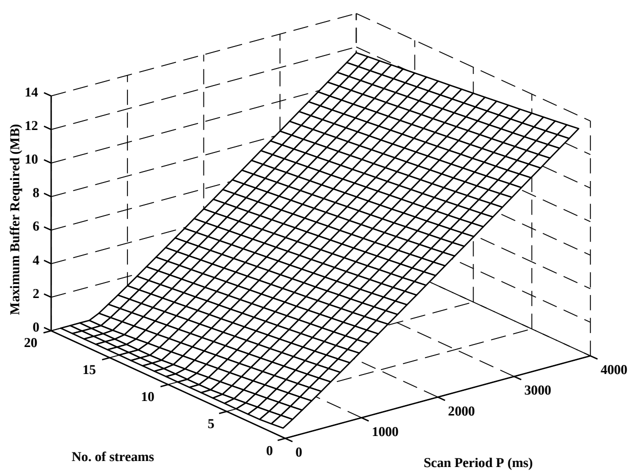

Figure 5.7: Maximum buffer requirement in *FSCAN*(*P*,*B*) with differing numbers of data streams when block size equals to one track.

We now examine the behaviour of FSCAN with different disk drives given in Table 5.1. We have made worst-case assumptions with the zoned disk HP C2247: the number of sectors per track (and hence the *DTR*) is assumed to be the lowest possible. Figure 5.8 plots $\beta(P,B)$ against P with B equal to 1 track and 6 data streams. The maximum buffer required for different configurations is shown in Figure 5.9. 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. HP 97560 has a relatively fast seek ability and therefore it outperforms the other disks. Although the HP C2247 has the fastest seek ability it does not perform well because of its large cylinder size and a worst-case zoning assumption used.

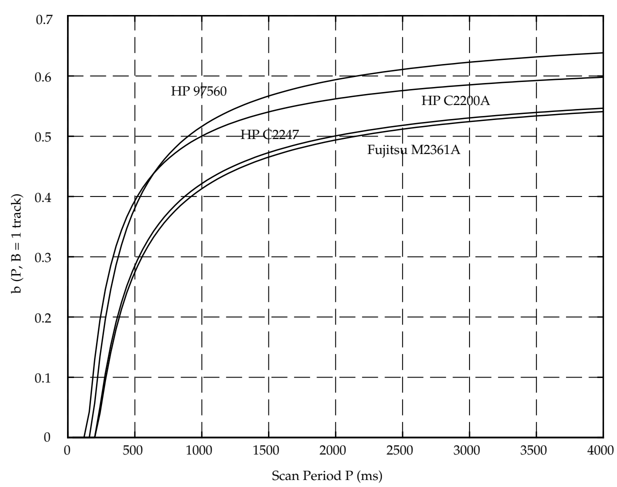

Figure 5.8: β(*P*, 1 track) for different disks.

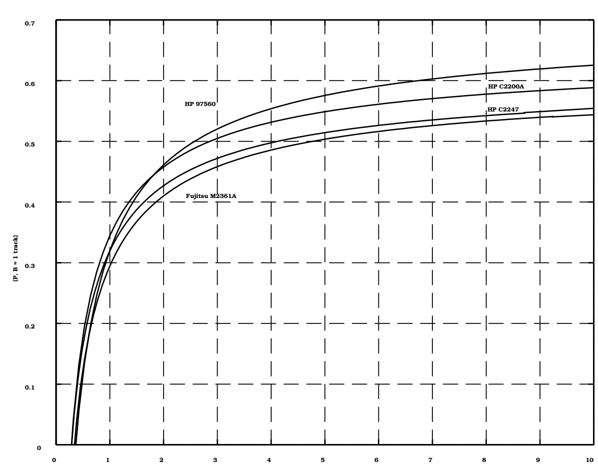

Figure 5.9: Maximum buffer required for β(*P*,1 track) for different disks.

As an illustration for how our real-time file systems works in real applications, if we use HP 97560 in our file system and we transfer 1 track at a time and the scan period P is 1000 ms, then the effective schedulability is $\beta(1000,1) = 0.52$ from Figure 5.8. The corresponding buffer requirement is approximately 3 MB according to Figure 5.9. Since DTR is 2.5 MB/s (or 20 Mbps) for the HP 97560, the file system can support $\beta(1000,1) \cdot DTR = 10.4$ Mbps of throughput. In Section 1, we argue that a useful hard real-time system should support approximately 8.6 Mbps of concurrent data transfer from 6 separate streams. Thus, the FSCAN algorithm meets the throughput goals of this paper.

