SUPPORT FOR CONTINUOUS MEDIA IN FILE SERVERS

by

D. James Gemmell

M.Math., University of Waterloo, 1990

B.Sc., Simon Fraser University, 1988

THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in the school

of

COMPUTING SCIENCE

SIMON FRASER UNIVERSITY

April 1995

APPROVAL

Name: D. James Gemmell

Degree: Doctor of Philosophy

Title of thesis: Support For Continuous Media In File Servers

Examining Committee: Chair: Dr. Stella Atkins

Dr. Jiawei Han, Senior Supervisor

Dr. Tiko Kameda, Supervisor

Dr. John Dill, Supervisor

Dr. Ze-Nian Li, S.F.U. Examiner

Dr. Ling Liu, External Examiner

Date Approved: April 13, 1995

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

Title of Thesis/Project/Extended Essay

Support for Continuous Media In File Servers.

Author: _____

(signature)

David James Gemmell

(name)

April 18, 1995

(date)

Abstract

Continuous media (CM), such as audio and video, fundamentally differ from traditional text and numeric data in that they have large transfer rate and storage space requirements, and real time deadlines must be met during their storage and retrieval. This thesis covers the issues involved in designing file servers that support continuous media. A rigorous model of the real time requirements is presented, and lower bounds are demonstrated for parameters such as buffer space. The sorting set disk scheduling algorithm is proposed, which balances disk latency reduction against successive read latencies. The sorting set approach has the advantage of being a generalization of previous approaches (including SCAN and round-robin), as well as yielding improved performance in certain cases. Conventional approaches to file system implementation are analysed for suitability in CM file servers, and new, or hybrid, solutions are proposed.

DEDICATION

To Isaac Peter Payan

Save a front row seat for me!

ACKNOWLEDGEMENTS

I have been privileged to have not just supervisors, but tremendous encouragers during my studies. Dr. Jiawei Han, in supervising my work often seemed to have more faith in me than I had in myself. The same could be said for Dr. Stavros Christodoulakis, my supervisor for my Master's degree, who continues to encourage me and offer good advice years afterwards. I must also acknowledge the prophetic encouragement of my father, David Gemmell – now my play has become successful work, just as you said!

During this study, I have been supported by NSERC, the BC Science council, and various sources at SFU. MPR Teltech Ltd. has graciously given me equipment to work with, and excellent people to confer with, among whom Richard Beaton especially stands out. However, when it comes to support, no-one has paid the price like my wife, Elizabeth.

Where do ideas come from? This thesis is not a product of my ability or hard work. "Not by might, nor by power, but by my Spirit, says the Lord".

Contents

Figures

Tables

1. INTRODUCTION

1.1 Motivation

The communication interface between man and computer is evolving at a rapid pace. For input, punch cards have been superseded by keyboards, which are now finding competition in mice, touch screens, and pen-based systems. In the area of computer output, text interfaces are being replaced with multimedia interfaces that include graphics, images, animations, audio and video. The step already taken from text-only systems to graphical systems was a significant one, requiring more powerful hardware, efficient algorithms for handling new types of data, and new paradigms for user interfaces. The step forward into audio and video will be at least as significant.

Audio and video belong to a class of data known as delay-sensitive, or continuous. The term delay-sensitive is used because they are sensitive to delays in presentation to the user. In order for the presentation to be acceptable, real time deadlines must be met. The term continuous is used because the deadlines occur in a continuous, repetitious fashion. We will use the term ***continuous media*** (CM) in this thesis.

Historically, continuous media has not been cost-effective in the digital domain because of its large bandwidth requirements for transmission, and large storage space requirements. However, recent advances in both transmission and storage technology have made digital continuous media feasible. In particular, the advent of optical disk storage technology and fibre optic transmission technology has encouraged attention on digital continuous media. This is not to say that these technologies are required for continuous media. In fact, magnetic disk storage and wire transmission are now advanced enough to support it also, as we shall see later.

Currently, continuous media is beginning to find its way into common operating systems and hardware platforms. Voice mail has been implemented, and the inclusion of video within a window on a graphics screen is popular. Generally speaking, the support of continuous media is for a single user accessing a single file in a local context. In this thesis we consider the more difficult problem of simultaneous multiple file access, such as would be required of a file server.

We assume that a multimedia server is connected via a network to ***display sites*** belonging to clients. Clients can request the real-time retrieval of a multimedia object for playback at their display sites. The retrieval can be interactive, in the sense that clients can stop, pause, resume, and even (in some cases) record and edit the media. Thus, the multimedia server operates much like a remote VCR, or audio tape deck. Prospective applications range from information kiosks connected via LAN's, to nationally supported "video-on-demand" provided to households by cable companies.

1.2 The Nature of Continuous Media

The design of multimedia servers differ significantly from the traditional text/numeric storage servers due to two fundamental characteristics of digital audio and video:

- ***Large data transfer rate and storage space requirement***: Digital video and audio playback consumes data at a very high rate (see Table 1). Thus, a multimedia server must provide efficient mechanisms for storing, retrieving and manipulating data in large quantities at high speeds.
- ***Real time storage and retrieval:*** Continuous media (such as audio and video) consist of a sequence of media quanta (such as video frames or audio samples) which convey meaning only when presented continuously in time. This is in contrast to a textual object, for which

spatial continuity is sufficient. It is also possible that a multimedia presentation may included the timed display of non-CM data. For example, images may be displayed at specified times to create a "slide-show". Furthermore, several media objects may need to be combined (e.g., super-imposing one image on another), or displayed synchronously.

Media	Bandwidth
Voice-quality audio[1]	8 KB/sec
MPEG-encoded audio[2]	48 KB/sec
CD-quality audio[3]	172 KB/sec
MPEG-2-encoded video[4]	512 KB/sec
NTSC-quality video[5]	27,000 KB/sec
HDTV-quality video[6]	81,000 KB/sec

Table 1: Bandwidth requirements for continuous media

The real time requirements for the retrieval (or storage) of continuous media differ significantly from those of conventional "real time" data bases. For a standard real time data base, a deadline is usually set for the retrieval of the first item satisfying a query, or less commonly the last item (i.e., the whole set of data) satisfying the query. In contrast, continuous media demands that a deadlines be met for each retrieved item. There is no deadline for the first item, or the last item, but the time between items is crucial.

[1] Monophonic; 8 bit samples at 8 kHz.

[2] 48 KB/sec is highest quality – very close to CD quality. MPEG allows for rates as low as 4 KB/sec for low quality applications.

[3] Stereo (2 channel); 16 bit samples at 44.1 kHz.

[4] 640 x 480 pixels per frame; 30 frames per second; 24 bit pixels.

[5] 640 x 480 pixels per frame; 30 frames per second; 24 bit pixels.

[6] 1280 x 720 pixels per frame; 30 frames per second; 24 bit pixels.

1.2.1 Compression and Irregular Media

Continuous media requires large amounts of storage space and transmission bandwidth. In order to be cost-effective, various approaches are taken to reduce these requirements. A natural choice for reducing both storage space and transmission bandwidth is data compression. Compression can either be lossless or lossy. The data resulting from compression and decompression with a lossless compression scheme is identical to the original uncompressed data. With a lossy algorithm, it is not identical.

Since some data is lost, lossy schemes can achieve higher compression ratios than lossless schemes. In addition, some lossy schemes can guarantee a constant compression ratio. In contrast, lossless schemes cannot guarantee any particular compression ratio, and in some cases can actually lead to an increase in data. Audio and video have constant bandwidth requirements, so it is attractive to utilize a lossy compression which achieves a constant bandwidth for its output for transmission purposes. Constant ratio, lossy compression schemes are becoming the standard in broadcasting, telecommunications, and for consumer audio products such as the mini-disk and digital compact cassette (DCC). The most popular lossy schemes for digital audio are those which apply an understanding of human hearing to the compression [48]. These schemes attempt to ensure that the lost data is virtually inaudible. Similarly, compression schemes for video can attempt to exploit an understanding of human vision to yield subjective improvements.

The distinction between constant bandwidth schemes and variable bandwidth schemes makes a significant difference in the complexity of system design. We categorize continuous media as being either ***regular*** or ***irregular***. Regular continuous media has deadlines occurring at regular

intervals, with a fixed amount of data being required at each deadline. Irregular continuous media has deadlines at irregular intervals and/or requires variable amounts of data at each deadline.

An example of regular media is video. A video is made up of a number of pictures, or frames, which are successively displayed. There is no strict timing requirement for when the first frame must appear. However, once it does appear the successive frames must be displayed at a regular rate for the desired smooth motion of the video to be achieved. Playback of digital audio proceeds in a manner analogous to video playback. A digital audio record consists of a number of samples taken at fixed intervals of the original analog source. To play back a digital audio record, the samples must be fed to a digital-to-analog converter at the precise rate at which they were sampled. Any deviation from this precise rate will result in audible clicks, pops or pauses.

Audio and video have real time deadlines which repeat in a regular manner. Other media types may have more irregular deadlines. For example, a slide show consists of a number of images which have display times set by the composer (commonly timed to music). Thus it will have deadlines which may fall in to any sort of pattern according to the whim of the composer.

Another way for media to be irregular is to require different amounts of data for each deadlines. For example, an animation may be implemented by storing graphic display commands (such as line draw or area fill). Some scenes in the animation may require many commands, and others very few. In general, variable rate compression schemes lead to irregular CM, and one can easily think of graphic commands as a compressed format for storing images. MIDI[7] data will also be

[7] Musical Instrument Digital Interface – a standard for transmission of musical data. In contrast to digital audio which samples the actual sound waves, MIDI information consists of records of notes played, e.g., F# played for a duration of 0.5 sec with a volume of 76 out of 128.

irregular, since the MIDI composer will sometimes play very few notes, and other times play many notes simultaneously. One could think of MIDI as a compressed format of digital audio.

Irregular CM is difficult to handle simply because it is irregular. The simplest way to deal with irregular media is to characterize it in terms of its worst case bandwidth requirements, and thus over-allocate resources to be safe. In order to utilize resources more efficiently, a more precise mathematical characterization of the bandwidth requirements is required. Several proposals for characterizing irregular CM have been made (see section 4.4) but this is an area of on-going research. In this thesis, we will focus on regular CM, except where irregular CM is explicitly mentioned. Note that constant rate compression simply changes the data rates required, it does not alter any of the basic properties of regular media, so although we will not always explicitly mention constant rate compression it can be assumed included in our discussion.

1.2.2 Single-Stream Playback

Digitization of video yields a sequence of frames and that of audio yields a sequence of samples. Since media quanta, such as video frames or audio samples, convey meaning only when presented continuously in time, a multimedia server must ensure that the recording and playback of each media stream proceeds at its real-time rate. Specifically, during recording, a multimedia server must continuously store the data produced by an input device (e.g., microphone, camera, etc.) so as to prevent buffer overruns at the device. During playback, on the other hand, the server must retrieve data from the disk at a rate which ensures that an output device (e.g., speaker, video display) consuming the data does not starve. Although semantically different, both of these operations have been shown to be mathematically equivalent with respect to their real-time performance requirements [10]. Consequently, for the sake of clarity, we will only discuss

techniques for retrieving media information from disk for real-time playback. Analysis for real-time recording can be carried out similarly.

Continuous playback of a media stream consists of a sequence of tasks with deadlines, where tasks correspond to retrievals of media blocks from disk and deadlines correspond to the scheduled playback times. With regular media, these scheduled times are periodic, and we can talk of the ***consumption rate*** of the system – the rate at which data is consumed for playback. Although it is possible to conceive of systems that would fetch media quanta from the storage system just in time to be played, in practice the retrieval is likely to be bursty. Consequently, information retrieved from the disk may have to be buffered prior to playback[8].

Because the data transfer rates of disks are significantly higher than the requirements of a single stream (aside from uncompressed video, all data types are well below the 2-4 MB/sec of a modern disk – consult Table 1), employing a modest amount of buffering will enable conventional file and operating systems to support continuous storage and retrieval of isolated media streams. However, such solutions are not always rigorous.[9] We now give a precise formulation of the problem, and a solution which is rigorous.

[8] Just-in-time retrieval is simply a special case where the buffer size is limited, so that the server cannot produce data ahead of schedule without buffer overflow.

[9] For example, the documentation for the Multisound™ sound card by Turtle Beach Systems for use with Microsoft Windows™ contains the following advice: "When you record ... your hard disk may not be able to accept incoming information that fast ... and the recorded file will have audible stuttering".

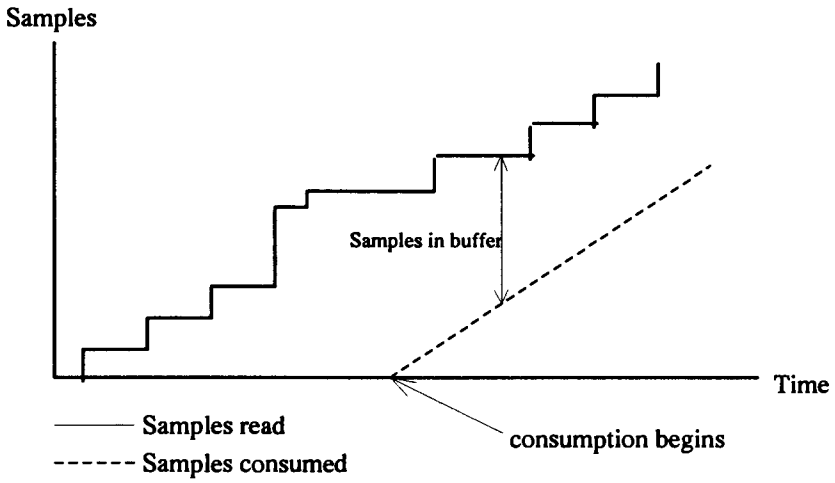


Figure 1: Ensuring continuous retrieval of a media stream

The real time deadlines for playback can be met if, after playback begins, the output buffer always contains some data (see Figure 1). It is therefore always possible to achieve playback by pre-fetching all the data into the buffer before playback begins. However, this will require an inordinate buffer size, as well as creating a long delay during pre-fetching before playback can begin. The problem then becomes one of minimizing buffer requirements and pre-fetching delays.

In [38] we have studied these minimization problems in detail. It is shown that they are in fact the one and the same; that minimizing one will minimize the other. The general solution for the minimization problem is as follows. First, let the following functions be defined:

$R(t)$ - The number of media quanta read from the storage device at time t.

$C(t, t_0)$ - The number of quanta consumed at time t, with consumption beginning at time t_0.

$B(t, t_0)$ - The number of quanta buffered at time t, with consumption beginning at time t_0.

If the D/A converter consumes quanta at a rate of r_c, then the number of quanta consumed is

$$C(t,t_0)=\begin{cases} r_c(t-t_0) & t \geq t_0 \\ 0 & t < t_0 \end{cases} \quad (1)$$

The number of quanta buffer will be the number read less the number consumed:

$$B(t,t_0) = R(t) - C(t,t_0). \quad (2)$$

We say that a start time, t_s, is ***feasible*** if $B(t,t_s)$ is always non-negative[10]. To find the minimal start time, and buffer space requirements, we first consider $B(t,0)$. If $B(t,0)$ is always non-negative, then it is the minimum feasible solution, and we are done. However, if $B(t,0)$ is negative at some point then let the minimum value for $B(t,0)$ be $-m$. The intersection of $R(t)$ with $B(t,0)+m$ is at the minimum feasible start time, and this start time yields the minimum buffer space requirements.

This solution is rigorously defined in [38]. Intuitively, we are taking advantage of the fact that for any choice of t_0, the graph of $B(t,t_0)$ has the same shape for $t>t_0$. Therefore, the graph of $B(t,t_0)$, for any choice of t_0, will look like $B(t,0)$ shifted upwards by some constant and intersected with $R(t)$. By shifting $B(t,0)$ up by just enough to make it feasible, we find the minimal feasible solution (see Figure 2).

The above solution is completely general. However, it requires knowledge of the exact timing of data retrieval in advance of playback, which in many cases is difficult to compute. By using upper bounds (worst case) figures for retrieval times, it is possible to simplify matters and quickly calculate buffer space and start time requirements. In [38] such simplifications are applied to several scenarios. We highlight some of the results here:

---

[10] Strictly speaking, $B(t,t_0)=0$ would mean that the buffer is empty and real time deadlines will fail to be met. However, we model consumption as a continuous function, while in reality it occurs in discrete one-sample steps. Therefore, if the discrete model would have rested at zero, our continuous model would go negative, and thus, we can use $B(t,t_0)<0$ as the error condition.

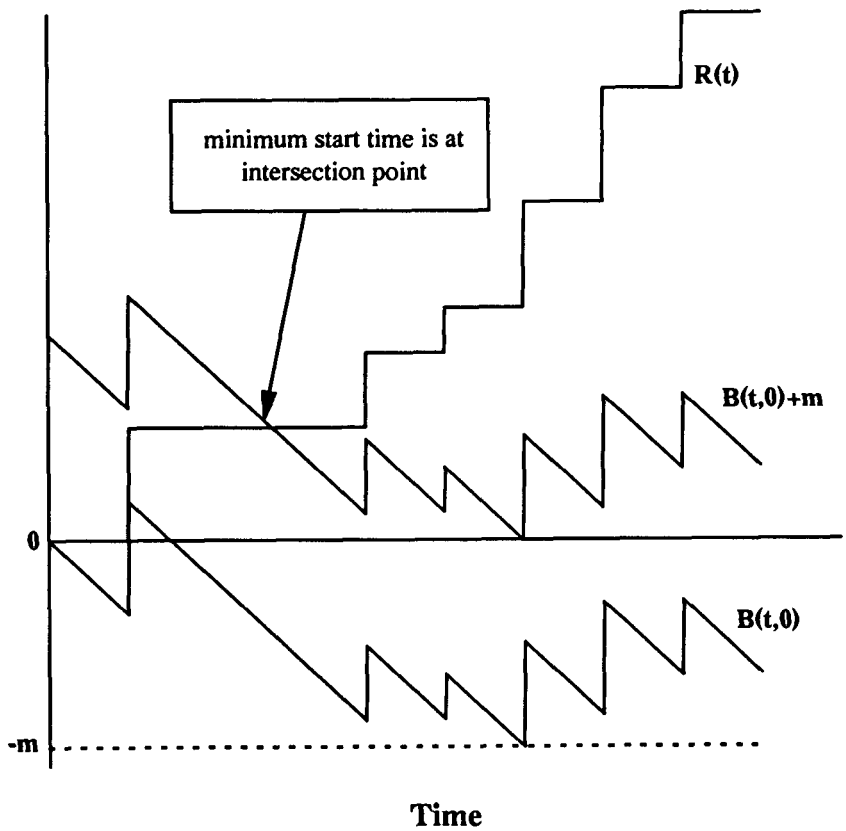


Figure 2: Finding the minimal start time and buffer space

1. If the consumption rate exceeds that of the storage device transfer rate, then buffer space requirements will be linearly dependent on the length of the playback. For most systems this would be an undesirable property.

2. If the device transfer rate exceeds the consumption rate, then some delays must be introduced in reading to keep buffer requirements reasonable. Given a minimum delay figure, buffer space requirements are formulated.

3. For many systems there will be unavoidable delays during reading (e.g. seeks). Given a bound on these delays, buffer space requirements are formulated.

The scenario described under number 1 is a case of using hardware which is inadequate for the application. It results in having an extra parameter (playback length) included in all calculations. In order to avoid this unnecessary complication, we will always assume that the transfer rate of the storage device is at least as great as the consumption rate.

1.2.3 Multi-stream Playback

A CM server, of course, will have to retrieve many media streams simultaneously in order to serve all its clients. While some situations may require the retrieval of only one stream with a multicast to all clients, in general different clients will request different streams. Even when the same stream is requested by clients, it is unlikely that they will retrieve the stream synchronously – at any given time they may be viewing different portions of the stream. Different clients may be requesting different kinds of media (e.g. one client may request audio only, while another requests video). It is also a significant point that users will desire independent control of the playback; the server must allow for some clients to pause their playback while others continue viewing.

One could handle the problem of multi-stream playback by simply throwing hardware at the problem – dedicate a storage disk for each stream to be retrieved. However, this is expensive and under-utilizes the hardware. The transfer rate of modern disks is significantly higher than the requirements of most CM streams (especially if compressed). Therefore, many more users can be supported if disk use is multiplexed between users. It is this multiplexing that forms the central problem in CM server design, because any multiplexing must ensure that all real time deadlines are still met.

In the design of a multimedia server, many issues need to be addressed, including disk scheduling, admission control, storage management, and file system implementation. The primary

contributions of this thesis are in the area of disk scheduling. We will outline strategies for use of multiple-disk systems, but our studies are directed toward scheduling of a single disk (or multiple disks which are accessed as a single logical disk, e.g. RAID systems).

1.3 Overview of thesis

We can divide the issues related to a file server into two areas: those relating to network transmission, and those relating to the implementation of the server itself. As we shall see in the survey, the area of network transmission is already well covered. We will focus on issues directly related to server design, with particular emphasis on meeting real time deadlines for retrieval and storage. Furthermore, synchronization and combination of media objects will not be covered, as they may be performed at the display site rather than the server. The reader may consult the work of Little and Ghafoor [62] for a discussion of where synchronization and combination should take place.

As an aid to clarity, we will sometimes focus our presentation on audio retrieval. Adapting the concepts to other media types and to storage is straightforward. Because our work is applicable to a number of media types we will use the words viewer, listener, client, and user synonymously, and will also treat the words display, playback, present, and output as synonyms.

Chapter 2 presents a survey of literature related to CM server implementation. Chapter 3 discusses the real time requirements of multi-stream retrieval, including disk scheduling. Chapter 4 deals with issues related to implementing a file system that supports continuous media, such as data placement, and file structures. Chapter 5 describes our experience with the Audition system, which implements a number of the ideas from this thesis.

2. SURVEY

In this chapter we present a survey of literature related to multimedia file servers. Because of the nature of multimedia file servers, applicable ideas are found in a broad range of literature; the papers we survey come from such diverse sources as the Journal of Audio Engineering Society and ACM Transactions on Information Systems. Only very recently (the past year) have journals such as IEEE Multimedia and Multimedia Systems come into existence. For this reason, our focus in this chapter is on completeness rather than detail. We hope to create a survey here that is otherwise hard to come by, and to fill out a larger perspective on the problem of CM server design. In so doing, we will naturally include some papers that go slightly beyond the scope of our own research. We will cover material directly related to our work in more detail just prior to taking up the subject ourselves. In particular, sections 3.2.1, 4.1, 4.3, and 4.4 give more detail on some of the papers surveyed in this section.

In the main, we have organized the papers in this chapter by topic. However, we have made an exception in the case of papers from the University of California, San Diego (UCSD) and from University of California, Berkeley (UCB). This is because they both have a large volume of papers, most of them related to some project at their respective universities. In order to keep these works in context we devote a section to each of these schools.

2.1 Historical Perspective

The earliest work on multi–stream audio retrieval came from the professional audio world. The researchers in this field were attempting to replace the multi–track tape recorder with a multi–stream digital audio recorder. Besides gaining fidelity by switching from analog to digital, they

also gained better editing capabilities by switching from tape to a random access media such as disk drives. As all the streams are being played to a single user in this context, synchronization between them is assumed. Furthermore, the materials being played back are homogeneous in nature, and generally will have the same consumption rate as well. Unfortunately, the literature from the pro–audio world yields little insight [17, 47, 50, 53, 69, 120, 133]. Papers tend to be either ad–hoc in their approach, or else they hide important details to protect their market share. In many cases excessive hardware is utilized in place of finding optimal solutions.

The traditional file server was designed to provide access to text and numeric information. The first wave of multimedia research widened the scope of this service to include support for documents containing images. Examples include the Diamond system [122], and the Muse system [41]. The next wave of multimedia research involved the addition of audio and video in analog form. It included the Etherphone project's support for video using analog transmission and storage [100], and Mackay and Davenport's work on video filing using consumer electronic devices [66]. As a final step, work on storage, retrieval and transmission of multimedia data all within the digital domain has taken place. In particular, the concept of an "on-demand" digital video server, which provides services similar to a neighbourhood video tape rental store over a metropolitan area network, has attracted attention [52, 107].

2.2 General

A survey of multimedia technology is found in [32]. It includes discussion on interactive videodisks, CD-ROM, CD-DA (regular compact disks), CD-I, CDTV, DVI, the Next computer, high speed networks, audio/video capabilities (ranging from slide projectors to CD-ROM), compression methods, and standards. An overview of the digital video interactive (DVI)

technology can be found in [44]. A survey of approaches to continuous media retrieval is found in [114].

Watkinson surveys the state of the art in digital audio recorders in [133], and provides an excellent reference for digital audio in general in [134]. Another good primer on digital audio is [81]. Grusec et al., discuss the compression of audio utilizing phsyco-acoustic models, and methods for evaluating such compression systems [48].

Clark discusses the difference in requirements between the authoring of multimedia presentations, and (interactively) viewing them [29]. He argues that it is misguided to build general purpose multimedia systems which cover both authoring and viewing; rather, that systems should be designed with one or the other in mind. We agree with Clark, and believe that only primitive editing facilities will be used by the average multimedia user; serious multimedia editing will be done by production houses, like the ones that produce CD-ROM titles today.

In [65], window systems are extended to include audio. The authors discuss the use of various audio effects to organize multiple sound sources, just as various visual effects may be used with windows (e.g. tiling, iconizing) to spatially organize and shift attention between them.

Gibbs et al., give a rigorous proposal for data modelling of continuous media [42]. They come up with a simple yet comprehensive model that includes media objects, media elements and timed streams. They then define structuring mechanisms (such as composition).

Chang and Zakhor discuss admissions control and data placement for variable bit rate (irregular) continuous media [22]. They compare two data placement techniques: constant time length (CTL) and constant data length (CDL).

Tangi [117] presents a video and audio disk file system, based on an analog, FM optical storage disk. While not in the digital realm, it does discuss how fast forward and fast reverse operations are performed, giving a method that may be applied to other disk technologies.

Work on general purpose real time systems has been extensive. Some representative papers are [56, 63, 84, 139]. By being general purpose, they miss exploiting the properties of continuous media (see section 3.2).

2.3 Speech/Statistical Transmission

Strathmeyer [115] provides an overview of available technologies for voice. This includes basic digital audio encoding/decoding, speech recognition, connection control (e.g. telephone signalling), and example applications.

Two papers, one by Malek [67] and one by Chen and Messerschmitt [26] provide a basic background to voice and data network communications. The Malek paper begins by discussing current voice communication networks (i.e., the public phone system), and the evolution of voice transmission and switching technologies. It then presents communication concepts for data, including switching techniques and network architectures. It briefly covers the issues involved in integrating voice and data, and then surveys the Integrated Services Digital Network (ISDN) which does just that. Chen and Messerschmitt concentrate on issues related to integrating voice and data. They discuss packet switching, circuit switching, and hybrid methods to combine data and voice at the switching level (c.f. ISDN which uses two physical carrier networks, one for data, one for voice).

Much of the work on speech transmission in packet networks relies on the statistical nature of conversations, i.e., that the average speaker is silent 40 percent of the time during a phone

conversation. This is sometimes referred to as "talkspurts". To take advantage of this property, a common approach is statistical multiplexing [18]. Assuming that a voice channel only uses about 60 percent of its capacity, this scheme overbooks the network to fully utilize it. However, it is possible for channels to use above their predicted rates, in which case the network is overloaded and some packets must be discarded. Gilge and Gusella modify this approach for compressed video, and dynamically adjust their compression according to available network bandwidth [43]. This results in lower quality when then network is heavily loaded. A similar approach is taken by Park and English for audio; only a certain number of the most significant bits of the audio data is transmitted when the network is heavily loaded [78]. Bially et al., [21] also handle the problem by reducing channel bit rates when necessary.

Woodruff and Kositpaiboon [136] look at integrated data/voice/video transmission in a broadband multimedia network (e.g. B-ISDN) using the asynchronous transfer mode (ATM). The ATM is fixed-length packet transport scheme which they use by statistically multiplexing bursty traffic flow at the expense of delay and loss. They find that connection peak rates must be low relative to network link speed to yield high probabilities of meeting deadlines. This approach has some similarity to Ferrari's [30,31] work (see section 2.8) in that it is preventative rather than corrective; i.e., problems are avoided at session set-up rather than waiting for them to occur. It differs in that no guaranteed class is offered. They find non-statistical best for everything except bandwidth utilization. Their method is to reserve resources in a way that is statistically more than enough, but non-statistically may fail.

Leung et al., [60] consider implementing delay-sensitive data transmission using packet-switched networks. They propose a method employing virtual circuits, which allows applications to multiplex several virtual circuits to create a synchronized "multimedia virtual circuit". Within the

virtual circuit, channels can be flow-controlled, or not flow-controlled (e.g. voice would not be flow-controlled). In reserving a call, the bandwidths requirements are specified in terms of average and peak bandwidth, and also degree of sensitivity to delay. For real time data, packets may be dropped in order to keep the stream data up to real time deadlines; that is, not all data is guaranteed to arrive, but that which does will meet its deadline (and presumably only a very small fraction would not arrive).

