

## **INFORMATION TO USERS**

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600

**UMI<sup>®</sup>**



## **NOTE TO USERS**

**This reproduction is the best copy available.**

UMI



RICE UNIVERSITY

**Real-Time Prefetching and Buffer Management  
for Parallel Multimedia I/O Systems**

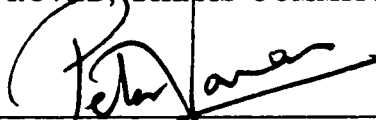
by

**Özgür Ertuğ**

A THESIS SUBMITTED  
IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE

**Master of Science**

APPROVED, THESIS COMMITTEE:



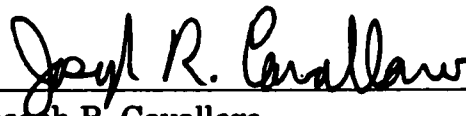
---

Peter J. Varman, Chairman  
Associate Professor in Electrical and  
Computer Engineering



---

Richard Baraniuk  
Professor in Electrical and Computer  
Engineering



---

Joseph R. Cavallaro  
Associate Professor in Electrical and  
Computer Engineering

Houston, Texas

September, 2000

**UMI Number: 1405662**



---

**UMI Microform 1405662**

**Copyright 2001 by Bell & Howell Information and Learning Company.**

**All rights reserved. This microform edition is protected against  
unauthorized copying under Title 17, United States Code.**

---

**Bell & Howell Information and Learning Company  
300 North Zeeb Road  
P.O. Box 1346  
Ann Arbor, MI 48106-1346**

# **Real-Time Prefetching and Buffer Management for Parallel Multimedia I/O Systems**

**Özgür Ertuğ**

## **Abstract**

Continuous media servers are increasingly used to support a number of application domains, e.g., entertainment industry, library information systems, educational applications etc. Objects of continuous media data type are large in size and their retrieval and display are subject to real-time constraints. Servers are required to accommodate these objects and ensure their continuous display. In this thesis, we introduce a model for resource scheduling of a video storage server delivering continuous media VBR video data with real-time requirements. The video streams are assumed to be stored in CDL format and distributed across multiple disks. Within a server-network-client model, our framework translates the requirements imposed by video and resource availability into constraints on prefetching in the real-time domain.

We present a novel algorithm RT-OPT for optimally prefetching blocks into the server buffer. We show that if the schedule created by RT-OPT fails to meet the deadline of any block, then no feasible schedule is possible for the same buffer size, data placement and single-disk scheduling policy. Simulations with MPEG traces show that RT-OPT achieves high scalability by dynamically multiplexing the buffer among different clients and disks optimally. The number of clients supported is shown to be uniformly superior to intuitive but suboptimal algorithms like GREED-EDF that aggressively keep the disks busy fetching in order of deadlines.

## **Acknowledgments**

I would like to start by thanking to my parents for their support throughout my graduate studies. Without their guidance and nurture, I would not have been able to complete this part of the journey.

I would like to thank my advisor, Professor Peter Varman, for his guidance throughout the course of this thesis. He has always provided me with another point of view from which to tackle problems. His moral motivation was the biggest support I had that enabled me to complete this road. I also would like to express my appreciation to Professor Joseph Cavallaro and Professor Richard Baraniuk for serving on my thesis committee. I would also like to thank my sponsor, Texas Instruments, for their financial support during my graduate studies.

Finally, I would like to thank my friend, Mahesh Kallahalla, whom I have bounced ideas off, and received insightful comments and helps which speeded up my research.