6. Conclusion

We have examined the problem of scheduling disk requests in a stand-alone personal hard real-time read/write file system. We showed that it can be very inefficient to optimally service disk requests for data laid down randomly on the disk. Our analysis shows there are some workloads that would cause an optimal policy to intrinsically thrash. These results motivate a heuristic approach to the problem. We propose an integrated fixed-period SCAN algorithm for scheduling hard real-time data streams. The approach is parameterized by disk block size and memory buffer size, and trades 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. In fact, our analysis allows us to build a hard real-time file system without having to obtain proprietary and sensitive information from the disk manufacturer. Results show that the FSCAN policy can transfer data at 40-70% of the maximum data throughput for modern disk drives, depending on the file system configuration.

Further investigations might use two or more block sizes to reduce the problem of disk fragmentation. The use of an aperiodic server to serve non-real-time requests is also a logical extension. We are developing software to automatically test and calibrate the FSCAN algorithm using the techniques suggested by Theorem 1.

7. Acknowledgements

We would like to thank Kendra Cooper for her comments on an earlier draft of this paper.

References

[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.

[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.

[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 IEEE Real-Time Systems Symposium*, Dec. 1991, pp. 129-139.

[Henn90] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, San Mateo: Morgan Kaufmann Publishers, 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.

[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.

[Pere88] A. L. Peressini, F. E. Sullivan, J. J. Uhl, The Mathematics of Non-Linear Programming, New York: Springer Verlag, 1988.

[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.

[Ruem94] C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *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.

[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.

[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 RAID™ -- 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

Theorem: For any function $F(\mathbf{x}) = \sum_{i=1}^{n} f(x_i)$ where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $\sum_{i=1}^{n} x_i = N$, 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 $x_1, x_2, \ldots, x_n$ are evenly distributed along a number line from 0 to N (i.e. when $x_1 = x_2 = \ldots = x_n = N/n$). Thus $\max(F(\mathbf{x})) = n \cdot f(N/n)$.

Proof: Let $f(x)$ be divided into L maximal differentiable regions with domains $[b_0, b_1]$, $[b_1, b_2]$, …, $[b_{L-1}, b_L]$. We divide $f(x)$ into $g_1(x), g_2(x), \ldots, g_L(x)$ 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 also 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 A.1 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 $x \notin [b_{k-1}, b_k]$.

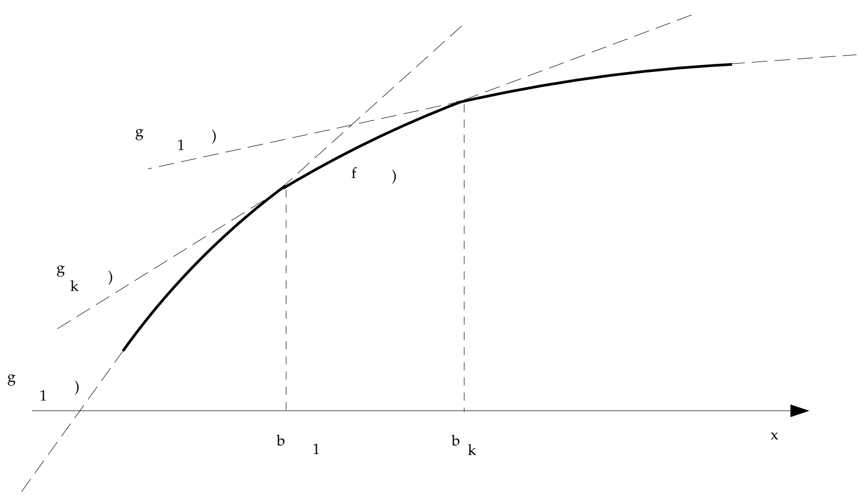

Figure A.1: A section of *f*(*x*) in the domain [b_{k-1}, b_k].

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 $1 \le j \le n-1$, 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

$$g_k'(x_j) = g_k'(N - \sum_{i=1}^{n-1} x_i). \tag{A.1}$$

To solve (A.1), we consider two cases: (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 $g_k'(x) \ge 0$, hence, $g_k'(x+\delta) = g_k'(x)$ only if $\delta = 0$. Thus, we get the critical point $x_j = x_n$ from (A.1). Since $g_k(x)$ is a concave function, therefore the critical point $x_j = x_n$ represents a global maxima $G_k(\mathbf{x})$ [Pere88]. Hence, $x_1 = x_2 = ... = x_n = N/n$ 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 $\mathbf{x} = (x_1, x_2, ..., x_n)$ obeying the constraint $\sum_{i=1}^{n} x_i = N$. For this reason, there is no unique solution to (A.1). Thus, $x_1 = x_2 = ... = x_n = N/n$ is a non-unique solution to (A.1).

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

$$\sum_{i=1}^{n} f(x_i) \le \sum_{i=1}^{n} g_k(x_i)$$

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

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 $x_1 = x_2 = ... = x_n = N/n$. ❑