2.4 Network Servers and Workstations

The Xerox PARC Etherphone system integrates audio and video a local area network [100,121,132]. The audio and data are transmitted over an ethernet. Video is handled by separate analog connections under computer control. The system allows conferencing between network users and also includes connections to the public telephone service, allowing a user to place calls, record messages, etc. The system is aimed at speech quality audio, assuming talkspurts and not guaranteeing error-free, real time playback. The papers include discussion of how the system tracks references to an audio record by "interested" users. This allows data to be shared when several parties are interested, and deleted when no one is using it any more.

The AudioFile project at DEC implements a device-independent, network-transparent computer audio system [61] . It functions in a manner similar to X-windows (in fact, they copied much of their code from X). They make the "fundamental ... assumption that the file system can supply audio data faster than it is required". In other words, they cannot guarantee meeting real time deadlines. Their work is interesting in the way that it makes explicit use of time. For example, rather than just initiating a record operation, as most systems would, their record function

specifies a time that recording should begin. This time may be in the past, in which case it is satisfied by pre-buffered data.

Kamel et al., [55] look at integrating voice in workstations by adding telephones and a specialized "circuit server" which connects the telephones and the public telephone network to the local area network. Each workstation can send control messages to the circuit server. Voice files are stored as conventional files on the workstation's file system (c.f. Etherphone which stores voice in a database). Aside from a brief mention of buffering included at the circuit server (32 KB), which points out how interrupt latency is affected by buffer size, there is no discussion of how real time demands may be guaranteed, or even statistically met. Perhaps because the performance requirements of voice are so low this has not been an issue for the authors.

Recently, work on disk scheduling for CM network servers has increased. Reddy et al., have create a hybrid of the SCAN and EDF disk scheduling algorithms called SCAN–EDF [108,109]. Concurrently with the work on this thesis, researchers at IBM's T. J. Watson Research Center have developed the GSS disk scheduling algorithm [25,26], which is functionally equivalent to our sorting set approach (see section 3.2 for more details on disk scheduling).

Other work from the T. J. Watson Research Center considers the problems of supporting fast forward and fast reverse playback of video from disk-array-based servers [24]. With disk arrays it is possible to store and retrieve data using any number of strategies, and several papers have been written on this subject [20,23,123] (see section 4.1.2).

2.5 Multi-stream Synchronization

Nicolaou has discussed multimedia networking, focusing on synchronization of related data streams and on handling heterogeneous multimedia hardware [76]. Synchronization is handled by

defining synchronization properties at the presentation level and giving application control and synchronization operations which utilize these properties. Heterogeneity problems are addressed by separating the data transport semantics (protocols) from the control semantics (protocol interfaces). For real time transmission, prioritized packetization is applied, and a protocol with no acknowledgements or retransmissions. The justification is that late packets might as well be lost, so if a packet doesn't transmit correctly it should be dropped and transmission on the next begun (a la Leung et al., [60]). Real time data streams are given a quality of service (QOS) parameter. Nicolaou does not favour multiplexing data for synchronization, pointing out that multiplexing would require streams with very different QOS parameters to transmit using a single QOS parameter and would also mean the streams must originate at a single point.

Little and Ghafoor [62] consider the problem of "composing" distributed multimedia objects. For example, images from different sources may be overlaid on a screen, or several audio channels may be mixed together. They examine where in a distributed environment such composition should be done. For instance, network bandwidth can be saved if image overlays are computed at the server before transmission, while synchronization of temporal data is better handled at the destination.

Shepherd and Salmony examine how OSI can be extended to support multimedia traffic, and in particular how to support synchronization of multiple data streams [113]. One suggested method inserts "synchronization markers" into the data streams at the sender. The receiver can then use these markers to synchronize the streams (via buffering). Their other suggestion is to carry an additional synchronization channel alongside the data streams, which indicates the nature of the synchronization required and also references to the points in the data streams which must be synchronized.

2.6 Retrieval and Browsing

Christodoulakis et al., [27,28] examine browsing of multimedia data, including voice data. Some of their ideas include "paging" through audio data by defining "pages" of a fixed length or having page boundaries at pauses in speech, identifying logical components of a speech record, and pattern matching.

Irven et al., [54] present a prototype of a multimedia information services network. The focus of the paper is on browsing techniques, and user interface issues, with some references to the technical requirements for such a network. The paper provides a historical overview of networked information services.

Retrieval of multimedia documents is discussed in [70]. Object-oriented semantic data models are used in the Multos system to improve the efficiency and effectiveness of document retrieval.

Ramanathan and Rangan consider the future in which most media is not broadcast, but is retrieved on demand. They envision that on-demand services are likely to evolve into personalised services that are customized to a users needs, with customization carried out by intelligent *Personal Service Agents* (PSAs), which negotiate with different content providers to schedule viewing selected programs at convenient times for the user. They then discuss the architectural considerations involved in implementing PSAs.

2.7 The UCSD Multimedia Laboratory

The work at the UCSD Multimedia laboratory is summarized in [93]. Their work falls into two categories: multimedia on-demand services (access to non-live media sources) and multimedia collaboration services (access to live sources). They have demonstrated prototypes of some of

their work [94,96,105]. Some of the Xerox PARC Etherphone work has been done in conjunction with the UCSD Multimedia laboratory [100,132].

For on-demand services, their approach is based on "constrained placement" of multimedia data, i.e., they bound the distance between successive blocks of a file in order to limit seeking latencies [102]. This approach requires elaborate algorithms to control of data placement for storage [97,103]. During retrieval, they utilize a servicing algorithm which retrieves a number of blocks proportional to the consumption rate for each channel in each "service round" [107]. A possibly maximum seek is incurred as reading switches between channels. Ideally, the number of blocks to be read for a channel may be fractional, which is practically impossible to accommodate. To get around this problem, they propose a "staggered toggling" technique, which toggles between reading the floor and the ceiling of the fractional amount [129]. The toggling up for one channel is matched with the toggling down of another so that overall there is no net increase in the service time of any reading period. Their algorithm permits dynamic additions and deletions of channels without causing any discontinuity of service to existing channels. This is done by allocating enough time to read one extra block for each channel during playback. When an additional channel is requested, the amount read can be increased by single blocks without missing any deadlines, until the read size permits the addition of the extra channel (i.e., buffers enough for the time to read for that channel as well). They discuss admission control in [128].

In order to deal with synchronization requirements between multiple channels, they have developed feedback techniques and protocols which allow for network jitter and non-deterministic mismatches in playback rates of media capture/display sites. This is in contrast to other work which assumes global clocks or single-site workstations. The algorithm has the multimedia server creating a relative time system in which recorded blocks whose recorded times

are within a bounded time window are assigned the same relative time stamp. During retrieval, the display sites periodically transmit light-weight messages called feedback units back to the multimedia server. Feedback units are transmitted at the moment that playback of the block begins. The server can then use an estimate of the transmission time of the feedback unit to detect impending asynchronies at display sites. The server can then adjust its transmission to bring the display sites back towards synchrony. They show that media synchronization different from clock synchronization, and that clock synchronization becomes unnecessary and infeasible when the objects to be played back have been recorded at different sites and different times [85, 86, 87, 89, 98, 99, 106].

The UCSD work on multimedia collaboration services deals with both collaboration management and media mixing. For collaboration management, they have proposed a taxonomy of multimedia collaborations that include various types of conferencing [95,104,90]. They have also put forward a model for collaborations that includes streams (at the lowest level), sessions (semantically related streams) and conferences (temporally related streams) [101,130].

Media mixing involves combining the many media streams from various sources that exist during a collaboration into a single composite image and audio stream for display to the user. They have developed a mixing algorithm designing to work without globally synchronized clocks, and with transmission delay jitter [106]. They also propose a hierarchical mixing architecture for supporting scaleable conferences, and have developed algorithms to design hierarchies which minimize end-to-end delays. For non-hierarchical networks such as the internet, they propose a "Packet Train" protocol, which allows routing nodes in the network to perform mixing, thereby integrating mixing with routing [91,92,131].

2.8 UCB Publications

Ferrari and Verma, from the University of California at Berkeley (UCB), study real time channels in wide area networks [30,31]. They assume a store-and-forward network in which the link between each node pair has a known and finite bound on the link delay of each packet, otherwise real time guarantees could not be offered (Note: this means contention-based networks, e.g. Ethernets, are unsuitable). Their paradigm provides three levels of service: deterministic, statistical, and "other". The deterministic class guarantees meeting real time deadlines, the statistical class promises to meet real time deadlines with a certain probability, and the best effort class is for data without real time requirements. Clients are required to declare their traffic characteristics and performance requirements at the time of channel establishment, at which time a connection is either established or the request rejected. The characteristics of the channel are described using the x_{min}-x_{ave}-I model, which specifies s_{max}, the maximum message size, x_{min}, the minimum spacing between messages, x_{ave}, the maximum value for average spacing between messages, and I, the averaging interval for x_{ave}. Scheduling at the nodes is done by establishing a queue for each level of service and using a modified deadline strategy. Flow control is rate based, using the fact that at channel establishment time, the receiver can check whether it will be able to accept packets at the rate declared by the sender. This work is related to the DASH project (see below) but takes a slightly different approach. See Verma's dissertation for a comparison of the approaches[125]. Ferrari also co-authored a paper discussing pricing policies for networks implementing this sort of scheme [79].

An overview of the DASH project at UCB is given in [4,5]. The project is studying large high-performance distributed systems of the future and building an experimental system to aid the study. Their envisioned system of the future includes high-powered workstations on a high speed

network, with support for continuous media. Their interprocess communication (IPC) includes real time "message streams" [12], and to support this they employ pre-emptive deadline scheduling and eliminate copying of data in IPC to gain the necessary speed for real time applications [13,124]. System resources (disk, CPU, network, etc.) are reserved and scheduled in an end-to-end fashion, to support "sessions" of continuous media transmission [3,14]. A session has some values including a maximum message size (in bytes), a maximum message rate (in messages/second), a maximum burst size (in messages), a maximum delay, and a minimum delay [16,15] (this characterization of a session is called "linear bounded"). A session can also specify how reliable (deterministic, statistical or best effort) and secure (i.e., private) it needs to be [2]. These values are used to calculate needed resources, including network bandwidth and node buffering. They assume that the source generates messages fast enough to maintain a non-zero backlog. By ensuring that all messages are delayed at the receiving application to make the total delay at least the maximum delay then all output deadlines will be met. Because some parameters, e.g. delay, will be cumulative over all the nodes in the transmission, the session is first reserved for maximum performance available, and then if the end-to-end requirements are exceeded, the requirements at each node may be relaxed. Deadline scheduling is implemented at each node for transmission of messages. The DASH project has already implemented the Session Reservation Protocol (SRP) for network communications [8]. It uses the linear bounded arrival process for its model, and is an extension to TCP/IP. They plan to add to their system modifications of MACH and X11[11] [7].

Also from UCB, Verma's Ph.D. dissertation discusses guaranteed performance in B-ISDN networks using the Asynchronous Transfer Mode (ATM) [125] . His approach is to have the

[11] See the ACME project, below, for a description of some of the extensions to X11.

network enter into contracts with the clients to provide services. A contract must be specified, and mapped on to the network in an end-to-end fashion. A contract must be satisfiable (i.e., algorithms must exist for both the client and the network to satisfy the contract) and verifiable (i.e., algorithms must exist to verify that both the network and client are fulfilling their requirements under the contract). The specification problem is divided into two parts: the traffic specification, which the sending client must abide by, and the performance specification, which the network must provide. He compares some traffic models (including the linear bounded model used by DASH) and finds the x_{min}-x_{ave}-I model to be the best [31]. For performance specification, he proposes choices of quality of service based on delay, delay variation and packet loss rate. The algorithms he advocates for satisfiability are a leaky bucket scheme for traffic and admission control for performance. Verification of traffic is achieved via rate-control, while verification of performance is verified via statistics at the receiving end.

Moran and Wolfinger adapt the x_{min}-x_{ave}-I model to be more suitable for variable rate data (e.g. variably compressed data) [72]. They do this by setting $x_{min} = x_{ave}$ (i.e., the time between transmissions is always the same) and allowing differing amounts of data to be sent. Their protocol specifies the minimum, maximum, and average amount of data to be sent, as well as the period over which averaging is done. Their paper goes into some detail describing the design of a transport service and protocol for continuous media.

Work on file systems and disk storage/retrieval of continuous media at UCB is described in [10,11,82]. Session reservation is done as in the DASH system. Real time demands for playback are met by employing "workahead conserving" sets of reads; i.e., by the time the set of reads completes, more data has been read for each channel than has been consumed. Non-real time data is serviced with any excess time. In [10,11], their prototype employs reads of varying lengths on

contiguous files. In [82] this is altered to allow non-contiguous files, with a description of the file layout stored in the file descriptor (however, the ideal for them is still contiguous files, and the file is expected not to have many discontinuities).

The ACME project at UCB includes an I/O server for continuous media [9,51] . They assume a heterogeneous network which includes continuous media. Their server is a low-level software layer offering network-transparent access to continuous media I/O hardware, in a similar way a network window system, such as X, provides access to a display, keyboard, and mouse. ACME supports split-level CPU scheduling of lightweight processes in multiple address spaces, and memory-mapped streams (c.f. DASH message passing without copying) for data movement between address spaces [45]. The extensions to X include strands (streams of audio or video data), ropes (combinations of several strands), logical time systems (reference frames in which several strands or ropes can be played synchronously), and logical devices (representing microphones, speakers, video cameras and video windows) [6]. To implement flow control a buffer is defined to have a near empty point and a near full point. When the amount of data falls below the near empty point, a request is made for more data. When the amount of data rises above the near full point, a request is made for transmission to slow down or stop. The authors deal with such problems as start up synchronization and differing clock rates.

2.9 Commercial Multimedia Servers

Recently, a significant amount of work has been done in commercial development of multimedia servers. The products resulting from these efforts range from low-end servers intended for small work groups to high-end servers designed to provide service to thousands of users.[12]

The low-end servers are targeted for a local area network environment. The clients are typically PC's, equipped with video-processing hardware (such as the IBM/Intel ActionMedia adapters). The multimedia files generally consist of short video clips. Common applications are on-site training, information kiosks, or they may even be employed as small-scale video servers for environments such as hotels and conference centers. For example, the IBM LANServer Ultimedia product [19] can serve 40 clients at MPEG-1 rates. This product extends the capabilities of the LANServer file system to support for multimedia streams, using a file-system oriented client interface (see section 4.2). Other systems in this class include FluentLinks, ProtoComm, and Starworks [123]. To obtain sufficiently large storage capacity and throughput, disk arrays are commonly employed.[13] As the PC's continue to evolve and gain more power, it is expected that these servers will be able to support more clients.

Most servers also impose admission control mechanisms to restrict the number of simultaneous users. These admission control mechanisms take into account the load on the disks, as well as the processing overheads related to disk and network access. They also implement simple network load control mechanisms such as limiting the number of clients on a LAN segment. However, at

[12] I am indebted to Dilip D. Kandlur for the material in this section. It was developed as part of a paper we co-authored.

[13] It is noteworthy that many state-of-the-art disk controllers, such as those for SCSI-2 systems, now implement a SCAN-like algorithm for rescheduling I/O requests (see section 3.2).

present, there is no generally accepted standard for bandwidth reservation on the LAN. Hence, some systems such as FluentLinks, also adapt to changes in the network load to scale down the multimedia traffic.

High-end servers are targeted for applications such as video-on-demand, in which the number of simultaneous streams is expected to be in the thousands, and the distribution system is expected to be cable-based, or telephone-wire-based. Since the distribution area is large, network connectivity is an important aspect of these systems and high-speed networking technology such as ATM is frequently employed. In order to provide a large collection of videos in a cost-effective solution, a hierarchy of storage media is required, which ranges from high-cost, high-bandwidth semiconductor memory through disk storage to low-cost, high-capacity tape/disk libraries. Note, also, that any admission control mechanisms must be extended to the distribution network, including allocation of bandwidth on the backbone network and TV "channels" on the cable plant. The control mechanisms must also interact with a large transaction processing system to handle bookkeeping operations such as authorization and customer billing.

High-end video servers are based on collections of powerful workstations (IBM, DEC, Silicon Graphics, Oracle/NCube) or mainframe computers (IBM). The SHARK multimedia server is a stream server. It runs on the RS/6000–AIX platform and uses its own file system [49] to ensure continuous throughput from the disk subsystem. Microsoft's TIGER video server uses a collection of PCs to construct a scaleable server [71]. It uses striping to distribute segments of a movie across the collection of servers to balance the access load across the servers. It also uses replication at the segment level as a mechanism for fault-tolerance. Oracle's Media Server is based on the NCube massively parallel computer. It exploits the large I/O capability of the NCube and is slated to deliver approximately 25,000 video streams.

3. MULTI–STREAM RETRIEVAL

In this chapter we turn our attention to simultaneous playback of multiple media streams. Almost all approaches to multi–stream CM retrieval have the following two characteristics:

- ***Processing stream requests in reading periods:*** Due to the periodic nature of CM playback, a multimedia server can service multiple streams simultaneously by proceeding in ***reading periods***. During each reading period, the multimedia server can retrieve a sequence of media blocks for each stream. There is no gap between successive reading periods.
- ***Production keeps up with consumption in each reading period:*** During each reading period, the amount of data retrieved for a stream is at least as much as will be consumed by the playback of the stream. This means that in each reading period, the production of data never falls behind the consumption, and there is never a net decrease in the amount of buffered data. We say that algorithms having this property are ***buffer–conserving***.[14]

It is conceivable that an algorithm may be developed which proceeds in reading periods, but is not buffer–conserving. Such an algorithm would allow production to fall behind consumption in one reading period, and then make up for it in a later reading period. However, this would necessarily be more complex. Furthermore, while buffer-conservation is not a necessary condition for preventing starvation, it can be used to guarantee starvation. For instance, before initiating playback, if enough data is pre–fetched so as to meet the consumption requirements of the longest possible reading period, and if each reading period thereafter is buffer-conserving, then it is clear that starvation is impossible.

[14] Another term that has been used is "workahead–augmenting" [11].

No algorithm will be buffer-conserving at all times. When all reading is completed, several reading periods may be required to consume the last of the data, and of course these reading periods will not be buffer-conserving. Furthermore, most algorithms will not need to maintain buffer-conservation during certain conditions, such as the buffer being full and/or the user pausing output. However, this is not a matter of falling behind consumption and later catching up, it is a matter of getting so far ahead of consumption that reading must be paused to avoid buffer overflows. The essential strategy is still one of buffer-conservation.

In order to read enough data to maintain buffer-conservation, it is clear that the maximum duration of each reading period must be anticipated. Since the duration of a reading period is governed by the total time spent in retrieving media blocks from disk, the number of blocks retrieved and the disk scheduling algorithm used are critical. Furthermore, the server must employ admission control algorithms to ensure that it is not attempting to service more clients than is possible without violating real time demands.

3.1 Meeting Real Time Requirements

In this section we will first give a rigorous description of the real time requirements of multi–stream retrieval, irrespective of disk scheduling (we will assume processing in reading periods, and buffer-conservation). We will begin with the simplest case where all streams are synchronized with the same consumption rate, and move on to consider asynchronous requests with different consumption rates.

3.1.1 All Channels Synchronized

When all streams are synchronized and have the same consumption rate, there is a uniformity that allows a straightforward formulation of solutions. In [38] we consider the following scenario:

- There are n_p playback streams.
- Each stream has the same consumption rate, r_c.
- Playback is divided into reading periods during each of which a logical block of data is read for each stream.
- During each reading period there are delays (e.g. due to seeking latencies) which are bounded above by d_{max}.

We also define the following parameters:

- r_t: The transfer rate of the storage device.
- b: The number of bytes in a logical block.
- p_{max}: The maximum length of a reading period.

Of particular interest to us are the maximum reading period length, p_{max}, and the logical block size, b. This is because the reading period is indicative of the delay between a request for playback and sound actually being heard, while the logical block size gives a good indication of the amount of buffer space required. For example, consider a simple double–buffering system. In such a system, data is consumed from one buffer while the other is filled. In this case we could have each buffer equal in size to a logical block, and logical block size is then always half of the buffer requirements.

Regardless of what sort of buffering strategy is implemented, we can approach the problem as follows. First, the worst case (maximum) reading period length is the maximum delay time, plus the time needed to transfer one logical block for each stream:

$$p_{max} = d_{max} + n_p b / r_t \tag{3}$$

For retrieval to be buffer-conserving, there must be enough buffered in each reading period to satisfy consumption for the duration of a reading period; that is,

$$b \geq r_c p_{max}. \tag{4}$$

Equations (3) and (4) can be combined to eliminate b and obtain

$$p_{max}(1 - n_p r_c/r_t) \geq d_{max}, \tag{5}$$

which yields

$$p_{max} \geq \frac{d_{max}}{1 - n_p r_c/r_t}. \tag{6}$$

This gives a formula for a lower bound on the maximum reading period length based only on the particulars of the application (i.e., the properties of the storage sub-system, the desired number of streams, and their bandwidth requirements). Note that to perform this simplification, $1 - n_p r_c/r_t$ must be positive:

$$r_t > r_c n_p. \tag{7}$$

We can replace p_{max} in equation (4) using equation (3) to obtain a formula for block size based only on the application particulars:

$$b \geq \frac{d_{max}}{1/r_c - n_p/r_t}. \tag{8}$$

Again, we must have $r_t > r_c n_p$. That is, the summed consumption rates cannot exceed the available bandwidth from the storage device; a reasonable requirement. In fact, as $r_c n_p$ approaches r_t, both the buffer space and reading period length will approach infinity (given $d_{max}>0$). So in the presence of non–zero delays, it is not feasible to utilize all available storage device bandwidth. Figure 3 shows an example of minimum block size versus bandwidth utilization. Note that to keep buffer space and delays reasonable, the utilization of the bandwidth must be approximately 75% or less of the total.

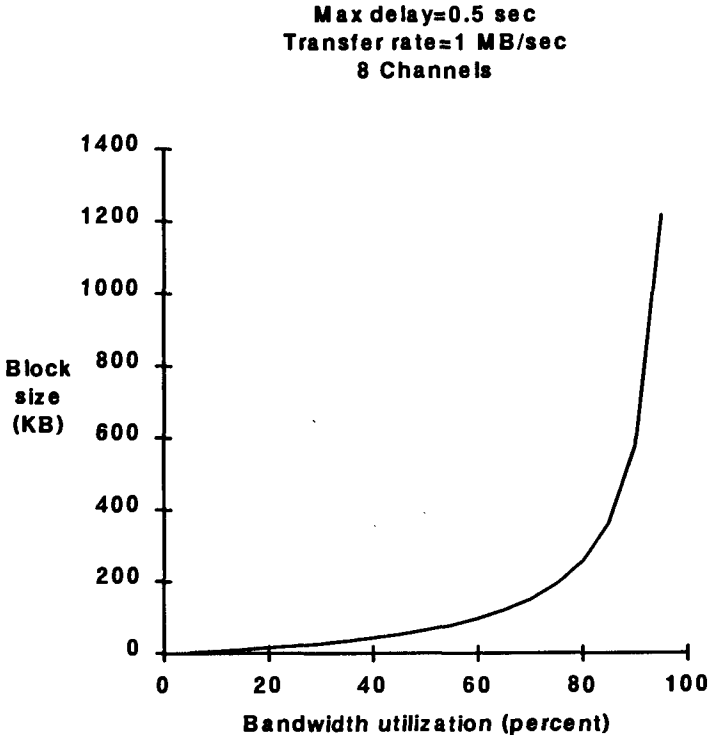


Figure 3: Block size vs. bandwidth

Observe that both minimum block size and reading period length are linearly proportional to the maximum delay length. As mentioned above, minimum block size is indicative of buffer requirements, while reading period length is indicative of the length of wait before playback can

begin. The delay term is thus an important parameter to minimize. As the delay term will normally be related to seeking latencies, we will later look at disk scheduling algorithms to reduce such latencies.

3.1.2 No Channels Synchronized

Above we have considered the case where streams are synchronized, with the same consumption rate. During playback, time was divided into reading periods, during which a fixed amount of data was read for each stream. A further assumption was that the data read for each stream in a reading period would take at least one reading period to consume – retrieval which is buffer-conserving.

In this section we will drop the assumption that all streams are synchronized and have the same rate. We will assume that streams may have different rates, and that playback for some streams may be paused while playback for the others continues. While a stream is paused it will not consume any data, and may have no free buffer space for reads to continue. After it resumes playing it may be out of step with the other streams by having a near full buffer, while those of the other streams are near empty. Therefore, the definition of reading periods must be relaxed so that the amount read for each stream is not fixed. Variable reading amounts may also be required when variable rate compression schemes are employed, or when different sampling rates have been used.

Our only assumptions are then:

- Starvation is never allowed to occur.
- Streams are serviced in reading periods.

- Streams must be able to be buffer-conserving in any given reading period (although they may not always be).

Disallowing starvation is necessary for meeting real time requirements. Reading periods and the capability of buffer-conservation, on the other hand, are design choices that we have made[15]. Buffer-conservation and non–starvation both have ramifications in terms of buffer space requirements. To avoid starvation, there must be enough buffered to satisfy consumption between each read. Let Δ_{max} be the maximum time difference between the completion of successive reads for a given stream. We will assume that data is not available until the read is complete.[16] If the consumption rate of the stream is r_c, then $r_c\Delta_{max}$ must be buffered to prevent starvation between reads.

The capability of buffer–conservation also puts a lower bound on buffer space, but it is less strict than that of non–starvation. Let p_{max} be the maximum length of a reading period. By definition, a read is done for each stream in each reading period, so $\Delta_{max} < 2p_{max}$ (note that it is strictly less than because a read takes non-zero time, preventing it from completing at the very start of a reading period). Also, it is impossible to guarantee a time difference between reads of less than a reading period so $\Delta_{max} \geq p_{max}$. In order to be buffer-conserving at least as much data must be read as is consumed. Since $r_c p_{max}$ may be consumed, streams capable of buffer-conservation must be able to read and buffer at least that much. Figure 4 shows the relationship between p_{max} and Δ_{max}.

---

[15] Our definition of reading periods is general enough that it is not restrictive; i.e., it is possible to formulate an algorithm which doesn't batch any requests together for servicing under our definition.

[16] Usually, when reading a physical block (sector) from a disk drive the block is transferred as an atomic unit; parity checks, error correction etc. must be performed after which the entire block is declared "ready". Therefore, none of the data may be consumed until the entire read is complete. This is also the normal handshaking process for multi–block reads – the process requesting the data is notified when it is all ready. Our approach may be readily adapted to withdraw this assumption, but at the cost of added complexity.

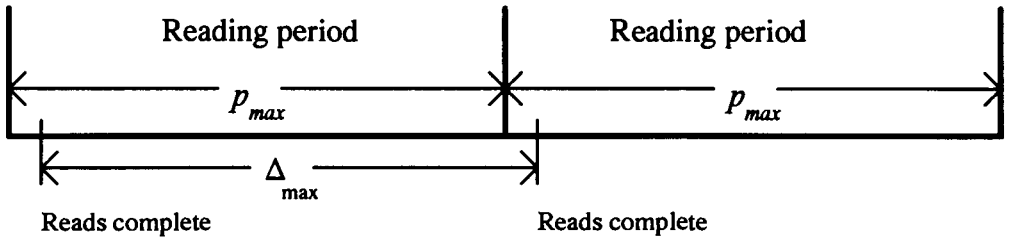


Figure 4: Maximum reading period length and delay between reads.

In addition to being necessary, the same amount of buffer space is sufficient. This can be demonstrated by the following simple algorithm. Let the buffer for each stream be of size $r_c\Delta_{max}$. Prior to initiating consumption, $r_c p_{max}$ is pre–fetched into the buffer, and the user must wait for the reading period to complete. In each successive reading period, a read of at most $r_c p_{max}$ is attempted, but is truncated before causing buffer overflow. Starvation would imply that a reading period began with less than $r_c p_{max}$ buffered. However, each reading period is either buffer-conserving, or else had a read truncated due to a full buffer. The first reading period began with $r_c p_{max}$ buffered, and a full buffer contains enough for a maximum delay, so starvation is impossible.

3.1.3 Handling non–contiguous files

We have stated above that variable sized reads are desirable for several reasons. If files are stored physically as contiguous units, this poses no problems. However, most file systems do not store files contiguously, but break them down into fixed sized ***clusters*** of contiguous physical blocks. With non–contiguous files, reads which correspond to more than one cluster may incur extra seek latencies.

There are two possible ways of dealing with this problem.

1. The extra seek time could simply be tolerated.

2. Extra buffer space could be allocated, cluster sizes fixed, and reading algorithms designed so as to avoid any extra seek latencies.