# Contents

Abstract	ii
Acknowledgments	iii
List of Illustrations	vi
<b>1 Introduction</b>	<b>1</b>
1.0.1 Continuous Media Server Overview . . . . .	5
1.0.2 Related Work . . . . .	7
1.0.3 Contribution . . . . .	8
1.0.4 Thesis Outline . . . . .	9
<b>2 Fundamentals of Continuous Media Display and Magnetic Disk Drives</b>	<b>10</b>
2.1 Continuous Media Display Overview . . . . .	10
2.1.1 Target Environment . . . . .	11
2.1.2 Modern Disk Drives . . . . .	12
2.1.3 Internal Operation . . . . .	13
2.1.4 Disk Drive Modeling . . . . .	14
<b>3 Real-Time Model</b>	<b>16</b>
<b>4 Algorithm RT-OPT</b>	<b>22</b>
<b>5 Simulations</b>	<b>28</b>
<b>6 Conclusions</b>	<b>32</b>

<b>Bibliography</b>	<b>33</b>
<b>A MPEG Overview</b>	<b>37</b>
A.0.5 Systems . . . . .	37
A.0.6 Video Hierarchical Structure . . . . .	39
A.0.7 Video Compression Algorithm . . . . .	44

# Illustrations

1.1	Video-on-demand server-network-client system architecture . . .	2
1.2	CDL and CTL storage schemes . . . . .	4
1.3	Focused Continuous Media Server Architecture . . . . .	6
2.1	Continuous Display of Multiple Objects . . . . .	11
2.2	Storage Subsystem Architecture . . . . .	12
2.3	Disk Drive Internals . . . . .	14
3.1	Server Model . . . . .	17
3.2	Time line of a Block . . . . .	18
3.3	Algorithm GREED-EDF . . . . .	20
3.4	Illustration of GREED-EDF and RT-OPT . . . . .	20
4.1	Initial step of Algorithm RT-OPT . . . . .	23
4.2	RT-OPT following the removal of disk-access conflicts . . . . .	23
4.3	RT-OPT following the removal of buffer conflicts . . . . .	24
4.4	Algorithm RT-OPT . . . . .	25
5.1	Number of Users Serviced vs. Number of Disks . . . . .	29
5.2	Number of Clients Serviced at Zero Dropping Probability vs. Buffer Size . . . . .	30
5.3	Number of Users Serviced vs. Drop Probability . . . . .	31
A.1	Flow of the MPEG System Stream . . . . .	38

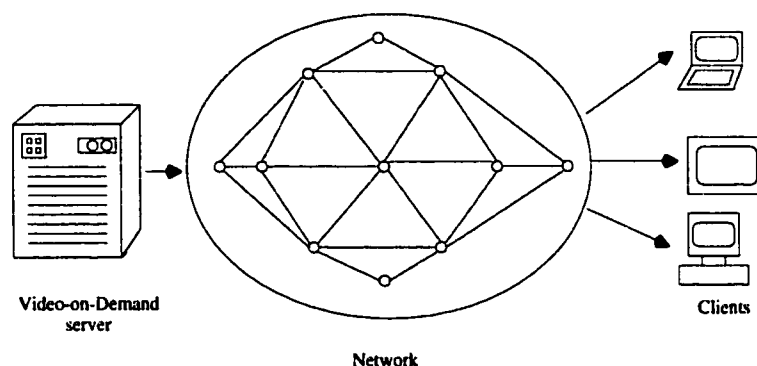
<b>A.2 Video Hierarchical Structure . . . . .</b>	<b>39</b>
<b>A.3 Common Pattern of Group of Pictures . . . . .</b>	<b>40</b>
<b>A.4 Group of Pictures Order . . . . .</b>	<b>41</b>
<b>A.5 An Example of MPEG Picture Sizes . . . . .</b>	<b>42</b>
<b>A.6 Prediction Processes . . . . .</b>	<b>43</b>
<b>A.7 Chrominance Format for Macroblocks . . . . .</b>	<b>44</b>
<b>A.8 MPEG Video Compression Algorithm . . . . .</b>	<b>45</b>

# **Chapter 1**

## **Introduction**

For decades, analog technology was used in the production of audio and video. Audio and video systems throughout the world are witnessing a transition to digital video technology . Advances in computer and communications technologies have stimulated the integration of digital continuous media with computing resulting in multimedia systems. Multimedia systems will revolutionize current life styles, especially those aspects associated with education, entertainment, medicine and commerce. To illustrate, in distance learning, a multimedia system may store lectures, educational videos, as well as an entire library thus enabling schools, universities and businesses to share a wide range of educational materials. An entertainment system may store a rich set of movies, sitcoms and video games, and permit users to view or play any of these videos and games on demand. Such systems are typically termed video-on-demand (VOD) servers. The effective support of continuous media is essential to the success of multimedia systems. This crucial task is performed by continuous media servers, thus making them an indispensable part of multimedia systems. Continuous media servers consist of three major components: continuous media objects, continuous media storage and the retrieval scheduler.