- Let r_{cmax} be the maximum consumption rate the system will allow, and let p_{max} be the maximum reading period length. Set the cluster size to $\chi = r_{cmax} p_{max}$, i.e., enough data to last a maximum consumption rate stream throughout the duration of a maximum length reading period.
- For each stream, let $\beta = \chi \lfloor \chi / r_c p_{max} \rfloor^{-1}$, i.e., β is the least value which evenly divides a cluster, and also is at least enough data to ensure buffer-conservation ($r_c p_{max}$).
- Allocate a buffer of size $r_c \Delta_{max} + \beta$.
- Pre–fetch β before allowing consumption to begin on a stream.
- For each stream in each reading period, attempt to read β from the file. If there is enough free buffer space to do so, perform the read. If there is not enough free buffer space, read nothing.

Table 2: An algorithm to avoid seeks.

For example, an algorithm that avoids seeks is given in Table 2. In the algorithm, the value of β is chosen so that it evenly divides the cluster size. In each reading period, either the amount β is read, or else nothing is read. Therefore, no reads cross a cluster boundary and require an extra seek. When there is not enough buffer space to read β in the current reading period then there must be at least $r_c \Delta_{max}$ buffered already. This means there is enough data buffered to last until a read from the following reading period completes, ensuring that starvation cannot occur.

By attempting to read β each period rather than $r_c p_{max}$ we tend to increase the values of p_{max} and Δ_{max} since the amount of data being retrieved is increased. However, the extra seeks are avoided, leading to a net decrease in the values of p_{max} and Δ_{max}. Observe also that in actual implementation a value of β must correspond to an integral number of physical blocks, so some further increase in its size may be necessary. This is also true of previously stated algorithms which read arbitrary amounts. All reads and buffer sizes must be rounded up to correspond to physical block sizes for actual implementation.

Our definition of β uses the value of p_{max}, but p_{max} of course depends on β (the more read, the more time required for reading). The β formula cannot be simplified to remove this circularity because of the floor function in it. Therefore the computation of β must be iterative. However, there is no such problem with χ, which may be computed in a straightforward way. As a first approximation, one could use β=χ to determine p_{max}. Then this value of p_{max} could be used to obtain more accurate values for β for each stream involved in the maximum reading period length scenario (maximum system loading). The process can be repeated iteratively until the values converge (or cycle, if they do not converge on a single value).

3.2 Disk Scheduling

In retrieving data from a disk, there are four factors which contribute to the latency:

(i) Seek latencies, that is, the time required to position the read head over the track containing the desired data.

(ii) Rotational latencies, that is, the time it takes for the start of the data to rotate underneath the head so that the transfer can begin.

(iii) The transfer rate of the drive, that is, the rate at which data is transferred from the drive once the head is in position over the data.

(iv) The time during a read required to cross track or cylinder boundaries (if the block of data being read does cross such a boundary).

The transfer rate is fixed for a particular drive. However, it is possible to improve transfer rates by using a disk array, i.e., multiple disks transferring data in parallel [46]. This yields a large logical drive composed of several drives in parallel.

Delays arising from crossing track or cylinder boundaries may be avoided by careful placement of data during storage. As long as no block of data straddles the boundary of a track or cylinder such delays will not arise. If it is impossible to avoid such situations, a worst case delay time may be added to the latency estimate. We will assume for now that data has been stored so as to avoid these delays. For a discussion of delay sensitive data placement, see our previous work [38].

Spencer Ng [75] has studied strategies for reducing rotational delays, such as storing multiple copies of data on a track and using multiple actuators. His strategies reduce the worst case rotational delay, and may be used in conjunction with our methods. However, they are not applicable to most off–the–shelf hardware. Abbot [1] has done some work with calculated rotation times, but provides improvements in latency only with a given probability; no guarantees for improved latency are given. It is possible to eliminate rotational latencies when blocks occupy entire tracks of data. In this case reading can begin as soon as the head arrives over the track without any rotational latency, as all the data on the track is desired. Many drive controllers now support buffer management so that this is performed transparently. Because track sizes can be quite large, we will assume in this thesis that rotational latencies may be incurred, and, furthermore, that they may take on any value between zero and some maximum.

Some drives vary the rotational speed and/or the transfer rate, depending on which track the head is over. This allows greater storage densities and/or increased transfers rates. For simplicity we will use only a single value for rotational delay and transfer rate for a drive (for the drives which vary these parameters, the worst case values may be used). The possible exploitation of variable rotational latencies and transfer rates is a good area for future research.

To reduce seek latencies, it is common to schedule disk requests so that head movement is minimized. Section 3.2.1 reviews work by others on disk scheduling. In section 3.2.2, we describe

our contribution to disk scheduling for CM servers: the sorting set approach. Section 0 describes how to get optimal performance out of the sorting set approach, by performing "pre-seeks".

3.2.1 Previous Approaches

Traditionally, disk scheduling algorithms (e.g., first come first served (FCFS), shortest seek time first (SSTF), SCAN, etc.) have been employed by servers to reduce the seek time and rotational latency, to achieve a high throughput, and to provide fair access to each client [33, 119]. The addition of real–time constraints, however, make direct application of traditional disk scheduling algorithms inappropriate for multimedia servers.

Techniques for scheduling real–time tasks have also been extensively studied in the literature (e.g., [56, 63, 84, 139]). The best known algorithm for real–time scheduling of tasks with deadlines is the ***Earliest Deadline First*** (EDF) algorithm. In this algorithm, after accessing a media block from disk, the media block with the earliest deadline is scheduled for retrieval. Scheduling of the disk head based solely on the EDF policy, however, may yield excessive seek time and rotational latency, and hence, may lead to poor utilization of the server resources.

Several combinations of conventional disk scheduling algorithms and real–time scheduling techniques have been investigated in the recent past. The simplest of all such techniques is the ***round–robin*** scheduling algorithm, in which the order in which clients are serviced does not vary from one reading period to another. However, the major drawback of round robin scheduling is that it, like EDF, does not exploit the relative positions of the media blocks being retrieved during a reading period [10,11,102]. For this reason, data placement algorithms which inherently reduce latencies are sometime used in conjunction with reading period–robin (e.g. contiguous or constrained placement – see section 4.1.1).

To address the limitations of the reading period–robin scheduling algorithm, we have adapted the SCAN disk scheduling algorithm for multimedia servers [35,38,37]. SCAN operates by "scanning'' the disk head back and forth across the surface of the disk, retrieving a requested block as the head passes over it. If SCAN scheduling is utilized during each reading period of retrieval, seek latencies can be minimized. One variant of this basic algorithm combines SCAN with EDF, and is referred to as the SCAN–EDF scheduling algorithm [109]. As per the EDF algorithm, the request with the earliest deadline is served first. However, if several requests have the same deadline, then their respective blocks are accessed using the SCAN disk scheduling algorithm. Clearly, the effectiveness of the SCAN–EDF technique is dependent on how many requests have the same deadline. If stream playback is initiated only at reading period boundaries, then all deadlines will be batched at the end of reading periods, and SCAN–EDF effectively reduces to SCAN.

We present another variant of SCAN in this chapter: the pre–seeking SCAN algorithm. The pre–seeking SCAN algorithm utilizes knowledge of data to be played back in the next reading period to initiate a "pre–seek" in the current reading period [36]. However, such an algorithm would be inapplicable when non–CM data is also being retrieved.

Notice that in the case of the round–robin algorithm, since the order in which clients are serviced is fixed across reading periods, the maximum separation between the retrieval times of successive requests of a client is bounded by the duration of a reading period. However, in the case of SCAN, the relative order for servicing clients is based solely on the placement of blocks being retrieved, so it is possible for a client to receive service at the beginning of a reading period and then at the end of the next reading period (i.e., a separation of two reading periods between service). This difference has some implications in terms of playback initiation delay and buffer

requirements. For round–robin, it is possible to initiate playback immediately after all blocks from the first request have been retrieved. With SCAN, however, playback must wait until the end of the reading period. For buffer space, it is clear that to prevent starvation round–robin needs enough buffer space to satisfy consumption during one reading period, while SCAN needs enough buffer space to satisfy consumption during two reading periods. However, SCAN's reading periods will be shorter, so there is a trade–off between reading period length and the number of reading periods between successive service (see Figure 5). To exploit this trade-off, we have developed a generalization using ***sorting sets***, which we describe in the following sections. Concurrently, Yu et al., have developed the *Grouped Sweeping Scheme* (GSS), which is functionally equivalent to our sorting sets [138].

As a final caveat, we would like to mention that tailoring disk scheduling algorithms to service real–time traffic may yield large response times for all the non real–time requests. To address this limitation, Reddy and Wyllie have proposed a modification of SCAN–EDF, which enables the server to service non real–time requests immediately after the next real–time request [109][17]. With the sorting–set scheme, on the other hand, one or more sorting sets may be dedicated to non real–time requests. Furthermore, it is also possible to balance the load among sorting–sets to improve response time [138].

3.2.2 Sorting–set Scheduling

Because most mass storage systems utilize disks, it may be desirable to perform some sorting of the blocks to be read in a reading period in order to reduce disk latencies. Reducing latencies may

[17] Reddy and Wyllie make it sound as if SCAN–EDF is superior to SCAN in terms of non real–time request response time. However, their version of SCAN–EDF includes this modification, while their version of SCAN does not.

lead to a shorter reading period, with smaller buffer requirements. For this purpose, streams may be assigned to a ***sorting set***. We define each sorting set to be a set of n streams $\{s_1, s_2, \ldots s_n\}$. Let the sorting sets be $S_1, S_2, \ldots S_q$. The sets are always executed in fixed sequence, i.e., $S_1, S_2, S_3 \ldots$ However, within each set the reads may be ordered to reduce overall seek time. At one extreme, there is only one sorting set and the optimization for seek time is performed over all the requests. At the other extreme each set contains only one stream, and the order of reads in each reading period remains fixed with no seek optimization being done.

The use of sorting sets affects buffer requirements in an interesting way, due to the requirement of non–starvation. Consider the two extremes mentioned above. When there is only one sorting set, it is possible that the read(s) for a stream may be performed first in one reading period, and last in the next. Therefore the time between the reads is roughly the length of two reading periods. When there is a set for each stream (i.e., round-robin) the time between the reads is at most one reading period. Thus, the amount that must be read and buffered in the first case corresponds to two periods, but only one in the second case. This does not mean, however, that fixed ordering is always superior. With fixed ordering the reading period may be longer since no optimization for seek latencies can be performed (see Figure 5).

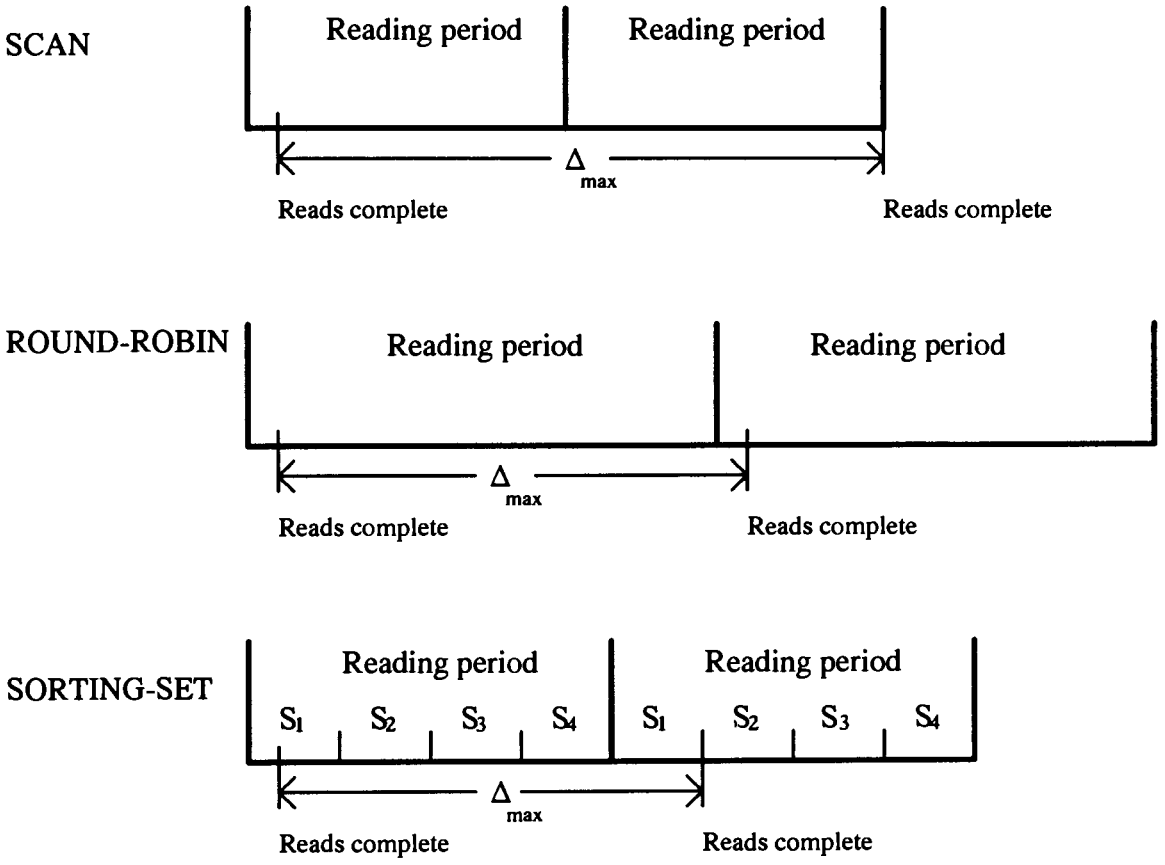


Figure 5: Worst case delay between reads.

To be more precise, let the maximum time to execute the reads of a sorting set S_j, including seek latencies and all other overhead, be $T(S_j)$. For a particular stream, s, the worst case (largest) Δ_{max} will occur if all the reads for s are performed first in S_j in one reading period, then last in the next (see Figure 5). The time between the reads will therefore be the time remaining in S_j after s is completed (which in the worst case is all the other streams in S_j to be read), plus the time for all the other sorting sets, plus the time for all of S_j (in the second reading period). That is,

$$\Delta_{max} = \sum_i T(S_i) + T(S_j - \{s\}) \tag{9}$$

Previously we have mentioned that the reading period length is indicative of start latencies (the time between requesting playback and having it begin). We can now be more precise regarding

the worst case start latency. In the worst case, a request for playback will arrive such that it just misses the only occasion in the current reading period that a read request for it could be scheduled. Therefore, the read will not occur until the next reading period. This is a delay of Δ_{max} – the maximum time between two successive reads for a given stream. Note that the delay will not be more than Δ_{max}, because Δ_{max} is calculated based on the second read in the worst case situation being at the very end of a sorting set, and that is when playback can begin with the buffer-conservation attempting algorithms we are discussing. Therefore, the worst case start latency is Δ_{max}.

3.2.3 The Pre–seeking SCAN Algorithm

As we have mentioned, the SCAN algorithm sweeps the disk head back and forth across all cylinders, servicing requests as the head comes over the desired block of data. A variant of SCAN, called N–SCAN guarantees service to the first N requests before servicing any new requests. For each group of N requests, N–SCAN considers the request which is nearest the inside track and the request which is nearest the outside track. It will select from these two the one which is closer to the current position and begin moving toward it. After it has serviced all requests in that direction, it will reverse direction to service any remaining requests. Both SCAN and N–SCAN seem likely candidates for application in the reading periods of a sorting-set scheme.

A sorting set may require reads on the innermost and outermost tracks of the disk. If a maximum (inside to outside) seek consists of m_{max} tracks, then any disk scheduling algorithm may be required to move the disk head by as much as m_{max} tracks in a sorting set. Applying the N–SCAN algorithm to sorting sets, the worst case delay would be encountered when the disk head is left in the middle track of the drive at the end of one sorting set, and in the next sorting set must visit

both the innermost and outermost tracks. This would mean the head would move over $1.5m_{max}$ tracks (see Figure 6). Applying the SCAN algorithm, the head would only move over m_{max} tracks, as a sweep would be done for each sorting set. However, an extra seek may be required to move the disk head to the edge of the disk after the last request in the sorting set has been performed. Therefore, the choice between SCAN and N–SCAN is a trade–off between total seek distance and the number of seeks.

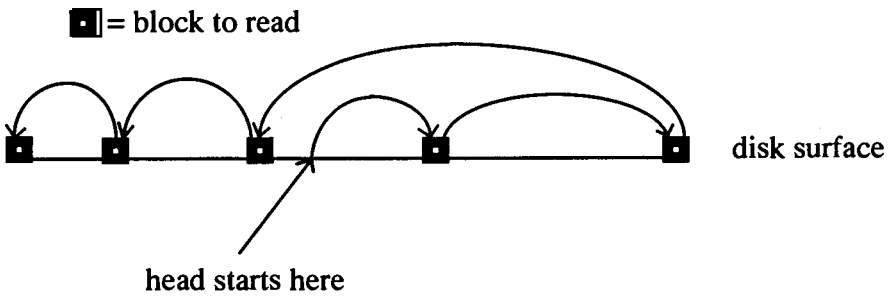


Figure 6: Worst case for the N-SCAN algorithm.

We now modify the N–SCAN algorithm as follows. First, consider the two blocks to be read in the next sorting set which are closest to each extreme edge of the disk. At the end of the current sorting set we will perform a seek to whichever of these is closest. We will call this the ***pre–seek***. Note that normally disk seeks are accomplished via read commands, which implicitly require the disk head to be moved. However, the pre–seek must be an explicit seek command. The read cannot be issued until the next sorting set begins and buffer space is guaranteed to be free. The modified algorithm is shown in Table 3. We will refer to this algorithm as the ***pre–seeking SCAN algorithm***, as it causes the disk head to move in a sweeping motion back and forth across the disk

(but not necessarily end–to–end, as in the SCAN algorithm). Figure 7 shows the how the pre–seeking SCAN algorithm moves the disk head.[18]

THE PRE–SEEKING SCAN ALGORITHM
• In each sorting set, sort the blocks to be read according to location.
• Prior to the first sorting set, seek to the head of its list.
• During each sorting set, a seek will already have been previously executed to either the head or the tail of the list. Perform the reads in the list, either in sorted order or in reverse sorted order; in sorted order if the seek was to the head, in reverse sorted order if the seek was to the tail.
• After the last read in a sorting set, consider the sorted list of reads for the next. Initiate a seek (the "pre–seek") to either the head or tail of the list; whichever is closer to the current head position.

Table 3: The pre-seeking SCAN algorithm

From the point of view of a conventional system the pre–seek is not very significant. It does change the order of reads, but not in a manner which would alter the performance of the N–SCAN algorithm. If a sorting set requires a full sweep of the disk to service its requests, and the pre–seek moves the head to the middle of the disk, we would again have a total distance of $1.5m_{max}$

[18] We have also discussed the pre–seeking SCAN algorithm in [36].

tracks, only now with an extra seek, so the performance for a given sorting set would appear to be worse.

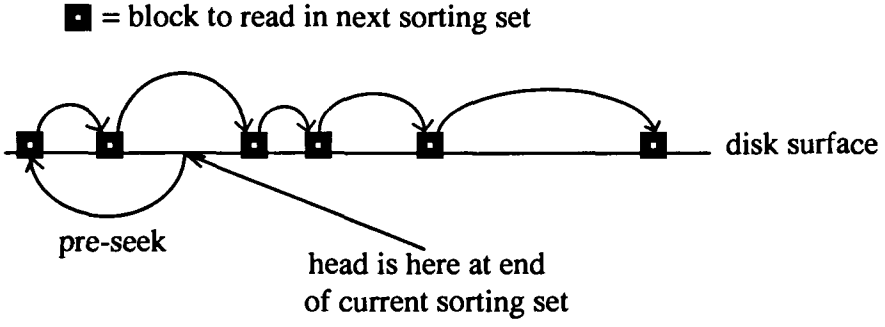


Figure 7: Disk head movement for the pre-seeking SCAN algorithm.

However, the fact that the pre–seek is an explicit seek rather than an implicit seek resulting from read means that it does not necessarily belong to a particular sorting set. It may be assigned to the current sorting set, or the next sorting set. The time it takes may actually be divided between the two sorting sets. By careful allocation of the pre–seek's time the pre–seeking algorithm can achieve optimal performance, i.e., a single sweep with one seek for each service request in the worst case. To prove this, we assume that seeking time depends only on the number of seeks and the total distance the head moves (a later section on seek time estimates shows this is in fact the case).

Claim: The pre–seeking SCAN algorithm is optimal.

Proof: Consider reading period i, which initiates the pre–seek, and reading period $i+1$, which follows it. We allocate the time for the pre–seek as follows. The time for initiating the pre–seek is allocated to reading period i. Thus, a reading period with j

blocks is allocated the time for j seek initiations: none for the first block, one for each of the rest, and one for the pre–seek. The time resulting from the distance of the pre–seek is divided. Reading period i is allocated as much of this distance as it can take without its total distance allocation exceeding a single sweep of the disk (m_{max} tracks). The remainder is allocated to reading period i+1.

Without loss of generality, assume that in reading period i the disk head moves to the right. Suppose the pre–seek is also to the right (see Figure 8 (i)). As the head has not changed direction, the total movement including the pre–seek cannot be more than a single sweep of the disk. Therefore, in this case, we can assign all the time for the pre–seek to reading period i. Suppose now that the pre–seek is to the left. We then consider two possibilities for the following reading period:

The head moves to the left in reading period i+1 (see Figure 8 (ii)). In this case the pre–seek is in the same direction as all the head movement in reading period i+1. Therefore the pre–seek and seeks in reading period i+1 do not exceed a single sweep of the disk. Thus, the distance of the pre–seek can be allocated to reading period i+1.

The head moves to the right in reading period i+1 (see Figure 8 (ii)). In this case the head pre–seeks to the left, then sweeps to the right. This can only happen when the furthest request to the left is nearer to the current position than the furthest request to the right. At most, the furthest request to the right is at the edge of the disk. Therefore, the length of the pre–seek (which is to the furthest point left) is less than

to the right edge of the disk. The distance portion of the pre–seek can therefore be included in reading period i without the total distance exceeding a single sweep.

Thus, each reading period is allocated time to initiate N seeks, and to travel over at most one sweep of the disk. This is the lower bound for any algorithm, therefore the pre–seeking SCAN algorithm is optimal. Our assignment of time relies on seek time being a result of only the number of seeks and the distance. This is established in the next section which provides an estimate of seek times.

■

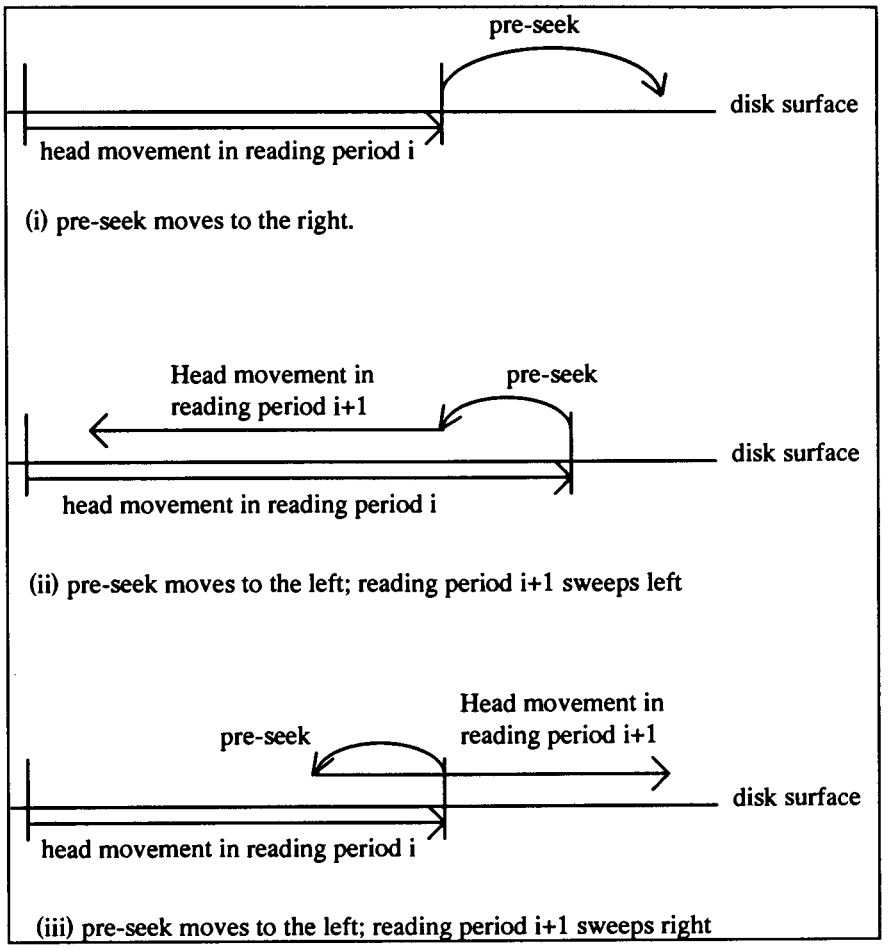


Figure 8: Cases for optimality proof.

The pre–seeking SCAN algorithm is an adaptation of the N–SCAN algorithm. As such, it is not particularly novel as a disk scheduling algorithm. However, in the context of delay sensitive retrieval it is significant, because the pre–seek allows flexibility in its time assignment. Furthermore, we have demonstrated that an optimal algorithm does exist – a fact which is not intuitively obvious. The technique of pre–seeking bears a superficial similarity to anticipatory disk arm movement suggested by King [57]. However, King's anticipatory seeks are performed only when the next request is *not* known, in contrast with the pre–seek, which is to the next known request. Additionally, the pre–seek is not simply performed to the next request according

to the SCAN or N–SCAN algorithms. Instead it is to either the innermost or outermost request for the next sorting set. As this does not change the distance which the disk head must travel it is an insignificant alteration according to conventional measurements, while to the delay sensitive retrieval problem it is pivotal in achieving optimal performance.

3.3 Minimum Read and Buffer Requirements

For contiguous files, we know that $r_c\Delta_{max}$ is sufficient buffer space, and at most $r_c p_{max}$ needs to be read in a single reading period. We have shown an algorithm that meets real time deadlines given non–contiguous files. We now prove lower bounds for reading and buffer space, given that only one seek is permitted for each stream in a single reading period. The proofs use an adversary argument. Meeting the real time deadlines will be considered a game in which the algorithm supplying the data is the player and the adversary controls the amount consumed, disk latencies, etc. The game is as follows:

- Time is divided into reading periods, and each reading period is divided into sorting sets. Without loss of generality, we assume that the algorithm's reads are performed in the first sorting set, so that the beginning of the sorting set is the beginning of the reading period.
- The algorithm chooses an initial amount to read, and signals that the adversary may begin at the start of the next reading period.

At the start of each reading period:

- The algorithm chooses some amount to read.
- The adversary then chooses an amount to consume.

- The adversary chooses when to allow read to happen: either immediately, or after $\Delta_{max}-p_{max}$ time (This corresponds to when the read is performed due to disk scheduling). Consumption begins immediately.

The following rules apply:

- The adversary can consume at most $r_c\ p_{max}$ in a reading period, and $r_c\ \Delta_{max}$ between reads (by allowing one read to occur immediately, then in the next reading period delaying the read for $\Delta_{max} - p_{max}$).
- The player can only incur a single seek in each reading period. That is, a read must be from a single cluster. Therefore the player's read is bounded by the amount remaining in the current cluster.
- The adversary wins if starvation occurs.

From the rules of the game, we derive the following lemmas:

Lemma 1: The game will be lost if there is ever less than $r_c\ \Delta_{max} - r_c p_{max}$ buffered at the start of a reading period.

Proof: If there is ever less than $r_c\Delta_{max}-r_c\ p_{max}$ buffered at the start of a reading period, then the adversary can delay the next read by $\Delta_{max}-p_{max}$, and then consume all the buffered data.

■

Lemma 2: If $r_c\Delta_{max}-x$ is buffered at the start of a reading period, then the algorithm must read at least x or else the game will be lost.

Proof: Suppose $r_c\Delta_{max}-x$ is buffered at the start of a reading period, and the algorithm reads less than x. Then the adversary can cause a delay between that read and the next of Δ_{max}, and consume $r_c\Delta_{max}$. Because less than x was read, there will not be enough data in the buffer to prevent starvation during this time.

■

Lemma 3: The size of a cluster must be at least $r_c\, p_{max}$.

Proof: Suppose the cluster size is less than $r_c\, p_{max}$. Let the adversary consume $r_c\, p_{max}$ in each reading period. Since the algorithm can perform at most one seek per stream in a reading period, the most it can read is an entire cluster. Therefore in every reading period it must read less than is consumed, which must result in starvation and loss of the game.

■

Lemma 4: If buffer space is less than $r_c(\Delta_{max}+p_{max})-1$, then the adversary can arrange to reach a point where at the beginning of a reading period there are $r_c\Delta_{max} - 1$ buffered, and $r_c\, p_{max}+h$ remains in the current cluster, where $0 \le h < r_c\, p_{max}$.

Proof: By lemma 2, we know that the amount buffered at the start of a reading period, plus the amount read, must be at least $r_c\Delta_{max}$. At most it will be $r_c(\Delta_{max}+p_{max})-2$ (the maximum buffer space). The adversary can read any amount from 0 to $r_c\, p_{max}$. Therefore, it can always read the required amount to reduce the buffered data to $r_c\Delta_{max} - 1$ for the next reading period. It can then repeat this operation, keeping the buffered data at $r_c\Delta_{max} - 1$ for each successive reading period. Note that in each

reading period, the algorithm must read at least 1 (by lemma 2), and can read at most $r_c\, p_{max}$–1 (otherwise the buffer would overflow).

Now suppose that the adversary does use the above strategy, and the amount remaining in the cluster is never $r_c\, p_{max}+h$, with $0 \le h < r_c\, p_{max}$. Lemma 3 tells us that the cluster size is at least $r_c\, p_{max}$, so at the very least it must go from $2r_c\, p_{max}$ to $r_c\, p_{max}$–1 in a single reading period to accomplish this. However, this implies a read of at least $r_c\, p_{max}$+1, and we have seen that the adversary will limit the reads in each reading period to at most $r_c\, p_{max}$–1. Therefore the amount remaining in a cluster must reach a point of $r_c\, p_{max}+h$, with $0 \le h < r_c\, p_{max}$.

■

Claim 1: An algorithm must have buffer space of at least $r_c(\Delta_{max}+p_{max})$–1 to prevent losing the game.

Proof: Assume the contrary, that the algorithm only has $r_c(\Delta_{max}+p_{max}) - 2$ buffer space.

First, we invoke lemma 4, and assume that there is $r_c\Delta_{max} - 1$ buffered, and $r_c\, p_{max}+h$ remains in the current cluster, where $0 \le h < r_c\, p_{max}$. If the next read is of an amount less than h, then let the adversary also consume h, and we again have the same amount buffered, and a new, smaller, h. At some point, the algorithm must read more than h (certainly when $h=0$). Let the amount it reads be $h+z$, *with* $z>0$.

Some additional notation is now necessary. Let the reading period in which this read of $h+z$ occurs be numbered 0, the next 1, etc. Let the amount buffered at the start of reading period i be b_i, the amount read by the algorithm ϱ_i, the amount consumed by the adversary c_i, and the amount of data remaining in the cluster k_i.

Once the algorithm reads $h+z$, let the adversary consume z. In the next reading period, reading period 1, let the adversary consume $r_c\, p_{max}$. In reading period 2, let the algorithm also consume $r_c\, p_{max}$.

Reading period	Amount buffered at start of reading period	Amount remaining in cluster at start of reading period	Amount read by algorithm	Amount consumed by adversary
0	$b_0=r_c\Delta_{max}-1$	$k_0=r_c\, p_{max}+h$	$\varrho_0=h+z$	$c_0=h$
1	$b_1=b_0+\varrho_0-c_0$ $=r_c\Delta_{max}-1+z$	$k_1=k_0-\varrho_0$ $=r_c\, p_{max}-z$	ϱ_1	$c_1=r_c\, p_{max}$
2	$b_2=b_1+\varrho_1-c_1$ $=r_c\Delta_{max}-1+z+\varrho_1-r_c\, p_{max}$	$k_2=k_1-\varrho_1$ $=r_c\, p_{max}-z-\varrho_1$	$\varrho_2\le k_2$	$c_2=r_c\, p_{max}$
3	$b_3=b_2+\varrho_2-c_2$ $=r_c\Delta_{max}-1+z+\varrho_1-2r_c\, p_{max}+\varrho_2$ $\le r_c\Delta_{max}-1-r_c\, p_{max}$			

Table 4: How the adversary wins

Table 4 shows the values of b_i, k_i, ϱ_i, and c_i, in reading periods 0 through 3. In any reading period, if $b_i+\varrho_i > r_c(\Delta_{max}+p_{max})-2$, then the adversary could simply consume 0, and allow the buffer to overflow. Therefore

$$b_i+r_i \le r_c(\Delta_{max}+p_{max})-2. \tag{10}$$

Applying this to reading period 1 we obtain

$$r_c\Delta_{max}-1+z+\varrho_1 \le r_c(\Delta_{max}+p_{max})-2, \tag{11}$$

or

$$r_c\, p_{max}-z-\varrho_1 \ge 1. \tag{12}$$

Comparing this with k_2, we see that $k_2 \geq 1$. This means that the algorithm cannot complete reading the cluster before reading period 2. The earliest it could complete reading the cluster would be in reading period 2 with $k_2=\varrho_2$. Even with $k_2=\varrho_2$, we obtain $b_3=r_c\Delta_{max}-1-r_c p_{max}$, which by lemma 1 must lead to the algorithm losing the game.

■

Intuitively, the proof is using the fact that with $r_c\Delta_{max}-1$ buffered, the algorithm must read some amount, but this amount must be less than $r_c p_{max}$ to avoid buffer overflow. This allows the adversary to force the algorithm into a position with less than $r_c p_{max}$ remaining in the cluster. An appropriate choice for consumption amount by the adversary then makes sure that there is enough room in the buffer for all but one byte remaining in the cluster. The adversary consumes $r_c p_{max}$ during each of the next two reading periods, while the algorithm must take both reading periods just to finish off reading the cluster – an amount less than $r_c p_{max}$. This is enough to force the algorithm into a losing position.

Claim 2: Suppose the cluster size is χ. Then any algorithm must be able to read at least β, defined as:

$$\beta = \left\lfloor \frac{\chi}{\frac{\chi}{r_c p_{max}}} \right\rfloor \tag{13}$$

in each reading period to prevent losing the game.

Proof: Suppose that the algorithm must read less than β in each reading period. Let $n = \lfloor \chi / r_c p_{max} \rfloor$. Because the algorithm reads less than β in each reading period, it will take at least n+1 reading periods for the algorithm to read each cluster. During each of these reading periods

the adversary can consume $r_c\, p_{max}$, for a total of $(n+1)r_c\, p_{max}$. By the definition of n, we have $\chi < (n+1)\, r_c\, p_{max}$. Therefore, more is consumed than read during the reading of each cluster, which must lead to starvation, and losing the game.

■

***Claim* 3:** Using a buffer of size $r_c(\Delta_{max}+p_{max})-1$, and with reads of at most $\beta = \chi \lfloor \chi / r_c\, p_{max} \rfloor^{-1}$ in each reading period is sufficient to guarantee that the algorithm will not lose the game. (Note: The cluster size must be at least $\chi \geq r_c\, p_{max}$, by lemma 3).

***Proof*:** Let the algorithm be greedy, that is, it always reads as much as possible in each reading period. The amount it reads is limited by one of three factors:

(i) It can read at most β.

(ii) It cannot perform a read that may lead to a buffer overflow

(iii) It cannot read more than what is remaining in the cluster.

Suppose that the algorithm fails. Failure means starvation. We have shown above that if there is ever less than $r_c\Delta_{max}-r_c\, p_{max}$ buffered at the start of a reading period that the adversary can bring on starvation (lemma 1). Furthermore, this condition must occur sometime prior to starvation (it is in fact satisfied by starvation).

Consider, then, the first reading period which ends with less than $r_c\Delta_{max}-r_c\, p_{max}$ buffered. During this reading period there must have been a net decrease in buffer space, or else the previous reading period would be the first with less than $r_c\Delta_{max}-r_c\, p_{max}$ buffered. By a net decrease, we mean that more was consumed than read. Note

that $\beta \geq r_c p_{max}$, and the adversary can read at most $r_c p_{max}$, so in this reading period it is clear that reading was not limited by factor (i).

Suppose instead that reading was limited by factor (ii). This means that if the adversary consumed 0, we would have a full buffer. Under this condition if the adversary consumes its maximum, $r_c p_{max}$, then the reading period ends with $r_c p_{max}$ less than a full buffer of data, that is, $r_c \Delta_{max} - 1$. Clearly this is not less than $r_c \Delta_{max} - r_c p_{max}$, so it is impossible that reading was limited by factor (ii).

The only remaining possibility is that reading was limited by factor (iii), that is, the amount remaining in the cluster. Note that an amount of 0 is equivalent to being at the next cluster. However, the next cluster must contain at least β, which would mean condition (i) holds. We have already shown this is not possible, so it must be the case that the amount remaining in the cluster is at least one. The net decrease in the reading period is therefore, at most, $r_c p_{max} - 1$ (maximum consumption by the adversary less a read of one).

We have shown that the reading period which leads to buffer space satisfying the condition of lemma 1 must be one which reads the last portion of some cluster. Consider now all the reading periods which perform reads from this cluster. Suppose there are m such reading periods. As before, let us denote the amount buffered at the start of reading period i as b_i, the amount read by the algorithm ϱ_i, the amount consumed by the adversary c_i, and the amount of data remaining in the cluster k_i.

For all but the last reading period, reading must be limited by conditions (i) or (ii), that is either by β or by the amount of free buffer space. The amount of free buffer space must be at least what was consumed in the previous reading period. Furthermore, the amount consumed in the previous reading period is at most $r_c\, p_{max}$, which is never greater than β. Therefore, a greedy algorithm will always read at least as much as was consumed in the previous reading period. That is,

$$r_{k+1} - c_k \geq 0 \tag{14}$$

for $k < m - 1$. For any $k < m$, the final buffer space may be derived as

$$\begin{aligned} b_{m+1} &= b_k + \sum_{i=k}^{m} r_i - \sum_{i=k}^{m} c_i \\ &= b_k + r_k - c_{m-1} - c_m + r_m + \sum_{i=k}^{m-2} (r_{i+1} - c_i) \end{aligned} \tag{15}$$

Applying equation (14) yields

$$b_{m+1} \geq b_k + r_k - c_{m-1} - c_m + r_m. \tag{16}$$

In reading period k if condition (ii) holds then

$$b_k + r_k = r_c(\Delta_{max} + p_{max}) - 1. \tag{17}$$

Substituting this in equation (16) yields

$$b_{m+1} \geq r_c(\Delta_{max} + p_{max}) - 1 - c_{m-1} - c_m + r_m \tag{18}$$

We know that each $c_i \leq r_c\, p_{max}$, and $r_m \geq 1$, so

$$\begin{aligned} b_{m+1} &\geq r_c(\Delta_{max} + p_{max}) - 1 - r_c\, p_{max} - r_c\, p_{max} + 1 \\ &= r_c(\Delta_{max} + p_{max}) \end{aligned} \tag{19}$$

which contradicts our assumption that $b_{m+1} < r_c\Delta_{max} - r_c\, p_{max}$. Therefore condition (ii) cannot limit reading in any reading period prior to reading period m. We already

know that condition (iii) only applies to reading period m. Therefore condition (i) applies to all previous reading periods, that is, $r_k = \beta$ for $k < m$. β is defined to evenly divide a cluster. Therefore, if each previous reading period reads β, there must be a multiple of β remaining in the cluster for the last read. We know the last read is at least one, so it must in fact be β. But if the last read is of β then condition (i) holds to the last reading period, which we have already shown to be impossible.

Therefore, it is impossible for the greedy algorithm to fail.

■

The above proof shows that the greedy algorithm can achieve the lower bounds for buffer space and for reading amount. The algorithm presented in the previous section obtains the lower bound for reading, but may require more buffer space. The amount of extra space that it will require corresponds to rounding up to a value that evenly divides a cluster. The lower bound on buffer space is $r_c\Delta_{max}+r_c\, p_{max} - 1$. If $r_c\, p_{max}$ does not evenly divide a cluster, then the algorithm will round up to the nearest value that does, β, and use $r_c\Delta_{max}+ \beta$ instead. This may mean rounding up from 0.4 of a cluster to 0.5 of a cluster, or, in the worst case from 0.51 of a cluster to a whole cluster. How significant this increase is depends on the particular application. While the algorithm of the previous section may require this additional buffer space, it always reads the same amount, when it does read. This makes its implementation extremely simple, especially in the area of buffer management. While the greedy algorithm is not complex in conventional terms, handling variable read amounts is complex enough to cause some concern where very tight deadlines are required. For example, the scheduling of reads and admission of any non real time read requests may be sandwiched in between reading periods, with the requirement to take negligible time to execute. The system designer must consider the trade off of complexity (time)

and buffer space in selecting between a greedy algorithm or the simple algorithm of the previous section.

We have shown, then, that non–contiguous file layouts come at the cost of additional seeks, or different pre–fetching strategies. In the next section we consider how to minimize the impact of the seeks that cannot be avoided.

3.4 Estimating Seek Times

We have shown how to calculate block size requirements and reading period length, and how to minimize the seek delays to keep them reasonable. However, to make use of our formulas describing buffer requirements and reading period length we need to be able to deduce what the time delays will actually be. In this section, we consider the seek time. In the literature, seek times have been approximated by a linear function of the number of tracks the head moves over [33,119]. The approximation simply uses the minimum seek and maximum seek to define a linear seek time function. Given the minimum (track–to–track) seek time, t_{min}, the maximum seek time, t_{max}, the number of tracks to seek, m, and the number of tracks corresponding to a maximum seek, m_{max}, the seek time may be approximated by

$$L_s(m) = t_{min} + (m-1)\frac{t_{max} - t_{min}}{m_{max} - 1} \tag{20}$$

for a non–zero seek (see Figure 9).

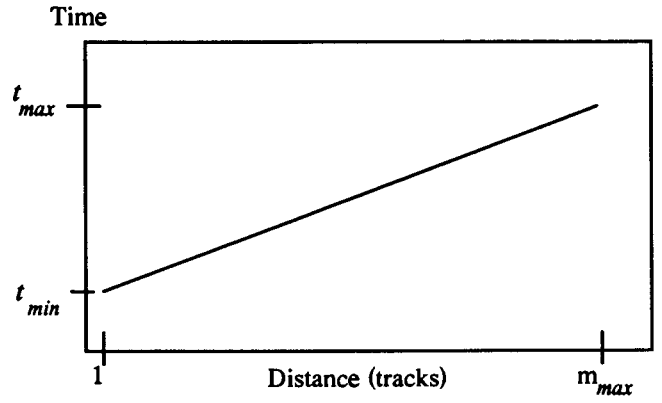


Figure 9: Seek time vs length of seek.

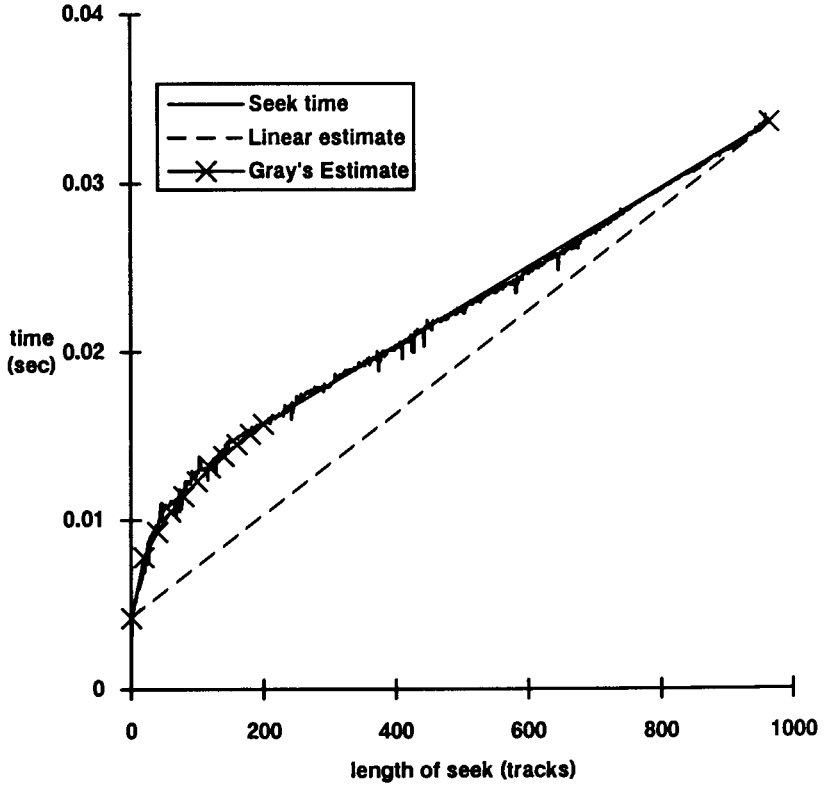


Figure 10: Seek time vs length of seek for Control Data WREN III 94161 (experimentally derived).

A more accurate estimate of seeks has been presented by Gray et al., [46] and Ruemmler et al., [112]. For seeks of less than a certain fraction of the disk surface they use a square root function rather than a linear function to estimate seek times. This reflects the fact that for short seeks the disk head its always accelerating or decelerating. For example, Gray et al., model the seek time as

$$\begin{cases} t_{min} + k_1\sqrt{m-1} & m \le 0.2m_{max} \\ k_2 + k_3 m & m \ge 0.2m_{max} \end{cases} \quad (21)$$

where k_1,k_2, and k_3 are drive dependent. Figure 10 shows the linear model and the square root model compared with the observed seek latencies of a particular disk drive.

The pre–seeking SCAN algorithm is optimal for both the linear model or the square root model. However, we will continue to use the linear estimate for two reasons. First, the linear estimate is not that bad – by adding an error term, ε, of about 5 msec it actually becomes rather conservative (in order to simplify the equations we will refrain from using the error term until we come to the case studies). Second, and more important, minimum and maximum seek times are commonly published values, so the linear estimate can be made from drive specifications. The square root estimate, on the other hand, requires an additional measurement of the seek time for a 20% seek, which can only be derived by experiment. Thus, the linear estimate is much more useful for drive evaluation.

Consider, then, reading a set of blocks. Suppose that j blocks are read. If the i'th seek is over m_i tracks then the total latency will be:

$$\sum_{i=1}^{j} L_s(m_i) = \sum_{i=1}^{j} \left(t_{min} + (m_i - 1) \frac{t_{max} - t_{min}}{m_{max} - 1} + \varepsilon \right). \tag{22}$$

This simplifies to

$$\sum_{i=1}^{j} L_s(m_i) = j t_{min} + \frac{t_{max} - t_{min}}{m_{max} - 1} \left(\sum_{i=1}^{j} m_i - j \right) + j\varepsilon . \tag{23}$$

From this we observe that it is not the individual seek length that is important in reducing seek time, but only the number of seeks and the sum of the seek lengths, that is, the total seek distance. This was our assumption in the optimality proof for the pre–seeking SCAN algorithm. A reading period utilizing the pre–seeking sweep algorithm may move the head by as much as m_{max}, so the delay due to seeking may be as much as

$$\sum_{i=1}^{j} L_s(m_i) = j t_{min} + \frac{t_{max} - t_{min}}{m_{max} - 1} (m_{max} - j) + j\varepsilon \tag{24}$$

$$\sum_{i=1}^{j} L_s(m_i) = j t_{min} + \frac{t_{max} - t_{min}}{m_{max} - 1} (m_{max} - 1 + 1 - j) + j\varepsilon , \tag{25}$$

which reduces to

$$\sum_{i=1}^{j} L_s(m_i) = (j-1) t_{min} + t_{max} - (j-1) \frac{t_{max} - t_{min}}{m_{max} - 1} + j\varepsilon . \tag{26}$$

Note that the second to last term has $m_{max} - 1$ in the denominator. As m_{max} is usually on the order of 1000, this term has negligible effect unless j values are large (on the order of 100). Thus, a rough approximation of total seek time is the time for a maximum seek plus *(j-1)* minimum seeks. We will use the more exact formula, but this rough approximation may be useful for quickly determining the suitability of disk drives.

3.5 Case Study A

In the previous sections we constructed a theoretical perspective on multi–stream delay sensitive data retrieval. In this section we give examples of how the results may be used to evaluate system design choices.

We have seen that reading can be split into sorting sets for the purpose of reducing seek latencies. Since the pre–seeking SCAN algorithm is optimal, we will assume that it is used for this purpose within the sorting sets. We will consider examples each playing back 16 streams. In order to keep our examples short, we will assume all the streams have the same consumption rate, r_c. We will consider having only 1 sorting set (i.e., pure pre-seeking SCAN), 2 sorting sets of 8 streams each, 4 sorting sets of 4 streams each, 8 sorting sets of 2 streams each, and finally 16 sorting sets of 1 stream each (i.e., round-robin). We will see how the choice of handling non–contiguous files affects the results of each. To compare the methods, we fix all the values except r_c and observe the values of the consumption rate and buffer space required.

From our study of the pre–seeking SCAN algorithm, we are now prepared to define T(S), which is the time to perform the reads on a sorting set S. Suppose that the method given in section 2.1 is employed to ensure that only one seek is required for each stream. Then |S| seeks must be performed. We know from equation (26) that the seek latencies may be at most

$$(|S|-1)t_{min} + t_{max} - (|S|-1)\frac{t_{max} - t_{min}}{m_{max} - 1} + j\varepsilon \tag{27}$$

For each seek we allow our estimate to be inaccurate by as much as ε=5 msec. In addition, each seek may have rotation delay, which we will denote t_r. Finally, it will take some time to read the data. We will assume that no track or cylinder boundaries are crossed, and calculate the time to be proportional to the transfer rate of the disk drive, denoted r_t. We can then calculate T(S) as

$$T(S) = (|S|-1)t_{min} + t_{max} - (|S|-1)\frac{t_{max} - t_{min}}{m_{max} - 1} + |S|(t_r + \frac{\beta}{r_t} + \varepsilon) \quad (28)$$

Having T(S) allows us to then calculate Δ_{max}. Recall that the buffer space required is $r_c\Delta_{max}+\beta$. Because we are not fixing r_c we must calculate it as

$$r_c = \frac{\beta}{p_{max}}. \quad (29)$$

The reading period length, p_{max}, is calculated as

$$p_{max} = \sum T(S). \quad (30)$$

For the case where seeks are tolerated, we will assume that a cluster size has been chosen so that at most two seeks are required per read. The calculations are the same as above, with the exception that twice as many seeks are required, and the buffer space is $r_c\Delta_{max}$. The variable β in this case represents $r_c\, p_{max}$, the maximum amount read in each reading period. In all cases, β represents a lower bound on the size of a cluster, since in the case of one seek being allowed β must be read contiguously, and in the case of two seeks, only one cluster boundary can be crossed in any read of length β, regardless of starting position.

Seagate WREN 6 ST2383N		
t_{min}	Track to track seek	5 msec
t_{max}	Maximum seek	28 msec
t_r	Rotational delay	17 msec
r_t	Transfer rate	2 MB/sec
m_{max}	Maximum seek distance	1260 tracks

Table 5: WREN 6 ST2383N Performance

For all examples we will use the drive performance characteristics of the Seagate WREN 6 ST2383N (see Table 5). Using β values of 4 KB, 10 KB, 20 KB and 50 KB, we compute the maximum supportable consumption rate (per stream) in Table 6, the buffer space required (per stream) in Table 7, and the start latency (Δ_{max}) in Table 8. To compare the efficiency of the various approaches, we plot cluster size, start latency, and buffer size, versus maximum supportable consumption rate in Figure 11, Figure 12, and Figure 13, respectively. In each case lower values indicate greater efficiency.

		β=4 KB	β=10 KB	β=20 KB	β=50 KB
1 seek	1 set	8.2	18.7	32.7	59.1
	2 sets	7.9	18.0	31.5	57.5
	4 sets	7.2	16.6	29.4	54.6
	8 sets	6.2	14.4	25.8	49.7
	16 sets	4.8	11.4	20.9	42.0
2 seeks	1 set	4.4	10.4	19.2	39.1
	2 sets	4.3	10.1	18.7	38.4
	4 sets	4.0	9.7	18.0	37.1
	8 sets	3.7	8.9	16.6	37.7
	16 sets	3.2	7.6	14.4	30.1

Table 6: Consumption rate supportable (KB/sec)

		β=4 KB	β=10 KB	β=20 KB	β=50 KB
1 seek	1 set	11.8	29.4	58.8	147.0
	2 sets	9.8	24.4	48.8	122.0
	4 sets	8.8	22.0	43.9	109.7
	8 sets	8.3	20.8	41.6	103.7
	16 sets	8.1	20.3	40.5	101.0
2 seeks	1 set	7.8	19.4	38.8	96.9
	2 sets	5.8	14.4	28.8	72.0
	4 sets	4.8	11.9	23.9	59.6
	8 sets	4.3	10.7	21.4	53.5
	16 sets	4.1	10.2	20.3	50.7

Table 7: Buffer space required (KB)

		β=4 KB	β=10 KB	β=20 KB	β=50 KB
1 seek	1 set	0.94	1.04	1.19	1.64
	2 sets	0.74	0.80	0.92	1.25
	4 sets	0.67	0.72	0.81	1.09
	8 sets	0.70	0.75	0.83	1.08
	16 sets	0.85	0.90	0.98	1.21
2 seeks	1 set	1.78	1.87	2.02	2.48
	2 sets	1.36	1.42	1.54	1.87
	4 sets	1.18	1.23	1.33	1.61
	8 sets	1.16	1.21	1.29	1.54
	16 sets	1.29	1.33	1.41	1.65

Table 8: Start Latency (seconds)

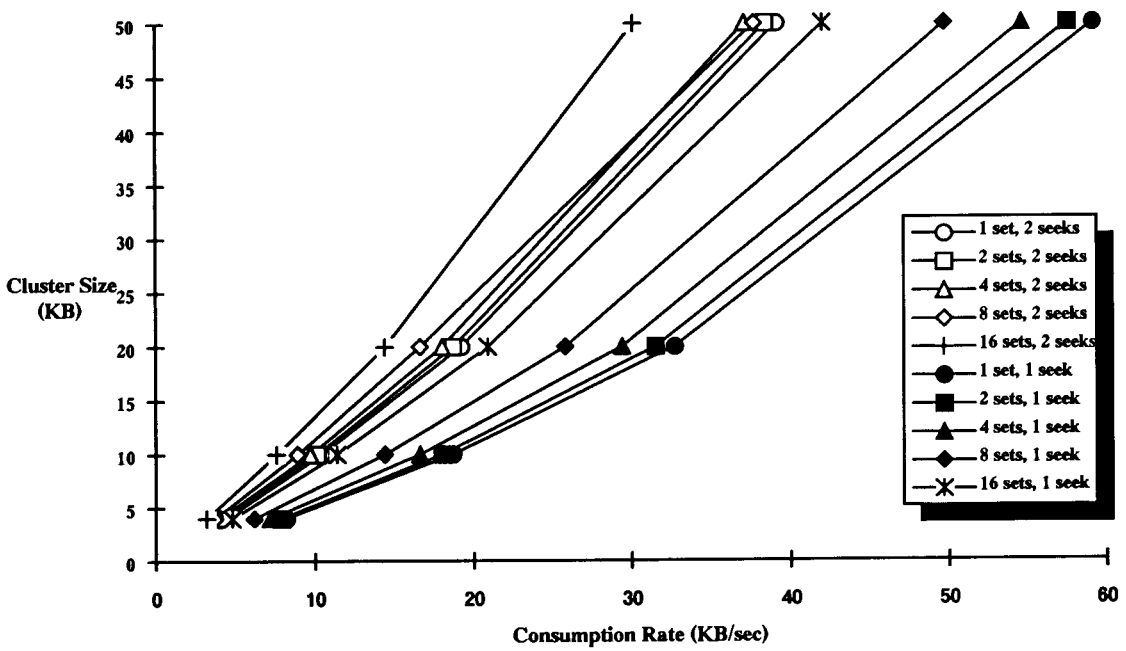


Figure 11: Cluster size required vs consumption rate supported (per stream)

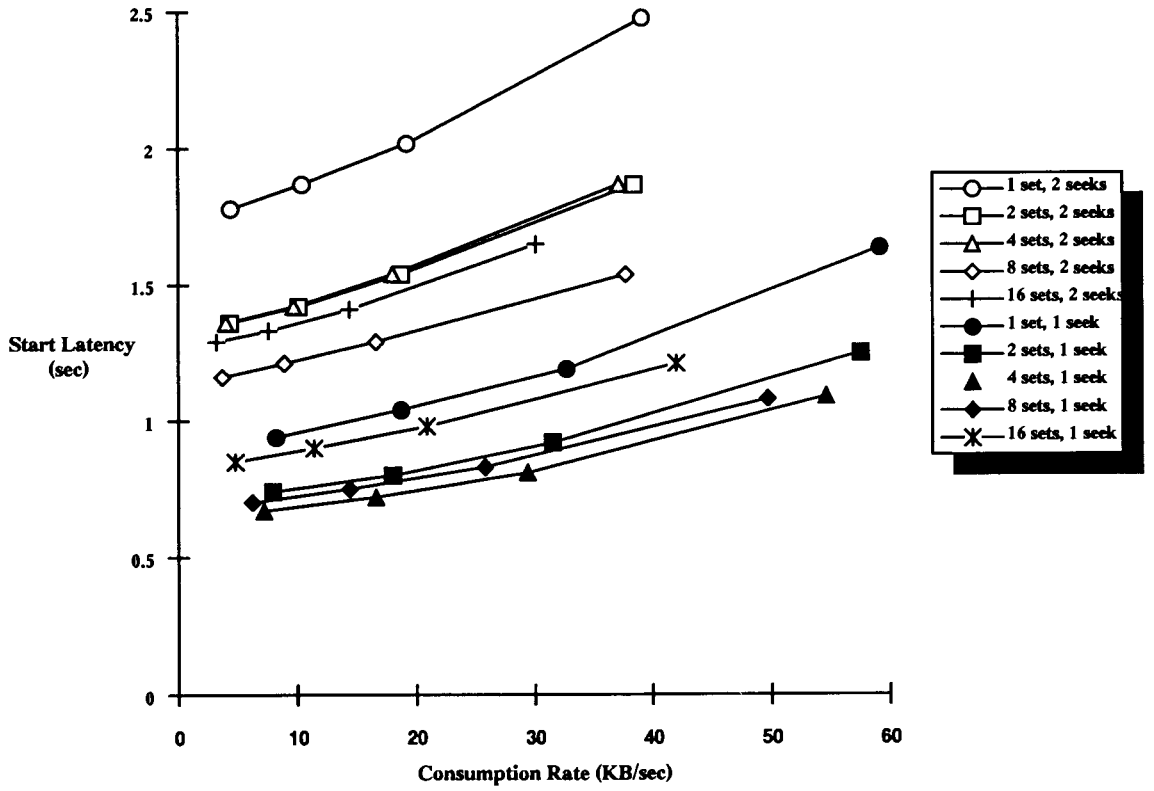


Figure 12: Start Latency vs consumption rate supported (per stream)

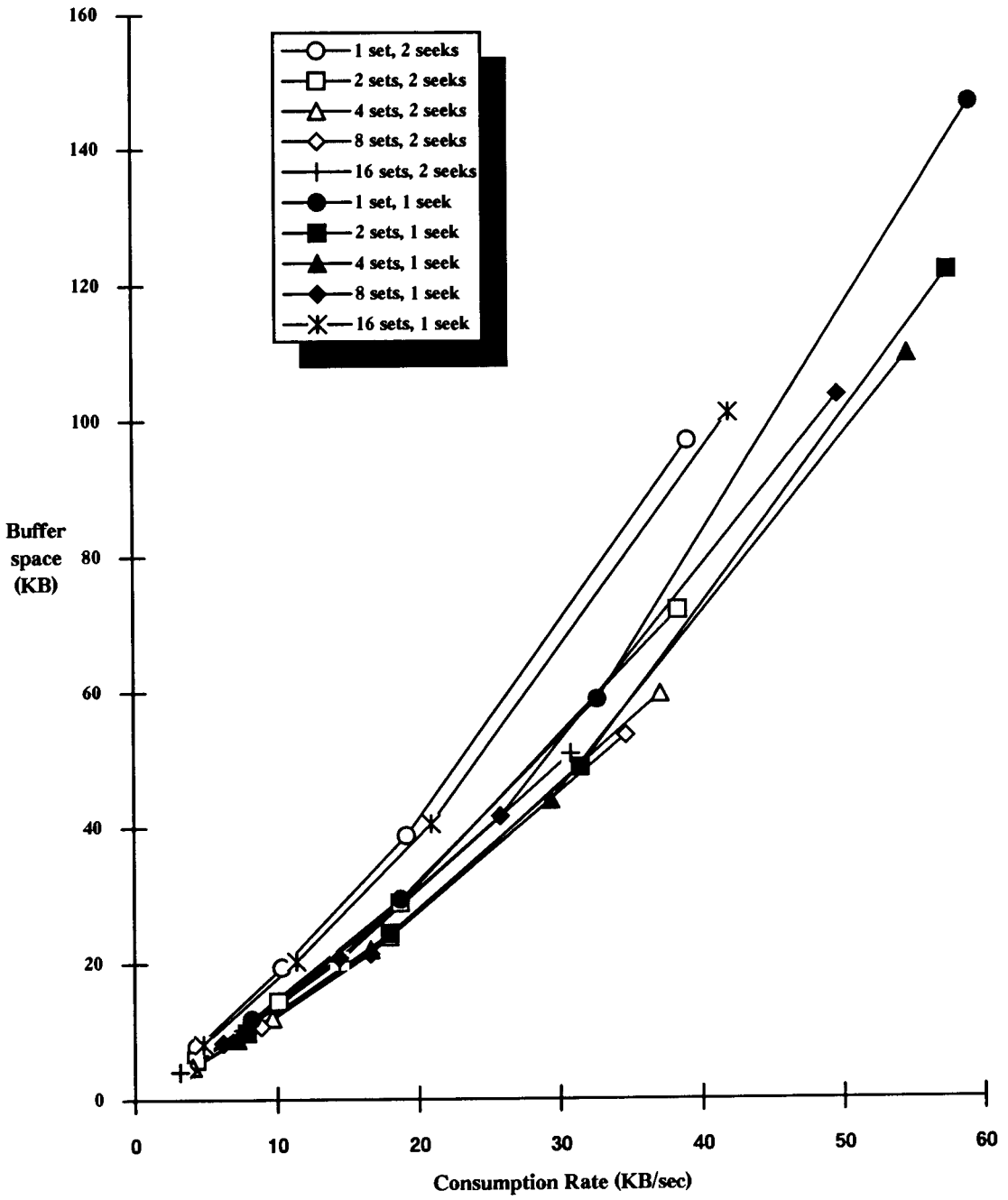


Figure 13: Buffer space required (per stream) vs consumption rate supported (per stream)

3.6 Case Study B

In this section we repeat case study A with the following changes:

- The storage system is assumed to be an array of 20 WREN 6 ST2383N disks accessed in parallel.
- We consider a range of values that gives approximately MPEG-2 video rates (512 KB/sec).

		β=300 KB	β=300 KB	β=400 KB	β=500 KB
1 seek	1 set	374	523	653	768
	2 sets	359	503	630	742
	4 sets	331	467	587	694
	8 sets	288	409	517	616
	16 sets	227	327	418	502
2 seeks	1 set	207	298	383	461
	2 sets	202	291	374	452
	4 sets	193	279	359	434
	8 sets	177	257	331	401
	16 sets	152	222	288	350

Table 9: Consumption rate supportable (KB/sec)

		β=300 KB	β=300 KB	β=400 KB	β=500 KB
1 seek	1 set	588	882	1175	469
	2 sets	488	732	976	1220
	4 sets	439	658	878	1097
	8 sets	415	623	830	1038
	16 sets	405	607	809	1011
2 seeks	1 set	387	581	775	969
	2 sets	288	432	576	730
	4 sets	238	357	477	596
	8 sets	214	321	428	535
	16 sets	203	305	406	508

Table 10: Buffer space required (KB)

		β=300 KB	β=300 KB	β=400 KB	β=500 KB
1 seek	1 set	1.04	1.11	1.19	1.26
	2 sets	0.80	0.86	0.92	0.97
	4 sets	0.72	0.77	0.81	0.86
	8 sets	0.75	0.79	0.83	0.87
	16 sets	0.90	0.94	0.98	1.02
2 seeks	1 set	1.87	1.95	2.02	2.10
	2 sets	1.42	1.48	1.54	1.59
	4 sets	1.23	1.28	1.33	1.37
	8 sets	1.21	1.25	1.29	1.33
	16 sets	1.33	1.37	1.41	1.45

Table 11: Start Latency (seconds)

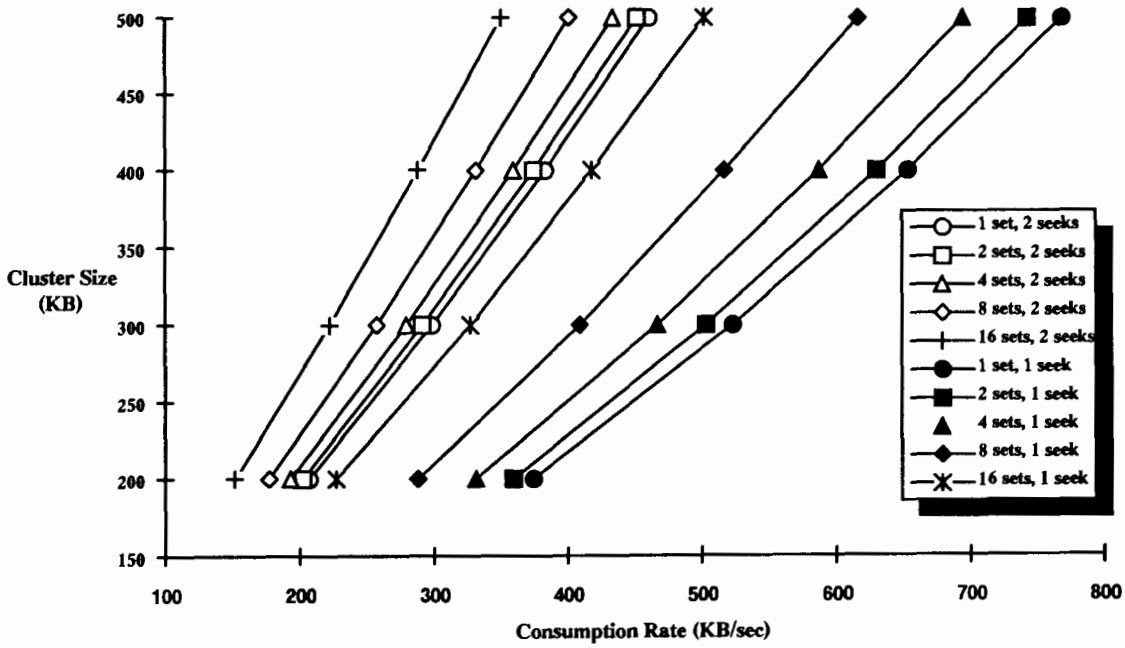


Figure 14: Cluster size required vs consumption rate supported (per stream)

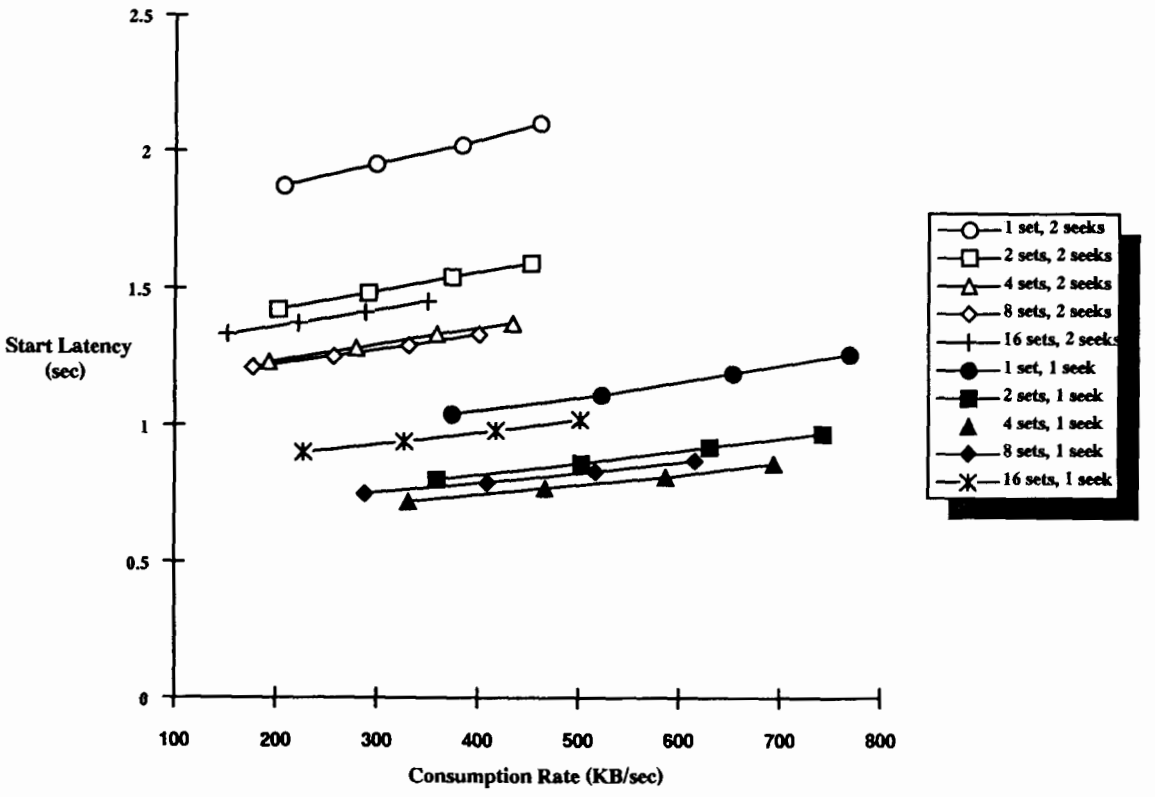


Figure 15: Start Latency vs consumption rate supported (per stream)

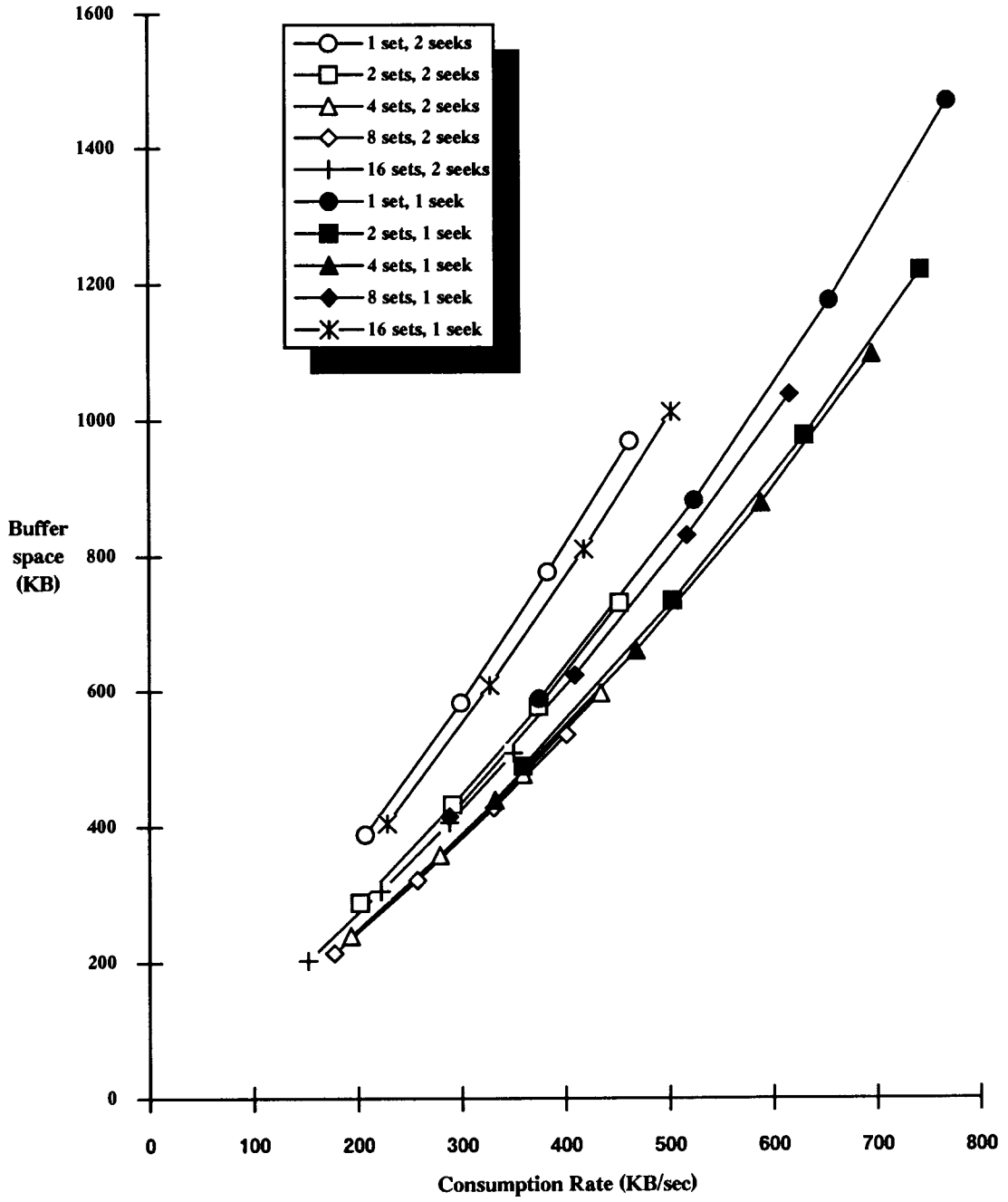


Figure 16: Buffer space required (per stream) vs consumption rate supported (per stream)

3.7 Analysis of Case Studies

The following comments apply to both case studies.

In terms of cluster size and start latency, there is a dramatic difference between allowing 1 seek and allowing 2 seeks per stream in each reading period. Buffer space, on the other hand, does not show such a dramatic difference. However, even without the consideration of buffer space we can conclude that allowing 2 seeks per channel is not worthwhile. The normal justification for requiring extra seeks is that cluster sizes cannot be made large enough to handle reading with only one seek. But here we see that allowing the extra seek requires larger clusters, and the start latency will be worse as well.

In terms of cluster size, using a single set yields optimal results. However, the 2 set and 4 set options are close contenders. For buffer space, the 2 set and 4 set options actually outperform 1 set when 1 seek is allowed. If 2 seeks are allowed, then using 1 set is actually the worst for buffer space. For start time, the best performance comes from using 4 sets, with 8 sets and 2 sets very close behind. These results for buffer space and start time are instances where sorting sets allow seek time reduction to be balanced out with the latency between successive reads to achieve optimal performance.

For these case studies, using 4 sets will optimize start time and buffer space, and provide near optimal cluster size. Using 2 sets or 8 sets will also give comparable performance. On the other hand 1 set (pure pre-seeking SCAN) would yield long start latencies, and 16 sets (round robin) would yield long start latencies and large cluster sizes. So the most important point is to use some number of sorting sets greater than one, and less than the total number of streams – the particular number is not as important.

The intuition behind this is that each time we increase the number of sorting sets by one, we add approximately one extra maximum seek because the items in the set may cause the disk head to reverse and go all the way back across the disk surface. On the other hand, the decrease in time is asymptotic – changing from 1 set to 2 sets reduces Δ_{max} from 2 readings periods to $1\frac{1}{2}$ reading periods, but changing from 10 sets to 11 only reduces from $1\frac{1}{10}$ to $1\frac{1}{11}$. Therefore, because the reduction follows the law of decreasing returns, we can expect the most significant gains in performance to be achieved with relatively few sets.

We can use this intuition to develop a heuristic estimate of the number of sets leading to optimal buffer space. Suppose the length of reading period for 1 set is T. Then increasing the number of sorting sets from 1 to n_s will give a time decrease of $T(n_s-1)/n_s - (n_s-1)t_{max}$. This will net to zero when

$$n_s = \frac{T}{t_{max}}. \tag{31}$$

As discussed earlier, a good rough approximation for T would be one maximum seek plus $(n_p\text{-}1)$ minimum seeks (assuming one seek per stream per reading period – not two). So this becomes

$$n_s = \frac{t_{max} + (n_p\text{-}1)t_{min}}{t_{max}}. \tag{32}$$

For the drive under consideration in our case studies, this is

$$n_s = \frac{28+(16\text{-}1)5}{28} = 3.7, \tag{33}$$

which corresponds to our observations well.

It is interesting to note that using 8 sets was worthwhile, because with only 2 streams in each set the disk scheduling requires only a simple comparison to schedule the nearest of the two first – a wonderfully simple computation for systems that cannot afford much overhead.

4. IMPLEMENTING THE FILE SYSTEM

4.1 Managing Storage Space

A storage server must divide video and audio streams into ***clusters***[19] while storing them on a disk. Each such data cluster may occupy several contiguous physical disk blocks. In this section, we will first describe models for storing digital continuous media on individual disks, and then discuss the effects of utilizing disk arrays as well as storage hierarchies.

4.1.1 Placement of Data Clusters

In the broadest terms, the clusters belonging to a file may be stored ***contiguously*** (one after another) or ***scattered*** about the storage device. Contiguous files are simple to implement – the directory need only keep a pointer to the start of the file – but they inherently have problems with fragmentation and enormous copying overheads may be required during insertions and deletions to maintain contiguity. In contrast, scattered placements avoid fragmentation problems and copying overheads. Thus, contiguous layouts may be usable in read-only systems (e.g. video on demand) but are not viable for flexible, read-write servers.

With regard to continuous media, the choice between contiguous and scattered files relates primarily to intra-file seeks. When reading from a contiguous file, only one initial seek is required to position the disk head at the start of the data to be read. No additional seeks are required, as the data is contiguous. However, when reading several clusters in a scattered file there may be a seek

[19] We are following DOS terminology here, rather than using the word "block", which may be confused with a physical disk block. A cluster is the atomic unit for the file system.

incurred for each cluster read. Furthermore, even when reading just a small amount of data, it may be possible that half of the data is stored in one cluster and the other half in the next cluster. So even with small reads it is possible to incur intra-file seeks by crossing cluster boundaries.

Intra-file seeks can be avoided in scattered layouts if the amount read for a stream always evenly divides a cluster. One logical approach to achieve this result is to select a cluster size that is sufficiently large, and to read one cluster in each reading period. There are several advantages to this technique, especially for large video servers. It improves the disk throughput substantially, and consequently the number of streams that can be served by the disk. Furthermore, since a file system has to maintain indices for each media cluster, choosing a large cluster size also yields a reduction in the overhead for maintaining indices. In fact, this may permit a server to store the indices in memory during the playback of a media stream.

If more than one cluster would be required to prevent starvation prior to the next read, then intra-file seeks are a necessity. Instead of avoiding intra-file seeks, another approach is to attempt to reduce them to a reasonable bound. This is referred to as the ***constrained placement*** approach [103]. Constrained placement systems ensure that the separation between successive clusters of a file are bounded. However, it does not do this for each pair of successive clusters, but only on average over a finite sequence of clusters. Thus, the latency due to intra-file seeks is constrained.

Constrained placement is particularly attractive when the cluster size must be small (e.g., when utilizing a conventional file system with cluster sizes tailored for text). However, implementation of such a system may require elaborate algorithms to ensure that the separation between clusters conforms to the required constraints. Furthermore, for constrained latency to yield its full benefits, the scheduling algorithm must retrieve all the clusters for a given stream at once before switching

to any other stream. If an algorithm like SCAN is used, which orders clusters regardless of the stream they belong to, then the impact of constrained placement is marginal [39].

A form of constrained placement was studied by Wells et al., [135] and Yu et al., [137]. In their analysis, the separation between each pair of successive clusters was fixed in order to eliminate burstiness in data transfer from disk. The applicability of such an approach, however, is quite limited due to its assumptions of intolerance to bursty transfers (i.e., severely limited buffer space), repeated exact latencies (possible with optical disks utilizing spiral tracks and tape, but not most magnetic disks), and single user playback.

In contrast to the contiguous and constrained placement approaches which reduces latencies by keeping clusters near each other, clusters may also be scattered on disk so as to make the rotational latency incurred while accessing successive clusters predictable [1]. Such an approach enables the designer to trade off seek time for predictable rotational latencies. Notice, however, that since such a placement policy may not always succeed in allocating appropriate disk clusters, the resulting access times can only be ensured probabilistically. To achieve deterministic results, rotational delays can only be reduced by data replication, multiple disk arms, or having clusters fill tracks [75].

4.1.1.1 Log-structured systems

A relatively new approach to improve I/O performance in conventional applications involves the use of ***log-structured file systems***, proposed by Ousterhout and Douglis [77]. Log-structured files systems are motivated by the success of disk caching. Typical systems can achieve 80-90% read hit rates with file caches of 0.5-5 Mbytes. With cache sizes expected to only increase, nearly all read requests will be satisfied by the cache, and the bulk of disk operations will be writes.

Furthermore, file lifetimes are relatively short in conventional systems (most files are deleted within a day). Therefore, if the cache were sufficiently reliable (and large), no disk access would ever be required for most files.

One way to make a cache reliable would be to have a battery backup powered memory. Alternately, the disk could be used to make a backup copy of cached clusters, with modified clusters written to the backup disk in a sequential stream. Because writing would be sequential, performance would be high. The cache could be reconstructed from the log disk after crashes. A log-structured file system dispenses with the conventional file system and represents the file system by the cache log only. Even file maps (inodes, indexes) are written as part of the log.

Log-structured file systems promise high performance, but face a problem with "wrap-around". Eventually the log will fill the disk space. When this happens, some sort of garbage collection will be required so that space is freed for the log to be written into. ***Log-structured systems must have large contiguous free areas to write in.*** In order for the log-structured system to perform well, disk utilization must be kept low (on the order of half the space free).

Lougher and Shepherd describe a multimedia server utilizing a "log" type file system [64]. However, it is not a true log file system. True log file systems are fundamentally associated with caches and write-intensive disk activity. There is no benefit to caching CM data, as data is nearly always accessed sequentially, not repetitively. The unique benefits of their system all stem from the fact that their "log" is compacted in idle periods, yielding a large contiguous area for writes. Like a log structured system, they do not update files in place, but rather write updates to the end of the "log" and update the file's map. Therefore, their system achieves the following benefits:

- **A single seek for writes** As all writes will be done to the end of the log, only one seek will be required for the lot of them. With seek time being the largest factor in access time, this will significantly improve performance for systems that anticipate a fair bit of writing.
- **Variable cluster sizes** Because data is always written to a large contiguous area, it is not necessary to have a fixed cluster size for the file system. When the write is performed, the size of the cluster can be chosen to suit the amount of data being written or the type of file. (See section 4.6 for further discussion of this property)
- **Temporal locality of writes** Lougher and Shepherd feel that it is likely that clusters written at the same time will be later read at the same time, especially for mixed multimedia data. Therefore, by clustering data together which is written at the same time, their system will improve performance when it is read. We would dispute the significance of this. Clusters written at the same time *by the same user* are likely to be read at the same time. Since the writes are clustered for all users, the benefit is not as great as might appear intuitively. Furthermore, in systems where writing is common, editing will be as well, meaning that data that is written together may be later presented separately.

In a multimedia server supporting conventional data, it would make sense to store conventional data in a log structure, even if log structures are not used for the CM data (e.g. partitioning the disk with a log-structured conventional partition, and a separate CM partition). This choice would be justified by the performance gain of having a single seek for writes. Compared to delay sensitive requests, conventional disk accesses are for small amounts of data. Therefore, performing more than one write with a single seek operation would mean that the throughput for conventional data would be significantly higher. To put this in perspective, in each sweep of the disk by the disk scheduling algorithm, the cost of each stop along the way is high even if very

little data is read. It is quite likely that a sweep that already contains a number of stops for delay sensitive data could only add one additional stop without violating real time deadlines. During this stop, there may be enough time to write, say, 64 KB, but not enough time for another stop even if only 1 KB was written. If data were written in 1 KB clusters, then the property of having a single seek for writes could increase the throughput by a factor of 64. Recalling that most read requests will be satisfied from the cache, we conclude that utilizing a log-structured storage for conventional data in a multimedia system could result in very good performance, where a non-log-structured storage would be very slow in supporting conventional data writes.

Performing all the writes with a single seek would also be advantageous for CM data. However, dealing with log wrap-around (i.e., generating large contiguous free space) may be more difficult. This is because conventional data is generally accessed in a bursty fashion, with many idle periods during which compaction could be done. However, CM data is accessed in a continuous way. Therefore, it may be difficult to find the time to perform compaction, depending on the application of the system. Lougher and Shepherd assume that their system is idle each night. Furthermore, the only guaranteed performance gains from this approach are for writes, not reads. We agree with Clark's assertion that aside from multimedia production houses, most users will primarily perform read-only access of CM data [29]. Therefore the benefits of this approach are quite limited. Table 12 shows the benefits due to the "log-structured" approach for some CM applications.

Application	*Writes*	*Reads*	*Edits*	*Conclusion*
Multimedia production house	Many	Very many	Very many	Some benefit
Typical computer network with CM support	Some	Very many	Some	Little benefit
Video on demand server	Few / non-real time	Very many	None	No benefit

Table 12: Benefits of "log-structured" approach

The crux of the matter in this approach is really the large contiguous free space available for writes. This yields the capability of variable cluster sizes, and reduction of seeks. While we would say that a full blown log-structured approach is ill-advised for CM applications, the benefits of a large contiguous free space make it an attractive goal in its own right. Schemes to utilize any unused system bandwidth to perform compaction are well worth researching. In the arena of multiple-disk configurations, it is intriguing to consider the possibilities of allocating space according to disk load (i.e., assign writes to idle disks, rather than in-place) or performing compaction on disks that are idle.

4.1.2 Multiple Disk Configurations

So far, we have considered storage on a single disk. However, a single disk may be inadequate for CM servers for two reasons. First, the amount of storage space required may not be satisfiable by a single disk. Second, a single disk may not be able to provide enough bandwidth to satisfy the number of concurrent accesses required of the system. The most straightforward solution to the bandwidth problem is simple replication: either replicate the entire server, or have multiple copies of a file on different disks on the same server. However, this is an expensive solution because it requires the purchase of extra storage space. A more effective approach to the problem is to

scatter each multimedia file across multiple disks. There are two general approaches taken to scattering data across multiple disks: "data striping'' and "data interleaving".

4.1.2.1 Data Striping

RAID (redundant array of inexpensive disks) technology has popularized the use of parallel access to an array of disks [80]. Under the RAID scheme, data is "striped" across each disk. Physical sector 1 of each disk is accessed in parallel as a large logical sector 1. Physical sector 2 of each disk is accessed as logical sector 2, and so on (RAID technology also generally involves some redundancy of data to increase reliability - we will ignore this aspect for the present). In this configuration, the disks in the set are spindle synchronized and they operate in lock-step parallel mode. Because accesses are performed in parallel, the access time is the same for a logical cluster and a physical cluster. Therefore, the effective transfer rate is increased by the number of drives involved.

With their increased effective transfer rate, disk arrays are a good solution to the problem of the high bandwidth requirements of continuous media. Furthermore, with disk arrays, a physical cluster is accessed from each drive in parallel, yielding an effective cluster size that increases with the number of drives in the array. Rather than being a problem, as they may be to conventional systems, large cluster sizes are highly desirable for continuous media file systems.

Observe, however, that while striping can improve the effective transfer rate, it cannot improve the seek time and rotational latency incurred during retrieval. Hence, the throughput of each disk in the array is still determined by the ratio of the useful read time to the total (read + seek) time. As with the single disk configuration, the throughput of the disk may be increased by increasing the size of the physical cluster. However, this would result in an increase in the logical cluster

size, and consequently increase the start-up delays and the buffer space requirements for the stream.

4.1.2.2 Data Interleaving

In this scheme, the clusters of the media file are stored interleaved across the set of disks. Successive clusters of the file are stored on different disks. A simple interleave pattern is obtained by storing the clusters in a cyclic manner across a set of N disks. In this scheme, the disks in the set are not spindle synchronized and they operate independently.

With this organization, there are two possible methods of data retrieval. One method of retrieval follows the data striping model in which for each stream, in every reading period, one cluster is retrieved from each disk in the set. This retrieval method ensures a balanced load for the disks, but requires more buffer space per stream. In the other retrieval method, for a given stream, in each reading period, data is extracted from one of the disks in the set. Hence, the data retrieval for the stream cycles through the set of disks in N successive reading periods. In order to maximize the throughput of the N disks, it is necessary to ensure that in each reading period the retrieval load is balanced across the disks. Given that each stream cycles through the set, this load balancing can be achieved by "staggering" the streams. With staggering, all the streams still have the same reading period length, but each stream considers the reading period to begin at a different time, so that their requests are staggered rather than simultaneous [123]. Other approaches are found in [20,23].

A combination of data striping and data interleaving can be used to scatter the media file across a large number of disks attached to a networked cluster of server machines. This technique makes it possible to construct a scaleable CM server that can serve a large number of streams from a single

copy of the media file. Moreover, redundancy techniques can be applied to the media file to increase availability and throughput. For example, by placing two copies of each cluster on different disks (machines) it is possible to protect the server against single-point failures.

4.1.3 Utilizing Storage Hierarchies

The preceding discussion has focused on fixed disks as the storage medium for the multimedia server primarily because they provide high throughput and low latency relative to other storage media such as tape libraries, optical jukeboxes, etc. In particular, they avoid the start-up delay associated with mechanical loading tapes or optical disks. However, the storage cost of fixed disks (per megabyte) is substantially higher than that of these other media, and, of course, library/jukebox strategies can offer much higher storage capacities.

In order to construct a cost-effective video-on-demand system that provides adequate throughput it is logical to use a hierarchy of storage devices. Several strategies for managing such storage hierarchies are possible. For example, the fixed disks may be used as a staging area (cache) for the secondary storage devices. Traditional cache management techniques could be applied, with an entire multimedia file being moved to fixed disk when it needs to be viewed. Alternatively, the fixed disk may store only the beginning segments of a file in order to reduce start-up latencies and ensure that real time deadlines may be met [73].

4.2 Interfacing With The Client

Based on their access model, servers can be roughly classified as ***file-system oriented*** or ***stream*** oriented. A client of a file-system oriented server sees the multimedia object as a large file and uses file-system operations such as *open*, *close*, *read* to access the file. It issues read requests to the server periodically to read data from the file. The server may use the open operation to enforce

admission control and initiate pre-fetching of the multimedia file. The server can also do periodic pre-fetching from the disk system into memory buffers to service read requests with a minimum delay. In this model, the client can implement operations such as *pause* and *resume* by simply stopping the issue of the read requests. On the other hand, a client of a stream oriented server issues commands such as *play*, *pause*, and *resume* to the server. The server uses the stream concept to deliver data continuously to the client. After the user initiates playback of the stream, the server periodically sends data to the user at the selected rate without further *read* requests from the user.

Another important issue in the server-client interface (and elsewhere) is the movement of data. Typically, data being transferred from one process (e.g. the server kernel) to another process (e.g. the client) is copied. For CM streams copying is unnecessary, takes extra time, and produces extra traffic on the system bus. Because of the high throughput requirements of CM, it is desirable to share memory, or re-map the memory into another address space to avoid copying of data [124].

4.3 Operating on Multimedia Objects

The file system must provide facilities for creating, editing, and retrieving multimedia objects. In order to guarantee continuous retrieval, editing operations on multimedia objects, such as insert and delete, may require substantial copying of their component streams. Since the streams can be very large in size, copying can consume significant amount of time and space. In order to minimize the amount of copying involved in editing, the multimedia file system may regard streams as immutable objects, and perform all editing operations on multimedia objects by manipulating pointers to streams. Thus, an edited multimedia object may contain a list of pointers to intervals of streams. Furthermore, many different multimedia objects may share intervals of the

same media stream. A media stream, no part of which is referred to by any multimedia object, can be deleted to reclaim its storage space. A garbage collection algorithm such as the one presented by Terry and Swinehart in the Etherphone system [121], which uses a reference count mechanism called *interests*, can be used for this purpose.

Additionally, a multimedia server must also support interactive control functions such as fast forward (FF) and rewind. These operations can be implemented either by playing back media at a rate higher than normal, or by continuing playback at the normal rate while skipping some data. Since the former approach may yield significant increase in the data rate requirement, its direct implementation may be impractical. The latter approach, on the other hand, may also be complicated by the presence of inter-data dependencies (e.g. in compression schemes that store only differences from previous data).

There are several approaches possible to achieve FF using data skipping. One method is to create a separate, highly compressed (and lossy), file. For example, the MPEG-2 draft standard proposes the creation of special highly compressed 'D' video frames that do not have any inter-frame dependency to support video browsing. During retrieval, when FF operation is required, the playback would switch from the normal file, (which may itself be compressed, but still maintains acceptable quality levels) to the highly compressed file. This option is interesting in the fact that it does not require any special storage methods or post-processing of the file. However, it requires additional storage space and, moreover, the resulting output is of poor resolution due to the high compression.

Another approach is to categorize each cluster as either relevant or irrelevant to fast forward. During normal operation both types of clusters are retrieved and the media stream is reconstructed by recombining the clusters either in the server or in the client station. On the other hand, during

FF operation, only the FF clusters are retrieved and transmitted. A drawback of this approach is that it poses additional overheads for splitting and recombining clusters. Furthermore, with compression schemes that store differences from previous data, the majority of data will be relevant to FF. For example, the I and P frames of MPEG are much larger than the average frame size. These facts mean that the data rate required during FF operation would be higher than the normal rate.

Chen, Kandlur, and Yu [24] present a different solution for FF operations on MPEG video files. Their method performs cluster skipping using an intelligent arrangement of clusters (called segments) that takes into account the inter-frame dependencies of the compressed video. During FF operation entire segments of video are skipped, and the viewer sees normal resolution video with gaps. Their solution also addresses the placement and retrieval of clusters on a disk array using cluster interleaving on the disk array.

4.4 Admission Control Algorithms

Given the real-time performance requirements of each client, a multimedia server must employ admission control algorithms to determine whether a new client can be admitted without violating the performance requirements of the clients already being serviced. So far we have assumed that the performance requirements of a client includes meeting all real time deadlines. However, some applications may be able to tolerate some missed deadlines. For example, a few lost video frames, or the occasional pop in the audio may be tolerable in some cases - especially if such tolerance is rewarded with a reduced cost of service. Furthermore, in order to guarantee that all real time deadlines are met, worst case assumptions must be made regarding seek and rotational latencies. In reality, the seek time and rotational latency incurred may be much less than the worst case.

Hence, a multimedia server may be able to accommodate additional clients by employing an admission control algorithm that exploits the statistical variation in the access times of media clusters from disk.

Much of the work regarding exploiting statistical variations is intended for network congestion control. However, such work is equally applicable to statistical variations in server access times. Ferrari et al. have proposed three levels of *quality of service* (QOS) [31]:

- ***Deterministic:*** all deadlines are guaranteed to be met. For this level of service the admission control algorithm considers worst-case scenarios in admitting new clients.
- ***Statistical:*** deadlines are guaranteed to be met with a certain probability. For example, a client may subscribe to a service that guarantees that 90% of deadlines will be met over an interval. To provide such guarantees, admission control algorithms must consider statistical behaviour of the system while admitting new clients [127].
- ***Best Effort:*** no guarantees are given for meeting deadlines. The server just "tries its best'' – i.e., it schedules such accesses only when there is time left over after servicing all guaranteed and statistical clients.

Notice that, providing statistical service guarantees is essential not only due to the variation in the seek time and rotational latency, but also due to the variation in the data transfer requirements of compressed media streams. To provide statistical service guarantees, a server will be required to employ precise traffic characterizations, rather than the worst-case or the average-case values. For instance, data rate requirements of a continuous media stream can be modelled as a ***linear bounded arrival process*** [10]. It is also possible that when variable rate data is stored, a complete and accurate description of the rate changes could be computed, so that the server could use the

information during playback to reserve only the required amount of server resources. The use of models and descriptions of variable rate media is a subject of on-going research [22,114,126].

For best effort traffic, there are different strategies for dealing with missed deadlines. For example, it may be desirable not to skip any clusters of data so as to ensure that the information received is intelligible. However, such a policy would increase the effective playback duration of media streams. On the other hand, if playback of multiple media streams are being temporally co-ordinated, it may be preferable to drop media clusters so as to maintain the playback time-aligned. A significant departure from these simplistic schemes are techniques which dynamically vary the resolution levels so as to adjust to the overloaded system state. For instance, Park and English [78] have proposed that during heavy network congestion, the quality of audio being delivered can be degraded simply by transmitting only the higher order bits. In general, techniques used to vary resolution to deal with missed deadlines will be very similar to those used for implementing fast forward (see section 4.3).

4.5 File Structures

A fundamental issue in implementing a file system is to keep track of which disk clusters belong to each file; keeping a map, as it were, of how to travel from cluster to cluster in a file. For contiguous files, of course, this is not an issue. For scattered files a number of approaches are possible. In this section, we consider conventional approaches to this problem, and their relative merits for multimedia file systems.[20] Table 14 summarizes the pro's and con's of each approach.

A simple solution for mapping clusters is to use a ***linked list***, with each cluster containing a pointer to the next cluster in the file. In such a scenario, the file descriptor may need to contain

[20] A good overview of conventional approaches is found in [116].

only a pointer to the first cluster of the stream. A serious limitation of this approach, however, is that random access is highly inefficient as accessing a random cluster requires accessing all of the previous clusters.

To improve the performance of random access, some conventional file systems (e.g. DOS) have utilized a ***file allocation table*** (FAT), with an entry in the table for each cluster on the disk. Each entry in the table maintains a pointer to the next cluster of a file. Assuming that the entire FAT is kept in main memory, random access can be very fast. However, it may not be feasible to keep a FAT in main memory for the large file systems expected in multimedia servers. Table 13 shows the table size required to support disk spaces of various sizes.

Table size	Disk space mapped using 16 KB clusters	Disk space mapped using 32 KB clusters	Disk space mapped using 64 KB clusters
128 KB [21]	1 GB	2 GB	4 GB
384 KB [22]	2 GB	4 GB	8 GB
768 KB [23]	4 GB	8 GB	16 GB
1536 KB [24]	8 GB	16 GB	32 GB

Table 13: Using FAT's: table size.

A FAT contains information about the entire file system, but only a portion of this information relating to files which are currently open is needed. To exploit this, it is possible to store an ***index*** for each file separately (e.g. I-nodes in UNIX [83]). These indices can be a simple list, or a

[21] 64K by 16 bits.

[22] 128K by 17 bits required. Assuming each table entry must be a multiple of 8 bits we round this up to 128K by 24 bits.

[23] 256K by 18 bits, rounded up to 256K by 24 bits

[24] 512K by 19 bits, rounded up to 512K by 24 bits.

hierarchical structure such as a binary tree (so as to make the process of searching more efficient). Thus, rapid random access is still possible, but the need to keep the entire FAT in main memory is alleviated.

Although indexes will obviously be smaller than FAT's, we must consider that with potentially enormous CM files (e.g. multi-GB movies) that a file's index may still be fairly large, and keeping the indexes of all open files in RAM may still not be feasible. If a file index cannot be kept in its entirety in main memory, then retrieving a continuous media file will involve retrieving clusters of the index in real time, in addition to the clusters of the file itself. It is true that the index retrieval is much less demanding in terms of bandwidth, but it nonetheless will consume resources. In fact, managing such small bandwidth "streams" may require special algorithms to keep them from using a disproportionate amount of system resources. An obvious way around this is to implement a linked list as well, so that real-time playback can follow the pointers contained in the clusters of data, while random seeks can be achieved quickly through the index without reserving real time resources. This would add system overhead in keeping both the index and the link pointers up to date, but for applications which perform little editing, such as video on demand, the overhead may be worthwhile.

Method	Pro's	Con's
Linked List	Simple; efficient sequential retrieval.	Inefficient random access.
FAT	Efficient random access.	May be too large to keep in RAM.
File Index	Efficient random access.	Smaller than FAT – but still may be too large to keep all open file indexes in RAM in some applications.
Index & Linked List	Efficient random access; no need to keep in RAM.	Must update two sets of pointers when file is changed.

Table 14: Options for mapping files.

Finally, since each multimedia object may contain media information in various forms: audio, video, textual, etc., in addition to maintaining file maps for each of the media streams, a multimedia server will be required to maintain characteristics of each multimedia object, such as its creator, length, access rights, and most importantly, inter-media synchronization relationships. Some information may have to be added to the file map. For instance, in a compressed audio file the file map should store the playback duration of each cluster so that random access to a desired time offset can be achieved without reading the entire file (this would also be the case for the partially filled clusters in our cut and paste scheme in section Cutting and Pasting).[25]

4.6 Storing Heterogeneous Data

For audio and video data to be retrieved so as to meet real time deadlines, it is necessary that the system cluster size be quite large. For example, for the Audition system (described in chapter 5) we have used 32 KB and 64 KB clusters. Furthermore, such files are also large, typically in the multi-megabyte range. In contrast, the average file size on a conventional UNIX system is 1KB

[25] Instead of a duration, the actual time offset could be stored. However, use of a time offset would require the time offsets to be re-written when cuts or pastes are performed.

[68]. If a cluster size of 64K were used, the average conventional file would waste 63 KB of disk space.[26] Thus, although we would like large clusters for the sake of performance, they present a problem in terms of storage efficiency.

This trade-off of performance and efficiency has been faced in the implementation of UNIX [59, 68]. Experience with the UNIX system showed that small clusters give the best storage efficiency, but severely reduce throughput for processing large files. This is because a seek may be incurred for each cluster read. The original (Berkeley) UNIX file system used 512 byte clusters. With this size of cluster only 4.2% of storage space was wasted, but the average throughput was not very good. Increasing the cluster size to 1024 bytes doubled the average throughput, and a 4096 byte cluster resulted in even greater performance. However, with 4096 byte clusters the wasted space rose to 45.6%.

To reduce the wasted space, each cluster was divided into a number of sub-clusters.[27] For example, a 4096 byte cluster may be divided into four 1024 byte sub-clusters (denoted as a 4096/1024 system). Each file would then be stored as zero or more 4096 byte clusters of data, and possibly a single sub-clustered cluster. If a file system cluster is sub-clustered to obtain space for a small amount of data, then the remainder of the cluster is made available for allocation to other files. By utilizing sub-clusters, wasted space is reduced to approximately the same level as if the cluster size were as small as a sub-cluster (i.e., a 4096/1024 system would waste about the same space as a 1024 byte cluster system). A problem with a sub-clustered system is that data

[26] This over-allocation of space is generally referred to as *internal fragmentation.*

[27] Referred to as "fragments", and a system utilizing the scheme is called "fragmented", but we will not use these terms to avoid confusion with fragmentation of disk space.

may be potentially copied each time a file grows from one sub-cluster to two sub-clusters to three, etc. This can be avoided if the user processes only write full clusters at a time.

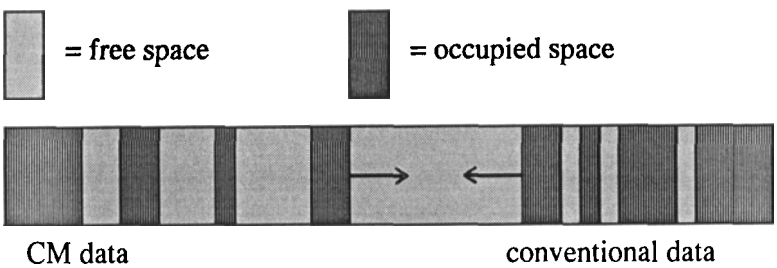
Suppose, then, that we wanted 64 KB clusters to support CM data. To mix conventional data in the same file system, it would be tempting to utilize 64KB/1KB system. However, this would lead to two problems. First, as mentioned above, for efficient writing user processes should write only full clusters. This means that for each file opened for writing, a 64KB buffer would be required. (On systems for which writes are rare compared to reads this may not be important). Second, sub-clusters are only allocated contiguously, so at the sub-cluster level a lot of space may be wasted. On a 4KB/1KB system there are only four sub-clusters within a cluster, so the impact of such wasted space is limited. However, with a 64KB/1KB system there are 64 sub-clusters, so the space wasted in sub-clusters could be very significant. Therefore, the UNIX sub-clustering approach does not seem feasible for heterogeneous multimedia data.

Without actual system use statistics, it is impossible to say how significant the fragmentation problem would be for multimedia servers. If the server is expected to handle the conventional data requirements of the typical computer user, then requiring a 64KB buffer for each file being written would be a significant problem. Under that circumstance, it would make sense to separate CM data from conventional data, either on different disks or different partitions on the same disk. The CM file space could utilize large clusters, while the conventional file space would utilize small clusters. Furthermore, this would allow the conventional file space to utilize a log-structured approach, while the CM file space could make use of some other method.

The drawback, of course, in having separate disks or partitions for each file space is that one must fix the size of each file space prior to actually using the system. A poor choice of sizes may take a large amount of system down time to correct (as the disk is re-formatted with different sized

partitions), or may be impossible (if each file system has a dedicated disk). Thus mixing CM and conventional data poses new challenges in storing heterogeneous data types.

To move beyond the fixed file space offered by partitioning, one could use a variable partition point. To achieve this, disk space would be allocated much like some main memory allocation schemes where some types of data are allocated space in upper memory proceeding downward, and other types are allocated beginning in lower memory proceeding upward (see Figure 17). That is, one could allocate large cluster (CM) data beginning in the lower disk addresses and growing upward, and small cluster (conventional) data beginning in the upper disk addresses and growing downward. We call this the ***heap approach***. Such an approach would alleviate the need for a fixed partition point – each "partition" could continue to grow until it ran into the other.



Each file space requires compaction to keep space available for the other file space.

Figure 17: The heap approach to heterogeneous data

While the heap approach allows more flexibility in the size of partitions, it lacks the flexibility to allow one file space to make use of free space in another. Suppose, for instance, that the CM file space was badly fragmented, with the overall file space occupying most of the disk, but only 50% of the file space actually being occupied. Under this scenario, the conventional file space may

become full, even though free disk space is still plentiful. The only way to make this free space available to both file spaces would be to compact the CM file space – a lengthy procedure. We should note that in our discussion of "log-structured" type systems, we mentioned the ability to support variable cluster sizes. However, this is not actually due to the log-structured approach, per se, but to that fact that data is compacted leaving a large free space. Naturally, when compaction is performed fragmentation is eliminated. So log-structured systems do not actually offer a solution to the heterogeneous storage problem – they just bring to mind the fact that any approach can benefit from compaction.

A promising approach, taken from dynamic storage allocation studies,[28] involves having zones of storage being obtained "wholesale" from the global allocation mechanism, and then "retailed" to the client [110,111]. In terms of file systems, one would ***piggy-back*** one file space on another. The CM file space, having the larger clusters, would be the global file system. The conventional file system could then obtain clusters from the CM file space, which could then break down into smaller sub-clusters, and allocate to conventional users[29]. This approach will have fragmentation problems for the piggy-backed conventional system, but the fragmentation will not be as serious as with the UNIX sub-clustering scheme, as the sub-clusters would not have the contiguity requirement that UNIX sub-clusters have. With a piggy-back approach, it may sometimes be necessary to compact the piggy-backed system to free space up for the global system, but the global system never needs to be compacted.

---

[28] Dynamic storage allocation is generally concerned with allocation of main memory, not disk space.

[29] One need not actually implement this scheme as two distinct systems. It is equivalent to have one system which considers the disk to be broken down into sections the size of the larger (CM) cluster. It always allocates the larger clusters on regular section boundaries, and always attempts to allocate smaller clusters into a section containing other small clusters before allocating space in new section. However, we speak a piggy-backed system because it is more intuitive.

Alternately, one may make the file space using the smaller clusters the global system, and simply add to it the capability to grant requests for contiguous clusters (file systems like those found in DOS or UNIX do not support such requests). If this technique is applied then the problem becomes almost identical to the classic dynamic storage allocation problem, with the alteration that the user is only able to request allocation of one of two sizes. Because of the disparity in the two sizes, it is likely that external fragmentation may become a problem (i.e., the placement of the smaller clusters may block the allocation of larger clusters, even though the total amount of free space may be much more than the size of a large cluster).

The most well known approaches to dynamic storage allocation are first-fit and best-fit [58]. With first-fit, the available storage space is scanned, and the first area that will fit the requested size is allocated. With best-fit, all free space is considered, and the area of free space closest in size to the requested amount (without being less, of course) is allocated. The relative merits of these schemes have been evaluated using simulations of memory requests, and also with logs of real requests. However, because of the restriction to only two sizes for allocation, the allocation requests for the heterogeneous storage we have been discussing are unlikely to be similar to general memory allocation requests. Therefore, the allocation schemes must be re-evaluated for this application.

The potential problem is that the smaller clusters will be spread out so as to inhibit allocation of the larger clusters. Therefore, the more compact the storage of the smaller clusters, the better. It has been observed that with first-fit, smaller allocations tend to take place in lower memory, leaving large contiguous spaces in upper memory and keeping external fragmentation low [58]. Best-fit will also keep down external fragmentation by trying to fill in small gaps in allocated

space before taking space out of large gaps. This kind of behaviour sounds ideal for the purposes of heterogeneous storage.

First-fit and best-fit could be modified to make them more efficient for two-size storage. First-fit could take on the feel of the heap approach by finding the first fit from the bottom up for conventional data and from the top down for CM data. Best-fit could find a best fit modulo the large cluster size, rather than just absolute best fit. This would help to keep large cluster size free spaces, rather than just leaving the largest free spaces.

At this point in time it is impossible to precisely evaluate the relative merits of the allocations schemes we have been discussing, because use of multimedia data is relatively new and hence statistics of typical allocation patterns are as of yet unknown. In fact, the typical multimedia system is still a matter of speculation. However, we can evaluate them in a rather general way. As we have mentioned, the UNIX sub-clustering approach appears to be too inefficient when dealing with a large disparity between cluster and sub-cluster sizes. Partitioning alleviates this problem, but is rather inflexible. The heap approach is more flexible than simple partitioning, but it becomes identical to partitioning if fragmentation becomes extensive. The most flexible and efficient schemes are first-fit, best-fit and piggy-back. In terms of allocation efficiency it is difficult to differentiate between these three without actual usage statistics, as they will all tend to keep the smaller clusters grouped together so as to keep space free for larger clusters. Table 15 summarizes the options for heterogeneous data storage.

Storage Scheme	*Complexity*	*Compaction required*	*Efficiency*
UNIX SUB-CLUSTERS	somewhat complex	file system and sub-clusters	poor
PARTITION	simple	none	good
HEAP	relatively simple	both file systems	better
FIRST-FIT	relatively simple	whole file system	best
BEST-FIT	somewhat complex	whole file system	best
PIGGY-BACK	complex for piggy-backed system; simple for global (CM) system	piggy-backed system only	best

Table 15: Options for heterogeneous data storage.

It is also conceivable that more than two file spaces would be desired. For instance, a system supporting video, audio, and text may call for a file space suited to each data type. If more than two file spaces are required, then clearly the heap approach would be infeasible. The piggy-back approach could be made multi-level, with the piggy-backed system supporting a further piggy-backed system with even smaller sub-sub-clusters. Alternately, the piggy-backed approach could be hybrid with the UNIX sub-cluster approach. Where the difference in cluster size is not too great between two file systems, sub-clusters could be adopted. Where the difference in size is large, then piggy-backing could be employed. As usage statistics become available, it may become apparent that first-fit or best-fit could also be utilized, either alone or as part of a hybrid system.

A final point of consideration in designing storage systems for heterogeneous data is addressing. It is possible that the total size of multimedia storage systems would be very large. Therefore, if an address must be given to every small cluster, a large number of bits may be required for an address. For example, using 1 KB clusters rather than 64 KB clusters would require an extra 6 bits for the address. If the same number of bits for an address were used by both the CM and

conventional file spaces, then the conventional file space would map a smaller area than the CM file space. The partition and heap approaches could easily handle this limitation, as the two file spaces are distinct. However, for piggy-back, first-fit, and best-fit, the restriction could cause problems both for implementation and performance. In terms of implementation, it would be somewhat awkward to deal with a conventional file space that only partially overlaps the CM file space. In terms of performance, the CM file space could take full advantage of any free space in the conventional area, but free space in the CM area may be out of the address range of the conventional file space, and hence unavailable to it.

4.7 Cutting and Pasting

Our experience with third-party software for editing CM files is that following a cut or paste there is a very long wait while the remainder of the file was re-written. While re-writing everything after a cut or paste may be practical for, say, word-processing, it is not for continuous media. For example, consider a stereo audio file for Audition (see Chapter 5). At 192 KB per second, a two minute file requires about 23 MB of storage. Performing a cut or paste near the beginning of the file means that 23 MB must be re-written to complete the operation. Even if the write could be performed at twice real time, there would be a one minute delay. In addition, the longer the file, the longer the delay. This again demonstrates the new approaches required by delay sensitive data. In this case, it is the extremely large storage requirements that call for different approaches to cutting and pasting. In this section we propose a solution to the cut/paste problem for delay sensitive data.

Consider a paste of n_c clusters. If n_c is an integer, then the paste need only involve updating the list of clusters in the file. If, however, n_c is not integral, then data must be borrowed from the next

cluster to fill up the last one in the paste, which will then have to borrow data from the next after it, etc. Thus the entire file from the point of the paste will have to be read and re-written. Similarly, if a cut is of an integral number of clusters then it is a simple update to the cluster list for a file, otherwise it involves re-writing.

Because it is only the fractional portion which is of interest, we will only consider cuts or pastes of less than one cluster. Also, a cut of less than one cluster is equivalent to cutting the entire cluster, and then pasting back in the remaining portion which was not cut. It is therefore sufficient to consider only pastes of less than one cluster.

Our goal is to be able to perform a paste of less than one cluster in constant time. Any paste can then be decomposed into two pastes: one consisting of a whole number of clusters, and one consisting of a fractional portion of a cluster. Pasting a whole number of clusters does not involve re-writing the file at all, and hence takes time dependent only on the length of pasted material. Pasting the partial cluster will take constant time. Therefore, any paste may be done in time dependent only on the length of the paste, regardless of file size. Similarly, a cut may be performed in time dependent only on the length of the cut.

The reason a file is re-written following a cut or paste is to eliminate gaps in the data. An obvious alternative to re-writing the file would be to simply mark the cut area in some fashion so that it is skipped in playback. This would be fine for conventional data, but with delay sensitive data it could lead to problems. When retrieving a cluster of delay sensitive data, we expect it to satisfy consumption for some amount of time, but if some of it is marked as cut then it will not last long enough and real time demands will not be met. We propose a method of cutting and pasting that utilizes over-sized logical clusters. Each logical cluster must contain a certain amount of data to meet real time deadlines, but the remainder may be marked as "empty". In performing cuts or

pastes, logical clusters may be split and combined in a way reminiscent of B-tree index clusters. When a cluster has less than the required amount of data to meet deadlines, it may "borrow" data from its neighbours (see Figure 18).

For simplicity, we will apply the filling requirement for the logical cluster at the cluster level. Each cluster will contain a header indicating how much valid data it contains. For example, if the logical cluster is over-sized such that only 90 percent of it need be filled with media quanta, then each cluster may have up to 10 percent taken by the header and blank space.

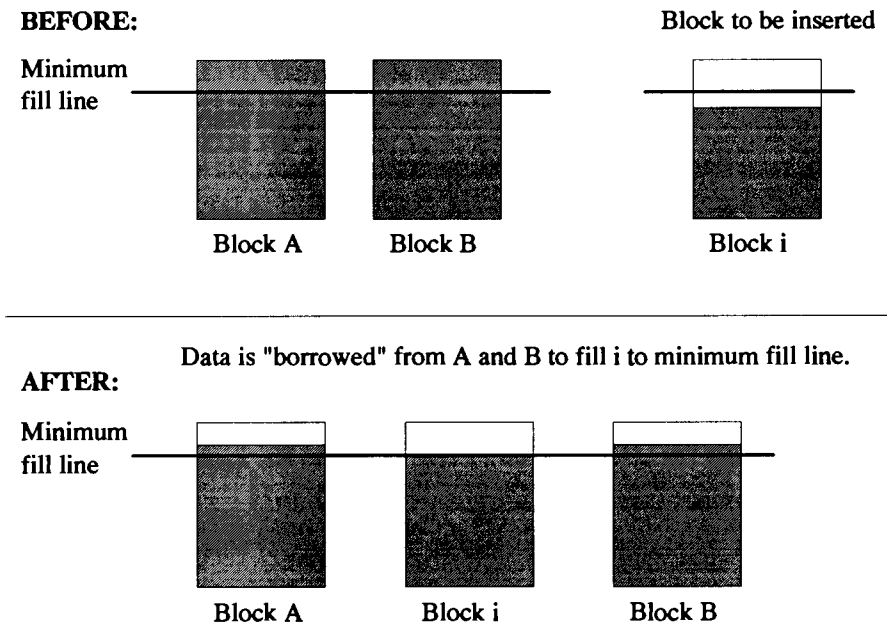


Figure 18: Pasting a partially filled cluster.

Let c_h be the size of a cluster less the size of the header, that is, the amount of usable space in a cluster. Let f be the fraction of usable space that must be filled with data in order to keep up to real time ($0<f<1$). Suppose then that a paste of size x must be made, with $0 < x < c_h$. If $x \geq f c_h$ then the new data will form a valid new cluster, with enough data to maintain real time schedules. However, if $x < f c_h$ then the data must be merged with other clusters.

Claim: Using partially filled clusters, as described above, a sub-cluster paste can be performed in constant time, reading $\lceil f/(1-f) \rceil$ clusters and writing at most $\lceil f/(1-f) \rceil + 1$ clusters.

Proof: For a sub-cluster pastes, $x < f c_h$. Consider a group of $\lceil f/(1-f) \rceil$ contiguous clusters into which the paste is being performed. We will pool the data from these clusters with the newly pasted data, and subdivided the whole into new clusters.

Each cluster has at least $f c_h$ valid data in it and at most c_h valid data in it. Therefore, the total amount of pooled data is greater than $\lceil f/(1-f) \rceil f c_h$, and is less than $\lceil f/(1-f) \rceil c_h + f c_h$. Now let $q = \lceil f/(1-f) \rceil c_h$. Note that q is within the bounds for the total amount of data. If the total amount of data is less than or equal to q, then the data can be divided into $\lceil f/(1-f) \rceil$ equally filled clusters, each containing more than $f c_h$ data and at most c_h data. If the total amount of data is greater than q, then it can be divided into $\lceil f/(1-f) \rceil + 1$ equally filled clusters. At the very least each cluster will contain more than

$$\frac{\lceil f/(1-f) \rceil}{\lceil f/(1-f) \rceil + 1} c_h = \frac{c_h}{1 + \lceil f/(1-f) \rceil^{-1}} \geq \frac{c_h}{1 + (1-f)/f} = f c_h. \tag{34}$$

Also, because $f<1$, each cluster will contain less than c_h data.

Therefore, by reading $\lceil f/(1-f) \rceil$ clusters, we can perform the paste and then write the data back as either the same number of clusters, or one more.

■

By selecting the value for f, system designers can choose what the upper limit on time for cuts and pastes will be. For example, with $f = 0.5$, only one cluster needs to be read, and at most two need to be written. Table 16 shows the numbers of reads and writes required for various values of f. Also, it should be noted that the portion of the cluster which need not be filled for real time is not

necessarily wasted space. Files initially would use this space, and it would only become empty to perform quick cuts and pastes. It would always be possible to then compact the file to free up extra disk space. The only absolute penalty in disk space is the addition of the headers, which only need contain one value (quantity of valid data in cluster) and thus will be of negligible size.

% of cluster which must be filled	# reads required	# writes required
50%	1	2
60%	2	3
70%	3	4
80%	5	6
90%	9	10
95%	19	20
100%	potentially entire file	potentially entire file

Table 16: Reads and writes required for sub-cluster paste.

Observe that the potentially empty portion in each cluster was generated by increasing the size of a logical cluster, not necessarily the cluster size. While the logical cluster size may of course be increased by increasing the cluster size, it is also possible to increase it by increasing the number of clusters in a logical cluster. This would avoid having cluster sizes which are too large to be suitable for other data types.

The general approach we have taken for cutting and pasting has been applied previously in conventional, non real-time systems. Our presentation of the technique is important in the context of delay sensitive systems because it demonstrates that sub-cluster cuts and pastes can be performed in constant time, leaving the file in a state which can still support real time storage and retrieval.

5. THE AUDITION SYSTEM

We had the opportunity to put some of our research into practice in the design of Audition, an audio evaluation system for subjective listening tests of high quality audio processing and coding algorithms at MPR Teltech Ltd. Clearly for such tests no degradation of the audio quality can be tolerated, and hence real time deadlines must be strictly observed. In this chapter we describe our experience in the design and implementation of Audition, and the lessons that were learned. Although Audition supports both playback and recording, we will continue to limit our discussion to playback only for the sake of clarity.

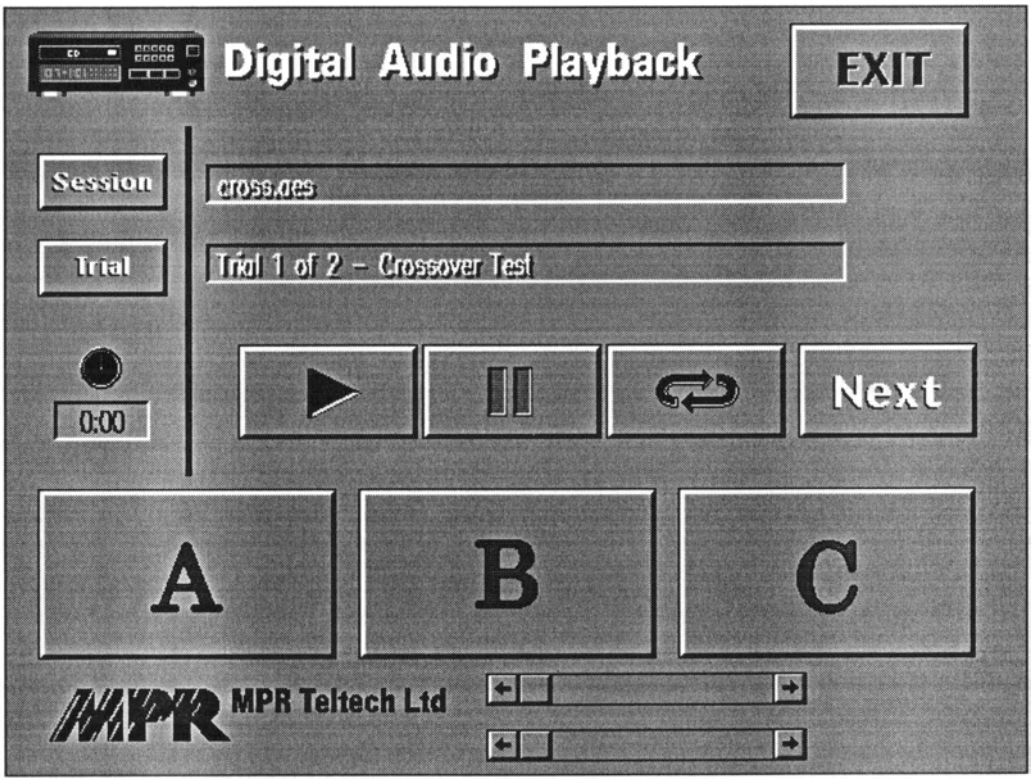


Figure 19: The Audition Interface.

5.1 The Application

Various compression schemes may be used to reduce the bandwidth requirements of digital audio. A compression scheme may be either lossless, i.e., after compression and decompression the data is exactly the same, or lossy, i.e., after compression and decompression the data is not exactly the same [81]. Since some data is lost, lossy schemes can achieve higher compression ratios than lossless schemes. Among the lossy schemes gaining popularity for digital audio are those which apply an understanding of human hearing to the compression [48]. These schemes attempt to ensure that the effect of lost data is virtually inaudible. In order to compare such compression methods it is not enough to just analyse the changes introduced to the data by compression and decompression. The object of the methods is to yield subjective improvements, and only subjective testing can evaluate their success.

The Audition system was designed to perform subjective listening tests, with its first application being the comparison of compression methods. Audition is based on an IBM PC compatible platform, equipped with a SCSI drive and DSP processing card. The DSP card includes an AES/EBU[30] digital audio interface for connection to a high quality external analog to digital/digital to analog conversion unit. Audition can record/playback three stereo files simultaneously.

To compare compression methods, a short audio segment is compressed and decompressed by some method and then recorded on Audition. The same segment is recorded using other compression schemes, and also with no compression. Audition then simultaneously retrieves three stereo files, known only as A, B and C to the user. At any time the listener only hears one

[30] Audio Engineering Society / European Broadcast Union – a standard for stereo digital audio transmission.

of the three, and can instantaneously switch between them with a keystroke or mouse click. Figure 19 shows the user interface. Selection A is always the original unprocessed audio source (control) , while B and C are randomized to either the control or the processed audio. The listener grades each of B and C with respect to A. Audition will continuously repeat (loop) the segment until the user is satisfied with the grading. Setting up a new set of trials for a listener is a simple matter of typing in a new list of files to play.

In contrast, the preparation of material for subjective listening tests is normally an extremely labour-intensive activity. Construction of an A-B-C listening test tape involves many hours of editing and assembling small segments, which must be repeated if another random ordering of the trials is desired. If listener-controlled switching is required, two or more tape machines must be synchronized and controlled together, and a reliable analog switching mechanism employed. Continuous looping of the trial is not possible using a tape-based system, unless the repetitions are pre-recorded onto the tape.

With a lossy compression scheme, applying compression to data which was already compressed and decompressed will degrade the data further. Therefore it is also important to compare lossy compression schemes after multiple "generations" of coding/decoding. In fact, this kind of comparison is relevant to any sort of audio processing equipment. To facilitate such testing, Audition includes a tandem recording program, which automates the production of multiple generations of processed audio material. Stereo files are simultaneously played through the equipment under test (e.g. a coder and decoder) and recorded back onto the disk. The random access nature of the disk medium allows compensation in real time for the delay through the equipment to produce precisely time-aligned files required for the A-B-C subjective assessment methodology.

5.2 Designing the Audition system

Audition was designed to run on an IBM PC compatible. In order to accommodate the high bandwidth requirements of the system, a SCSI disk interface was selected. An early prototype system had a data flow that went from disk, through the computer's bus to main memory, then through the bus again from main memory to an audio card that performed the output. This approach was undesirable for two reasons. First, it consumed an enormous amount of bus bandwidth, with the bus becoming a severe bottleneck. Second, the operating system on the PC could impact the data flow. This would make the use of conventional, non-real time, operating systems difficult. For these reasons the design was changed so that data flowed from disk to an audio card fit with a SCSI interface, and directly from the card to audio output (the card can be set up with either an AES/ABU digital audio interface or D/A and A/D converters). The PC now merely controls the operation of the card, without being involved in actually moving the data. Note that this also leads to a more general purpose solution, as only the SCSI disk and bus need to be dedicated to audio playback system; all other resources may be freely utilized by other processes.

Rather than designing a new file system for Audition, we decided to utilize the DOS file system. This gave us the ability to easily transfer files and make use of other software for editing them. It also meant that the disk didn't need to be dedicated to audio use; all the software for the system is stored on the same drive. The PC can read blocks from the drive via the audio card. However, it would have been too costly to implement the BIOS calls needed to utilize the audio card as a disk controller for DOS. Therefore, another SCSI controller was used to give the PC direct access to the drive. Figure 20 shows the layout of the Audition system.

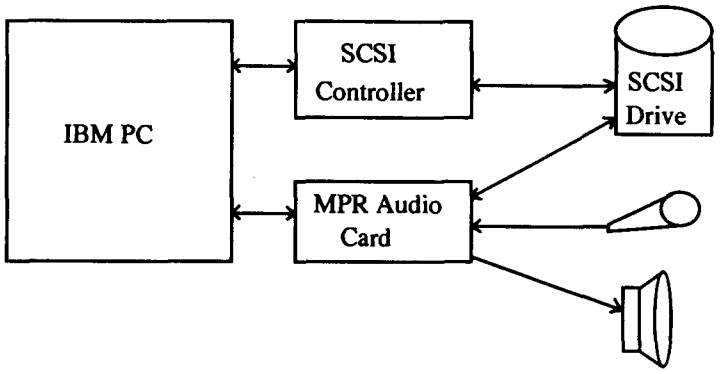


Figure 20: The Audition System.

Audition creates stereo audio files in RIFF format - the standard for Windows' multimedia extensions. Once created, data is written/read directly from the SCSI interface on the audio card. Audition allows the user to select one of three stereo files, and switch between them on the fly. To accomplish this, the audio card continuously retrieves data for all three files. The card has been equipped with a DSP processor, so that when the user switches from one channel to another, a quick cross-fade can be performed between the two channels[31]. The card contains enough memory to perform double buffering for each channel. While one buffer for a channel is being consumed, the other buffer is being filled with data.

The audio files are not compressed in any way. Clearly a lossy compression would be unacceptable for the purposes of the system, because it would distort the audio signal. Most common lossless compression schemes do not work well with audio; they yield unimpressive compression ratios, sometimes even increasing file size. Even had there been a suitable compression scheme, it is unlikely that it could be implemented without extra hardware, as the

[31] A cross-fade gradually reduces the volume of one source from maximum to zero, while at the same time increasing the new source from zero to maximum. This prevents a pop from being heard when switching channels.

DSP on the audio card was nearly fully loaded with the task of managing data movement for the three stereo files.

In the DOS file system, data is stored in fixed length clusters, consisting of several contiguous physical blocks. Typically, a physical blocks is 512 bytes, and a cluster is on the order of 8 blocks. The maximum cluster size DOS supports is 64 KB. DOS does not provide any mechanism for specifying the placement of the clusters so it is possible that a seek will be required for each cluster accessed.

To evaluate the suitability of a particular SCSI drive, we made use of the formulas in chapter 3. The first consideration is the bandwidth requirement of 3 stereo channels, sampled at 48 kHz, with a sample size of 2 bytes. Therefore, the transfer rate of the drive must be at least 3x2x48,000x2 = 576,000 bytes/sec. As is pointed out in 3.1.1 the transfer rate should actually be well above this to maintain reasonable buffer requirements.

In Audition, all channels are synchronized. Therefore, we can use equation (8) to describe the system. Suppose that for each channel in a reading period we read n_c clusters, each of size c_s bytes. Using $b = n_c c_s$ we can restate equation (8) as

$$n_c c_s \geq \frac{d_{max}}{1/r_c - n_p/r_t}. \tag{35}$$

We know that delays can include seek time delays. We also consider the possibility of a worst case rotational delay, T_{rot}, and a delay due to crossing track or cylinder boundaries, T_c. Assuming the pre-seeking SCAN algorithm will be employed, we can use equation (26) for our seek time. Including this with the other delay terms described above, we can rewrite equation (35) as

$$n_c c_s \geq \frac{T_{max} + (n_c n_p - 1)\left(T_{min} - \frac{T_{max} - T_{min}}{S_{max} - 1}\right) + n_c n_p (T_{rot} + T_c + \varepsilon)}{1/r_c - n_p/r_t}. \tag{36}$$

In order to determine our cluster size, we solve for c_s:

$$c_s \geq \frac{\frac{T_{max}}{n_c} + \left(n_p - 1/n_c\right)\left(T_{min} - \frac{T_{max} - T_{min}}{S_{max} - 1}\right) + n_p(T_{rot} + T_c + \varepsilon)}{1/r_c - n_p/r_t}. \quad (37)$$

Note that while increasing the number of clusters per block, n_c, will asymptotically decrease the required cluster size, it will linearly increase the delay time. To calculate the buffer space required by the system, we recall that for a double buffered system a buffer is the same size a logical block, $b = n_c s$. Therefore the buffer space requirement will be $2\, n_p\, n_c\, c_s$.

For Audition, n_p=3 and r_c = 48 kHz x 2 bytes x 2 (stereo) = 192,000 bytes/sec.

Seagate WREN 6 ST2383N		
	Drive capacity (unformatted)	383 MB
T_{min}	Track to track seek	5 msec
T_{max}	Maximum seek	28 msec
T_c	Cylinder crossing	6 msec
T_{rot}	Rotational delay	17 msec
r_t	Transfer rate	2 MB/sec
S_{max}	Maximum seek	1,260 cylinders
c_s	Size of cluster	26,804 + 6,166/n_c bytes
	Buffer space	160,824 n_c+ 36,996 bytes

Table 17: Using a WREN 6 ST2383N for Audition.

Table 17 shows the performance characteristics for a Seagate WREN drive, along with the derived cluster size and buffer space requirements. The value used for T_c was the time of one cylinder crossing, that is, we assumed that a cluster will always be less than two tracks of data in length. This is reasonable since two tracks of data would exceed 64 KB (the maximum cluster

size allowed by DOS). For this drive, we see that using more than one cluster per logical block is not economical. For instance increasing n_c from 1 to 2 will increase the buffer requirements by about 160 KB, while only decreasing the cluster size by about 3K. Using this drive, Audition would require clusters of about 33 KB and about 200 KB of buffer memory - both very manageable figures. If 300 MB of the disk were free for audio use (the rest being used by DOS, Windows, Audition and other software) then it could hold approximately 26 minutes of stereo audio data.

Using the figures given above, we expected to be able to meet all real time deadlines. However, we discovered that there are other features of a disk drive which may affect real time performance. For example, the first prototype of Audition would occasionally fail to meet real time deadlines for no apparent reason. This was a mystery to our design team and to many of the technicians of the drive manufacturer. Eventually we spoke to a technician who was aware of an undocumented feature of the drive, which is to perform a thermal re-calibration to compensate for contraction/expansion due to temperature changes. This re-calibration could take on the order of a second - more than enough to put playback behind its real time schedule. Fortunately, we were able to find drives by another manufacturer which didn't need to perform thermal re-calibrations.

Another feature which may affect real time performance is sector re-mapping. Sector re-mapping is used for bad blocks. When the user requests a bad block, the drive "re-maps" the request to another block that was previously marked as a "spare" (i.e., reserved as a replacement for bad blocks). This is transparent to the user, and would frustrate attempts at disk head scheduling. We found that on some drives it is possible to disable sector re-mapping and perform bad block management ourselves.

It is interesting that the current trend in disk drives is towards hiding information from the user. From a conventional systems standpoint, this is an improvement, just like information hiding in programming. However, for delay sensitive systems which require precise timing information to estimate times and to optimize, information hiding is at cross-purposes. Ironically, while we chose SCSI drives due to their high performance, the SCSI interface itself dictates that blocks are requested only by logical number. The actual cylinder and track location are hidden from the user. Thus, utilizing disk hardware for a CM system becomes an exercise in uncovering what has been hidden. Developers should take care in evaluating hardware for CM systems, as most hardware is not built, nor specifications written, with any consideration for real time requirements[32].

5.3 Results

Once the problems of thermal re-calibration and sector re-mapping were dealt with, the Audition system performed up to real time specifications without exception. The system has even run successfully with sample sizes increased from 16 to 24 bits. The Audition system has now been used extensively in recent CCIR testing of low bit rate audio compression algorithms for digital audio broadcast. CCIR TG 10/2 will produce a recommendation for the coding system to be used in the entire digital audio broadcast chain (studio, contribution, distribution, and emission). Audition was also used by the British Broadcasting Corporation (BBC) to produce multiple tandem coding generations through each proposed system in the selection of critical audio materials for subsequent subjective assessments. Additionally, Audition has been used in the

[32] When Audition was first developed (1992) this was certainly true. At the time of writing (Oct. 1994) disk drive manufacturers are just beginning to mention thermal recalibrations and other factors important to real time systems in their advertisements and spec sheets.

CCIR listening tests conducted at the CRC (Communications Research Centre, Ottawa, Canada) in contribution, distribution and post-processing tests. The use of a disk-based system was instrumental in producing reliable results for the high quality systems under test.

Use of the DOS file system and RIFF format files was very satisfactory. The problem of backups was handled by DOS. Files could even be copied on floppies. Note that the requirement for large cluster sizes only exists for real time playback/recording. For backups and copying an audio file is like any other DOS file. As the files were in standard format, software by other vendors was used to edit the files where necessary. This was convenient for removing glitches from recordings and trimming them to a precise time.

6. CONCLUSION

Continuous media presents new challenges to system designers because of its real time, high bandwidth nature. Supporting continuous media in a file server requires rethinking conventional server strategies including disk scheduling, admission control, data placement, and file mapping. In section 6.1 we highlight the contributions from our research. In section 6.2 we point out some likely candidates for future research.

6.1 Contributions

6.1.1 Real Time Retrieval

We have carefully formulated the requirements for real time retrieval (and hence storage, which is an entirely symmetrical problem). We have assumed the processing of requests in reading periods, and that retrieval has the capability of being buffer-conserving (the net change in buffer space in a reading period is non-negative – see chapter 3). From our formulation, we can make some general statements. First, one should not attempt to utilize all of the available bandwidth of a storage device; on the order of 75% utilization is probably about as high as is feasible (see section 3.1.1). Second, each stream should only be allowed one seek per reading period (see section 3.6).

Our formulation also yields some important, exact results. We have given lower bounds on buffer space and the amount read in each reading period in section 3.3. In that section we also demonstrate a greedy algorithm that achieves those lower bounds, and describe an alteration to the greedy algorithm which allows a simpler implementation but requires some extra buffer space.

6.1.2 Disk Scheduling

Because seek latencies tend to dominate retrieval times, disk scheduling is very important. We have introduced the concept of sorting sets (see section 3.2.2). Using sorting sets for disk scheduling trades off the length of a reading period with the number of requests processed by other clients between successive reads by a given client. Our analysis shows that this allows buffer space requirements to be minimized. This minimization will come at the expense of increased cluster size and system response time (as indicated by reading period length), so we expect that in general the number of sorting sets will remain relatively small to balance out these competing requirements (see section 3.6).

In addition to being able to produce improved performance, the sorting set paradigm is important by virtue of being a generalization of previous approaches. Round robin scheduling and unmodified SCAN scheduling, both popular in CM literature previously, are both special cases of the sorting set model. Therefore, the sorting set approach allows a more direct comparison of approaches than was previously possible, making it easier for system designers to select among the alternatives.

The sorting set paradigm does not necessarily specify the disk scheduling strategy used for each sorting set. However, we have presented a modification to the SCAN algorithm, pre-seeking SCAN, which we have proved to be optimal (see section 0). While pre-seeking SCAN is not especially novel in terms of conventional disk scheduling, it is important for CM disk scheduling for two reasons. First, it demonstrates the existence of an optimal algorithm. Second, in describing the algorithm, we also explain how to use an explicit seek command and careful assignment of time to satisfy real time requirements.

6.1.3 File System Implementation

In our discussion of file system implementation, we survey approaches and bring together concepts from different sources to give the reader a full perspective on the issues (see sections 4.1 through 4.5). In considering storage of heterogeneous data, we propose the use of "heap"-like strategies, and a "piggy-back" strategy in addition to the conventional approaches of partitioning or UNIX-like "fragmenting" (see section 4.6). To deal with the issue of cutting and pasting, we show how to utilize partially filled clusters to guaranteed constant time sub-cluster cuts and pastes – a great improvement over conventional approaches, which can potentially re-write the entire file after such a cut or paste (see section 4.7).

6.2 Future Research

The number of researchers studying continuous media servers has exploded during the writing of this thesis, and at this point many of the fundamental problems are now well understood. However, there are still some strong candidates for future research.

Handling of irregular media remains the most pressing problem in the field. As we have mentioned, there have been some models proposed to describe irregular media, using with applications for these models being for network transmission. However, we are still a long way off from models which can be used for deterministic results and proved optimal in any regards, be it for disk utilization or for buffer space requirements.

Another area which is just beginning to receive attention is the use of multiple disks. Novel suggestions for exploiting the parallelism of multiple disk accesses have been forthcoming during the past year, but to date no one has unified and formalized the study.

Furthermore, the technology of disk storage is changing. More and more disks are utilizing variable rotation rates and/or variable storage densities and/or variable transfer rates. Essentially all the work in this field assumes fixed rotation and transfer rates, so it is possible that these properties may be exploited to further improve performance.

Finally, specific applications allow for special algorithms. For example, we have heard of one approach that exploits the fact that videos may occupy nearly a full disk, and are used in primarily read-only fashion. Similarly, research is justified into algorithms tailored to support for extensive editing, coarse interactive control,[33] or numerous small bandwidth streams.

[33] For example, video-on-demand applications may be able to tolerate playback restricted to beginning at, say, a five second boundary, rather than any point at random. Such a restriction would allow batching of clients into these five second boundary groups.

7. APPENDIX: SYMBOLS

Symbol	Description
b	The size of a logical block (cluster).
β	The amount actually read during a reading period for a given stream.
$B(t,t_0)$	The amount buffered at time t, with consumption beginning at time t_0.
$C(t,t_0)$	The amount consumed at time t, with consumption beginning at time t_0.
c_h	The size of a cluster, less the size of the header.
c_s	The size of a cluster.
d_{max}	The maximum duration of delays in a reading period.
Δ_{max}	The maximum time between the completion of successive reads for a given stream.
ε	Error term to compensate for inaccuracies in the seek time model.
f	The fraction of usable cluster space (size c_h) that must be filled to ensure buffer-conservation.
$L_s(m)$	The seek time caused by moving the disk head m tracks.
m	length of seek (tracks)
m_{max}	The number of tracks in a maximum seek.
n_c	Number of clusters (e.g. in a cut/paste or read in a reading period)
n_p	The number of playback streams.
p_{max}	The maximum duration of a reading period.
$R(t)$	The amount of media quanta read from the storage device at time t.
r_c	The rate at which media quanta are consumed for output.
r_{cmax}	The maximum consumption rate the system will allow for a given stream.
r_t	The transfer rate of the storage device.
s_i	Stream i.
S_i	Sorting set i.
$T(S)$	The maximum time to execute the reads of sorting set S, including all latencies.
T_c	The duration of a cylinder crossing.
T_{max}	The duration of a maximum seek.
T_{min}	The duration of a minimum (track-to-track) seek.
T_{rot}	The duration of a rotational delay.

8. BIBLIOGRAPHY

[1] Abbot, Curtis, Efficient Editing of Digital Sound on Disk,*J. Audio Eng. Soc.*, Vol. 32, No. 6, (June 1984) pp.395-402.

[2] Anderson, David P., A Software Architecture for Network Communication, *Tech. Rep. UCB/CSD 87/386, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Dec. 9, 1987.

[3] Anderson, David P., Meta-Scheduling for Distributed Continuous Media, *Tech. Rep. UCB/CSD 90/599, Computer Science Div., EECS Dept., Univ. of Calif. atBerkeley,* Oct. 1990. (Also to appear in ACM TOCS)

[4] Anderson, David P. and Ferrari, Domenico, The DASH Project: An Overview,*Tech. Rep. UCB/CSD 88/405, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Feb. 1988.

[5] Anderson, David P., Ferrari, Domenico, Rangan, P. Venkat, and Tzou, Shin-Yuan, The DASH Project: Issues in the Design of Very Large Distributed Systems, *Tech. Rep. UCB/CSD 87/338, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Jan. 1987.

[6] Anderson, David P., Govindan, R., and Homsy, G., Abstractions for continuous media in a network window system, *Proceedings of the International Conference on Multimedia Information Systems '91 (Singapore)* pp. 273-298, McGraw-Hill, New York, 1991.

[7] Anderson, David P., Govindan, R., Homsy, G., and Wahbe, Robert, Integrated Digital Continuous Media: A Framework Based on MACH, X11, and TCP/IP, *Tech. Rep. UCB/CSD 90/566, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* March 1990.

[8] Anderson, David P., Herrtwich, Ralf G., and Schaefer, Carl, SRP: A Resource Reservation Protocol For Guaranteed-Performance Communication in the Internet, *Tech. Rep. UCB/CSD 90/562, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Feb. 1990.

[9] Anderson, David P. and Homsy, George, A Continuous Media I/O Server and Its Synchronization Mechanism, *Computer*, Vol. 24, No. 10 (October 1991), pp. 51-57.

[10] Anderson, David P., Osawa, Yoshitomo, and Govindan, Ramesh, A File System for Continuous Media, *ACM Transactions on Computer Systems,* Vol. 10, No. 4 (November 1992), Pages 311-337.

[11] Anderson, David P., Osawa, Yoshitomo, and Govindan, Ramesh, Real-Time Disk Storage and Retrieval of Digital Audio/Video Data, *Technical Report No. UCB/CSD 91/646, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Aug. 8, 1991.

[12] Anderson, David P., and Tzou, S., The DASH Local Kernel Structure, *Technical Report No. UCB/CSD 88/463, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Nov. 1988.

[13] Anderson, David P., Tzou, S., and Graham, G. S., The DASH Virtual Memory System, *Tech. Rep. UCB/CSD 88/461, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Nov. 1988.

[14] Anderson, David P., Tzou, Shin-Yuan, Wahbe, R., Govindan, R., Andrews, M., Support For Continuous Media in the DASH System, *10th International Conference on Distributed Computing Systems, May 28-June 1, 1990 Paris, France.*

[15] Anderson, David P., and Wahbe, Robert, A Framework for Multimedia Communication in a General-Purpose Distributed System, *Tech. Rep. UCB/CSD 89/498, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* March 31, 1989.

[16] Anderson, David P., and Wahbe, Robert, The DASH Network Communication Architecture, *Tech. Rep. UCB/CSD 88/462, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Nov. 1988.

[17] Banger, Colin, and Pennycook, Bruce, Gcomp: Graphic Control of Mixing and Processing,*Computer Music Journal,* Vol. 7, No. 4 (Winter 1983), pp. 277-283.

[18] Barberis, Giulio, and Pazzaglia, Daniele, Analysis and Optimal Design of a Packet-Voice Receiver, *IEEE Transactions on Communications,* Vol. COM-28, No. 2 (Feb. 1980) pp.217-227.

[19] Baugher, Mark, French, Steven, Stephens, Alan, and Van Horn, Isabel, A Multimedia Client to the IBM LAN Server, *Proceeding of ACM Multimedia '93,* Anaheim CA (August 1993), pp. 105-112.

[20] Berson, Steven, Ghandeharizadeh, Shadram, Muntz, Richard, andXiangyu, Ju, Staggered Striping in Multimedia Information Systems, *SIGMOD 94 - 5/94 Minneapolis, Minnesota, USA*, pp. 79-90.

[21] Bially, Theodore, Mclaughlin, Alan J. and Weinstein, Clifford J., Voice Communications in Integrated Digital Voice and Data Networks, *IEEE Transactions on Communications* Vol.28, No. 9 (Sept.) pp. 1478-1488.

[22] Chang, Ed, and Zakhor, Avideh, Admissions Control and Data Placement for VBR Video Servers, *1st IEEE International Conference on Image Processing (Anaheim, Nov. 1994)*, pp. 208-221.

[23] Chang, Ed, and Zakhor, Avideh, Scalable Video Data Placement on Parallel Disk Arrays,*IS&T/SPIE International Symposium on Electronic Imaging: Science and Technology, Vol. 2185: Image and Video Databases II,* (Feb. 1994), pp. 208-221.

[24] Chen, Ming-Syan, Kandlur, Dilip D., and Yu, Philip S., Support For Fully Interactive Playout in a Disk-Array-Based Video Server, In *Proceedings of the ACM Multimedia '94,* San Francisco, ACM Press, New York, Oct. 1994.

[25] Chen, Mon-Song, Kandlur, Dilip D., and Yu, Philip S., Optimization of the Grouped Sweeping Scheduling (GSS) with Heterogeneous Multimedia Streams, *Proceedings of ACM Multimedia 93 (Anaheim, CA, August 1-6, 1993),* Association for Computing Machinery, New York, pp. 235-242.

[26] Chen, Thomas M., Messerschmitt, David G., Integrated Voice/Data Switching, *IEEE Communications* Vol. 26, No. 6 (June 1988) pp. 16-26.

[27] Christodoulakis, S., Ho, F., Theodoridou, M., The Multimedia Object Presentation Manager of MINOS: A Symmetric Approach, *ACM Proceedings of SIGMOD 1986.*

[28] Christodoulakis, S., Theodoridou, M., Ho, F., Papa, M., and Pathria, A., Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System, *ACM Transactions on Office Information Systems*, Vol. 4, No. 4 (Oct. 1986) pp. 345-383.

[29] Clark, David R., The Demise of Multimedia, *IEEE Computer Graphics and Applications,* July 1991, pp. 75-80

[30] Ferrari, Domenico, Guaranteeing Performance for Real-Time Communication in Wide-Area Networks, *Technical Report No. UCB/CSD 88/485, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Dec. 1988.

[31] Ferrari, Domenico, and Verma, D., A Scheme For Real-Time Channel Establishment in Wide-Area Networks, *IEEE Journal on Selected Areas in Communications*, No. 3 (April 1990), pp. 368-379.

[32] Fox, Edward A., Advances in Interactive Digital Multimedia Systems, *Computer*, Vol. 24, No. 10 (Oct. 1991) pp. 9-21.

[33] Frank, H., Analysis and Optimization of Disk Storage Devices for Time-Sharing Systems.*J. of the ACM*, Vol. 16, No. 4, (October 1969), pp. 602-620.

[34] Gemmell, D. James, Multimedia Network File Servers: Multi-Channel Delay Sensitive Data Retrieval, *Proceedings of ACM Multimedia '93,* Anaheim, CA (August 1993), pp. 243-250..

[35] Gemmell, D. James, *Principles of Storage and Retrieval of Delay Sensitive Multimedia Data,* Master's thesis, University of Waterloo, 1990.

[36] Gemmell, D. James, Beaton, Richard, Han, Jiawei, and Christodoulakis, Stavros, Delay Sensitive Multimedia on Disks, *IEEE Multimedia,* Vol. 1, No. 3 (Fall 1994), pp. 56-67.

[37] Gemmell, D. James, and Christodoulakis, Stavros, Principles of Delay Sensitive Multimedia Data Retrieval, *In Proceedings of the International Conference on Multimedia Information Systems '91 (Singapore),* McGraw-Hill, New York, 1991, pp. 147-158.

[38] Gemmell, D. James, and Christodoulakis, Stavros, Principles of Delay Sensitive Multimedia Data Storage and Retrieval, *ACM Transactions on Information Systems,* vol. 10, no. 1 (Jan. 1992) pp. 51-90.

[39] Gemmell, D. James and Han, Jiawei, Multimedia Network File Servers: Multi-Channel Delay Sensitive Data Retrieval, *Multimedia Systems,* Vol. 1, No. 6 (1994), pp. 240-252.

[40] Gerla, Mario, Kleinrock, Leonard, Flow Control: A Comparative Survey, *IEEE Transactions on Communications*, Vol. 28, No. 4 (April 1980), pp. 553-574.

[41] Gibbs, S., Tsichritzis, D., Fitas, A., Konstantas, D. and Yeorgaroudakis, Y., Muse: A Multimedia Filing System, *IEEE Software,* Vol. 4, No. 2 (March 1987), pp. 4-15..

[42] Gibbs, Simon, Breiteneder, Christian, and Tsichritzis, Dennis, Data Modeling of Time-Based Media, *Proceedings of SIGMOD 94 (May 1994)*, pp. 91-102.

[43] Gilge, Michael, and Gusella, Riccardo, Motion Video Coding for Packet-Switching Networks - An Integrated Approach, *Tech. Report TR-91-065, International Computer Science Institute 1947 Center Street, Suite 600, Berkeley, California, Dec. 1991.*

[44] Glass, L. Brett, Digital Video Interactive, *BYTE,* May 1989, pp. 283-289.

[45] Govindan, Ramesh, and Anderson, David P., Scheduling and IPC Mechanisms for Continuous Media, *Technical Report No. UCB/CSD 91/622, Computer Science Div., EECS Dept., Univ. ofCalif. at Berkeley,* March 1991.

[46] Gray, Jim, Horst, Bob and Walker, Mark, Parity Striping of Disc Arrays: Low-cost Reliable Storage with Acceptable Throughput, *Proceedings of the 16th VLDB Conference,* Brisbane, Australia, 1990, pp. 148-161.

[47] Griffiths, M. and Bloom, P.J., A Flexible Digital Sound-Editing Program for Minicomputer Systems, *J. Audio Eng. Soc.*, Vol. 30, No. 3., (March 1982) pp. 127-134.

[48] Grusec, Ted, Thibault, Louis, and Beaton, Richard J., Sensitive Methodologies For The Subjective Evaluation Of High Quality Audio Coding Systems, *Proceedings of the Audio Engineering Society UK DSP Conference,* (London) September, 1992, pp. 35-57.

[49] Haskin, R., The SHARK Continuous Media File Server, *Proceeding of CompCon,* (1993), pp. 12-15.

[50] Haus, Goffredo, Music Processing At L.I.M., *Microprocessing and Microprogramming* 24 (1988) 435-442.

[51] Homsy, George, Govindan, Ramesh, and Anderson, David P., Implementation Issues For a Network Continuous-Media I/O Server, *Technical Report No. UCB/CSD 90/597, Computer Science Div., EECS Dept., Univ. of Calif. at Berkeley,* Dept. 1990.

[52] *IEEE Communications Magazine,* Feature topic on video on demand, Vol. 32, No. 5, May 1994.

[53] Ingebretsen, Robert B., Stockham, Thomas G. (jr.), Random-Access Editing of Digital Audio, *J. Audio Eng. Soc.*, Vol. 32, No. 3, (March 1984).

[54] Irven, Judith, Nilson, Margaret E., Judd, Thomas H., Patterson, John F., Shibata, Yoshitaka, Multi-Media Information Services: A Laboratory Study, *IEEE Communications*, Vol. 26., No. 6 (June 1988), pp. 27-44.

[55] Kamel, Ragui, Emami, Kamyar, and Eckert, Robert, PX: Supporting Voice in Workstations, *Computer*, Vol. 23, No. 8 (Aug. 1990) pp. 73-80.

[56] Kim, K. H. and Naghibzadeh, Mahmoud, Prevention of Task Overruns in Real-Time Non-Preemptive Multiprogramming Systems, *Performance Eval. Review (USA),* Vol. 9, No. 2 (Summer 1980) pp.267-276 (*Proceedings of Performance 80,* Toronto, May 28-30 1980).

[57] King, Richard P., Disk Arm Movement in Anticipation of Future Requests, *ACM Transactions on Computer Systems,* Vol. 8, No. 3 (Aug. 1990), pp. 214-229.

[58] Knuth, D. E., The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Addison-Wesley, Don Mills, Ont., 1973.

[59] Leffler, Samuel J., McKusick, Marshall K., Karels, Michael, J., and Quarterman, John S., The Design and Implementation of the 4.3BSD UNIX Operating System, Addison-Wesley, Don Mills, Ontario, 1989.

[60] Leung, Wu-Hon F., Baumgartner, Thomas J., Hwang, Yeou H., Morgan, Mike J. and Tu, Shi-Chuan, A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks, *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3 (April 1990) pp. 380-390.

[61] Levergood, Thomas M., Payne, Andrew C., Gettys, James, Treese, G. Winfield, and Stewart, Lawrence C., AudioFile: A Network-Transparent System for Distributed Audio Applications, pre-print from *Proceedings of the USENIX Summer Conference,* June, 1993.

[62] Little, Thomas D. C., Ghafoor, Arif, Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks, *Computer*, Vol. 24, No. 10 (October 1991), pp. 42-50.

[63] Liu, C. L and Layland, James W., Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *J. of the ACM*, Vol. 29., No. 1, (Jan 1973) pp. 46-61.

[64] Lougher, Phillip and Shepherd, Doug, The Design of a Storage Server for Continuous Media, *The Computer Journal (special issue on multimedia)*, Vol. 36, No. 1 (Feb. 1993), pp.32-42.

[65] Ludwig, L. F., Pincever, N., and Cohen, M., Extending the Notion of a Window System to Audio, *Computer*, Vol. 23, No. 8 (Aug. 1990), pp. 66-72.

[66] MacKay, W. E., and Davenport, G., Virtual Video Editing in Interactive Multimedia Applications, *Communications of the ACM*, Vol. 32, No. 7 (July 1989), pp. 802-810.

[67] Malek, Manu, Integrated Voice and Data Communications Overview, *IEEE Communications*, Vol. 25, No. 6, (June 1988), pp. 5-15.

[68] McKusick, M. K., Joy, W. N., Leffler, S. J., and Fabry, R. S., A Fast File System For UNIX, *CSRG Tech Rep. 83/147, Berkeley*, July 27, 1983.

[69] McNally, G. W., and Gaskell, P. S., Editing Digital Sound, *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, IEEE, 1984, pp. 12B.4.1 - 12B.4.4.

[70] Meghini, Carlo, Rabitti, Fausto, and Thomas, Constantino, Conceptual Modeling of Multimedia Documents, *Computer*, Vol. 24, No. 10 (October 1991) pp. 23-30.

[71] Microsoft Unveils Video Software, *AP News*, May 17, 1994.

[72] Moran, Mark and Wolfinger, Bernd, Design of a Continuous Media Data Transport Service and Protocol, *Tech. Report TR-92-019, International Computer Science Institute 1947 Center Street, Suite 600, Berkeley, California, April 1992.*

[73] Mori, T., Nishimura, K., Nakano, H. and Ishibashi, Y., Video-on-demand System Using Optical Mass Storage System. *Japanese Journal of Applied Physics*, Vol. 1, No. 11B (Nov. 1993), pp. 5433-5438.

[74] Narasimhalu, A. Desai, and Christodoulakis, Stavros, Multimedia Information Systems: The Unfolding Reality, *Computer*, Vol. 24, No. 10 (Oct. 1990), pp. 6-8.

[75] Ng, Spencer W., Improving Disk Performance Via Latency Reduction, *IEEE Transactions on Computers*, Vol. 40, No. 1, (Jan 1991) pp.22-30.

[76] Nicolaou, Cosmos, An Architecture for Real-Time Multimedia Communication Systems, *IEEE Journal on Selected Areas in Communications*, Vol. 8, No. 3 (April 1990) pp. 391-400.

[77] Ousterhout, John and Douglis, Fred, Beating the I/O Bottleneck: A Case for Log-Structured File Systems, *Operating Systems Review*, Vol. 23, No. 1 (Jan 1989), pp.11-28.

[78] Park, A., and English, P., A Variable Rate Strategy for Retrieving Audio Data From Secondary Storage, *In Proceedings of the International Conference on Multimedia Information Systems '91 (Singapore)*, McGraw-Hill, New York, 1991, pp. 135-146.

[79] Parris, Colin, and Ferrari, Domenico, A Resource Based Pricing Policy For Real-Time Channels In A Packet-Switching Network, *Tech. Report, International Computer Science Institute 1947 Center Street, Suite 600, Berkeley, California.*

[80] Patterson, D., Gibson, G., and Katz, R., A Case for Redundant Array of inexpensive Disks (RAID), *ACM SIGMOD '88*, June 1988, pp. 109-116.

[81] Pohlmann, Ken C., Principles of Digital Audio, Knowledge Industry Publications Inc. White Plains, NY, London, 1985.

[82] Polimenis, Vassilios G., The Design of a File System That Supports Multimedia, *Tech. Report TR-91-020, International Computer Science Institute 1947 Center Street, Suite 600, Berkeley, California, March, 1991.*

[83] Quarterman, John S., Silberschatz, Abraham, and Peterson, James L., 4.2BSD and 4.3BSD as Examples of the Unix System, *Computing Surveys*, Vol. 17, No. 4, December 1985, pp. 379-418.

[84] Ramamritham, K. and Stankovic, J. A., Dynamic Task Scheduling in Distributed Real Time Systems, *IEEE Software,* July 1984, pp. 65-75.

[85] Ramanathan, Srinivas, and Rangan, P. Venkat, Adaptive Feedback techniques for Synchronized Multimedia Retrieval over Integrated Networks, *IEEE/ACM Transactions on Networking,* Vol. 1, No. 2 (April 1993), pp. 246-260.

[86] Ramanathan, Srinivas, and Rangan, P. Venkat, Continuous Media Synchronization in Distributed Multimedia Systems, *Proceedings of the Third International Workshop on Networking and Operating System Support for Digital Video and Audio,* San Diego, California, November, 1992.

[87] Ramanathan, Srinivas, and Rangan, P. Venkat, Feedback techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems, *The Computer Journal, Special Issue on Distributed Multimedia Systems,* Vol. 36, No. 1 (January 1993), pp.19-31.

[88] Ramanathan, Srinivas, and Rangan, P. Venkat, System Architecture for Personalized Multimedia Services, *IEEE Multimedia,* Vol. 1, No. 1 (Feb. 1994), pp. 37-46.

[89] Ramanathan, Srinivas, and Rangan, P. Venkat, Techniques for Continuous Retrieval of Multimedia Over High-speed Networks, *Proceedings of the First International Conference on Computer Communications and Networks,* San Diego, California, June 1992.

[90] Ramanathan, Srinivas, Rangan, P. Venkat, and Vin, Harrick M., Designing Communication Architecture for Inter-Organizational Multimedia Collaboration, *Journal of Organizational Computer,* Vol. 2, No. 3-4 (1992), pp. 277-302.

[91] Ramanathan, Srinivas, Rangan, P. Venkat, and Vin, Harrick M., Integrating Virtual Reality, Tele-Conferencing, and Entertainment into Multimedia Home Computers, *IEEE Transactions on Consumer Electronics,* Vol. 38, No. 2 (May 1992), pp. 70-76.

[92] Ramanathan, Srinivas, Rangan, P. Venkat, Vin, Harrick M., and Kaeppner, Thomas, Optimal Communication Architectures for Multimedia Conferencing in Distributed Systems, *Proceedings of the 12th International Conference on Distributed Computer Systems (ICDCS '92),* Yokohama, Japan, June 1992.

[93] Rangan, P. Venkat, Research Contributions of The UCSD Multimedia Laboratory, *UCSD Technical Report,* February 1993.

[94] Rangan, P. Venkat, Software Implementation of VCRs On Personal Computing Systems, *Proceedings of the International Conference on Consumer Electronics,* Chicago, Illinois, June 1992, pp. 635-640.

[95] Rangan, P. Venkat, Video Conferencing, File Storage, and Management in Multimedia Computer Systems, *Computer Networks and ISDN Systems,* Vol. 25 (March 1993), pp. 901-919.

[96] Rangan, P. Venkat, Burjhard, W., Bowdidge, R., Vin, Harrick M., Lindwall, J. W., Chan, K., Aeberg, I. A., Yamamoto, L. M., and Harris, I. G., A testbed for Managing Digital Video and Audio Storage, in Proceeding of *Multimedia - for now and the future, USENIX Summer Conference,* Nashville, Tennessee, June 10-14, 1991, pp. 199-208.

[97] Rangan, P. Venkat, Kaeppner, Thomas, and Vin, Harrick M., Techniques for Efficient Storage of Digital Video and Audio, *Proceedings of Multimedia Information Systems '91,* Phoenix, Arizona, February 1992.

[98] Rangan, P. Venkat, Ramanathan, Srinivas, Vin, Harrick M., and Kaeppner, Thomas, Media Synchronization in Distributed Multimedia File Systems, *Proceedings of Multimedia '92,* Monterey, California, April 1992.

[99] Rangan, P. Venkat, Ramanathan, Srinivas, Vin, Harrick M., and Kaeppner, Thomas, Techniques for Media Synchronization in Network File Systems, *Computer Communications Journal,* Vol. 16, No. 3 (March 1993), pp. 168-176.

[100] Rangan, P. Venkat and Swinehart, D. C., Software Architecture for Integration of Video Services in the Etherphone Environment, *IEEE Journal on Selected Areas in Communication,* Vol. 9, No. 9, (Dec. 1991) pp. 1395-1404.

[101] Rangan, P. Venkat and Vin, Harrick M., A Unified Framework for Modeling Synchronous and Asynchronous Multimedia Collaborations, *Proceedings of the First Workshop on Enabling Technologies for Concurrent Engineering,* Morgantown, West Virginia, April 1992.

[102] Rangan, P. Venkat and Vin, Harrick M., Designing File Systems for Digital Video and Audio, in *Proceedings of the 13th ACM Symposium on Operating Systems Principles (SOSP '91),* Operating Systems Review, Vol. 25, No. 5 (Oct. 1991), pp. 81-94.

[103] Rangan, P. Venkat and Vin, Harrick M., Efficient Storage Techniques for Digital Continuous Multimedia, *IEEE Transactions on Knowledge and Data Engineering,* Vol. 5, No. 4, pp. 564-573.

[104] Rangan, P. Venkat and Vin, Harrick M., Multimedia Conferencing as A Universal Paradigm for Collaboration, In *Multimedia - Principles, Systems and Applications,* Chapter 14, Springer-Verlag, April 1991.

[105] Rangan, P. Venkat, Vin, Harrick M., Chen, K. and Aeberg, I, A Window-based Editor for Digital Video and Audio, In Proceeding of the *25th International Conference on System Sciences, Hypermedia and Multimedia mini-track,* Kauai, Hawaii, January 7-10, 1992, pp.640-648.

[106] Rangan, P. Venkat, Vin, Harrick M., and Ramanathan, Srinivas, Communication Architectures and Algorithms for Media Mixing in Multimedia Conferences, *IEEE/ACM Transactions on Networking*, Vol. 1, No. 1 (Feb. 1993), pp. 20-31.

[107] Rangan, P. Venkat, Vin, Harrick M., and Ramanathan, Srinivas, Designing an On-demand Multimedia Service, *IEEE Communications Magazine,* Vol. 30, No. 7 (July 1992), pp. 56-64.

[108] Reddy, A. L. Narasimha, and Wyllie, Jim, Disk Scheduling in a multimedia I/O system, *Proceedings of ACM Multimedia 93 (Anaheim, CA, August 1-6, 1993),* Association for Computing Machinery, New York, *pp. 225-233.*

[109] Reddy, A. L. Narasimha, and Wyllie, J., I/O Issues in a Multimedia System, *Computer,* Vol. 27. No. 3 (1994), pp.69-74.

[110] Redel, David D., Dala, Yogen K., Horsley, Thomas R., Lauer, Hugh C., Lynch, William C., McJones, Paul R., Murray, Hal G., and Purcell, Stephen C., Pilot: An Operating System for a Personal Computer, *Communications of the ACM,* Vol. 23, No. 2 (February 1980), pp. 81-92.

[111] Ross, Douglas T., The AED Free Storage Package, *Communications of the ACM,* Vol. 10, No. 8 (August 1967), pp. 481-492.

[112] Ruemmler, Chris, and Wiles, John, An Introduction to Disk Drive Modeling, *Computer*, Vol. 27, No. 3 (March 1994), pp. 17-28.

[113] Shepherd, Doug, and Salmony, Michael, Extending OSI to support synchronization required by multimedia applications, *Computer Communications,* Vol. 13, No. 7 (Sept. 1990), pp. 399-406.

[114] Staehli, Richard, and Walpole, Jonathan, Constrained-Latency Storage Access, *COMPUTER,* Vol. 26, No. 3 (March 1993), pp.44-53.

[115] Strathmeyer, Carl R., Voice in Computing: An Overview of Available Technologies, *Computer*, Vol. 23, No. 8 (Aug. 1990), pp. 10-15.

[116] Tanenbaum, Andrew S., Modern Operating Systems, Prentice Hall, Englewood Cliffs, N.J., 1992.

[117] Tangi, M. et al, Video and Audio Disk File System, SPIE Vol. 529 Optical Mass Data Storage (1985), pp. 116-125.

[118] Teorey, T. J., Fry, J. P., Design of Database Structures, Prentice-Hall Inc., Englewood Cliffs, N.J. 1982.

[119] Teorey, Toby J. and Pinkerton, Tad B., A Comparative Analysis of Disk Scheduling Policies, *Communications of the ACM*, Vol. 15, No. 3 (March 1972) pp. 177-184.

[120] Terenzi, Firorella, The Computer Audio Research Laboratory at The Center for Music Experiment and Related Research, *Microprocessing and Microprogramming* 24 (1988) 443-446.

[121] Terry, D. B., and Swinehart, D. C., Managing Stored Voice in the Etherphone System, *Trans. Computer Systems*, Vol. 6, No. 1 (Feb. 1988), pp. 3-27.

[122] Thomas, R. H., Forsdick, H. C., Crowley, T. R., Schaaf, R. W., Tomlinsin, R. S., Travers, V. M., and Robertson, G. G., Diamond: A Multimedia Message System Built on a Distributed Architecture, *Computer*, Vol. 18, No. 12 (Dec. 1985), pp. 65-78.

[123] Tobagi, F. A., Pang, J., Baird, R., and Gang, M., Streaming RAID: A Disk Storage System for Video and Audio Files, *Proceedings of ACM Multimedia '93, Anaheim, CA,* August 1993, pp. 393-400.

[124] Tzou, S. and Anderson, D. P., A Performance Evaluation of the DASH Message-Passing System, *Tech. Rep. UCB/CSD 88/452, Computer Science Div., EECS Dept., University of California at Berkeley*, Nov. 1988.

[125] Verma, Dinseh Chandra, Guaranteed Performance Communication in High Speed Networks, *Tech. Rep. UCB/CSD 91/663, Computer Science Div., EECS Dept., University of California at Berkeley*, Dec. 11, 1991.

[126] Vin, Harrick M., Goyal, Alok, Goyal, Anshuman, and Goyal, Pawan, An Observation-Based Admission Control Algorithm for Multimedia Servers. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS '94)*, Boston, May 1994.

[127] Vin, Harrick M., Goyal, Pawan, Goyal, Alok, and Goyal, Anshuman, A Statistical Admission Control Algorithm for Multimedia Servers. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS '94)*, Boston, May 1994.

[128] Vin, Harrick M. and Rangan, P. Venkat, Admission Control Algorithms for Multimedia On-Demand Servers, *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, California, November 1992, pp. 50-62.

[129] Vin, Harrick M. and Rangan, P. Venkat, Designing a Multi-User HDTV Storage Server, to appear in *IEEE Journal on Selected Areas in Communications - Special Issue on High Definition Television and Digital Video Communications*, Vol. 11, No. 1 (January 1993).

[130] Vin, Harrick M., Rangan, P. Venkat, and Chen, M. S., System Support for Computer Mediated Multimedia Collaborations, In *Proceedings of ACM 1992 Conference on Computer Supported Cooperative Work (CSCW '92)*, Toronto, Canada, November 1-4, 1992.

[131] Vin, Harrick M., Rangan, P. Venkat, and Ramanathan, Srinivas, Hierarchical Conferencing Architectures for Inter-Group Multimedia Collaboration, *Proceedings of the Conference on Organizational Computer Systems (COCS '91)*, SIGOIS Bulletin, Vol. 12, No 2-3 (November 1992) pp. 43-45.

[132] Vin, Harrick M., Zellweger, Polle T., Swinehart, Daniel C., and Rangan, P. Venkat, Multimedia Conferencing in the Etherphone Environment. *Computer*, Vol. 24, No. 10 (Oct. 1991), pp. 69-79.

[133] Watkinson, John, Digital Audio Recorders, *J. Audio Eng. Soc.*, Vol. 36, No. 6, (June 1988), pp. 502-506.

[134] Watkinson, John, *The Art Of Digital Audio*, Focal Press, Boston, Mass., 1988.

[135] Wells, J., Tang, Q., and Yu, C., Placement of audio data on optical disks, *In Proceedings of the International Conference on Multimedia Information Systems '91 (Singapore)*, McGraw-Hill, New York, 1991, pp. 123-134.

[136] Woodruff, G. M. and Kostpaiboon, R., Multimedia Traffic Management Principles for Guaranteed ATM Network Performance, *IEEE J. Selected Areas in Comm.*, Vol. 8, No. 3, Apr. 1990, pp. 437-446.

[137] Yu, Clement, Sun, W., Bitton, D., Yang, Q., Bruno, R., Tullis, J., Efficient Placement of Audio Data on Optical Disks for Real-Time Applications, *CACM* 32, 7, 1989, pp. 862-871.

[138] Yu, Philip S., Chen, Mon-Song, and Kandlur, Dilip, D., Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management, *Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, November 12-13, 1992.

[139] Zhao, W., Ramamritham, K., and Stankovic, J. A., Preemptive Scheduling Under Time and Resource Constraints, *IEEE Transactions on Computers*, Vol. C-36, No. 8 (August 1987), pp. 949-960.