Continuous media objects are typically large in size. For example, digital component video based on CCIR-601 standard requires 270 Megabits per second for its continuous display. If a continuous media server delivers a clip at a rate lower than its prespecified rate without special precautions, the user might observe frequent disruptions and delays with video or random noise with audio. These artifacts are collectively termed hiccups. Due to redundancy in data, the bandwidth requirement of a clip along with its size can be reduced



**Figure 1.1 : Video-on-demand server-network-client system architecture**

by using compression techniques such as digital video interactive (DVI), joint photographic expert group (JPEG) and motion pictures expert group (MPEG).

Video-on-Demand (VOD) systems are distributed server-network-client architectures (Figure 1.1) that provide interactive or non-interactive video service to multiple clients with specified quality of service (QoS) guarantees in real-time. The video server that stores the video data and delivers it to the network consists of either a high-volume single disk or a parallel disk array [8] on which the data of different video clips are distributed in some fashion. Video data is a three-dimensional digital signal with two dimensions in the spatial domain and one dimension in the temporal domain. It is composed of a sequence of frames, each consisting of  $N \times M$  pixels, ordered in time and regularly spaced by the inverse of the frame rate. Typical frame rates in use today are 24 and 30 frames per second [28]. Raw video data requires an enormous amount of storage (megabytes per second of video) and a correspondingly large time to transmit across the network to a client. Consequently, the data is usually stored in compressed form on magnetic storage at the server using either MPEG-1 or MPEG-2 video compression formats [12, 13, 28, 29].

MPEG (Motion Picture Experts Group) (see Appendix A) video compression format is one of the widely used compression formats today. The MPEG compression algorithm uses three types of frame coding to remove the spatial and temporal correlations in the video sequence. The I or intra-coded se-

quences are coded to remove the spatial correlation within the frame by DFT (Discrete Fourier Transform), quantization and run-length entropy coding of the two-dimensional pixel domain, as in JPEG (Joint Pictures Expert Group) image compression format [29]. The motion correlation between consecutive frames is used to remove the temporal redundancy using predictive motion-based inter-frame decorrelation and compression. The P and the B frames are the predictively inter-coded frames. The P frames are inter-coded with respect to the previous I or P frame, and the B frames are motion-compensated with an interpolation of the previous and the next I or P frame. The coded frame sizes of the frame types are in general I, P and B in decreasing order. The frames are coded in a specific order repeating continually along the video sequence; the repeating unit called GOP (group of pictures), such as “IBBPBB” or “IBBPBBPBBPBB” [29].

The video data compressed in MPEG format inherently results in a variable bit rate (VBR) stream. VBR coded video results in improved picture quality over a constant-bit-rate encoding (CBR), but places additional stress on the server in meeting the real-time constraints of individual frames. The unit of storage of VBR coded bit streams on the disk and the units in which they are accessed are important factors in the design of the video server [6, 5].

Two main types of storage and retrieval schemes for VBR video are *constant data length* (CDL) and *constant time length* (CTL) (see Figure 1.2). In CDL storage scheme video data is stored and read in constant-size blocks. This simplifies disk storage allocation, avoids fragmentation, and eases buffer management by performing I/O in fixed-size units. However, CDL results in different number of frames stored in each block; therefore each block has a different playback duration in real-time. In CTL storage for VBR video, each block has a different size, but the playback duration for each block is the same since the number of frames in each block are equal. For flexibility in achieving guaranteed QoS, CDL is generally preferred [5] and many recent papers have dealt with issues in CDL-based retrieval [2, 3, 5, 6].

The network connecting the server and the geographically distributed

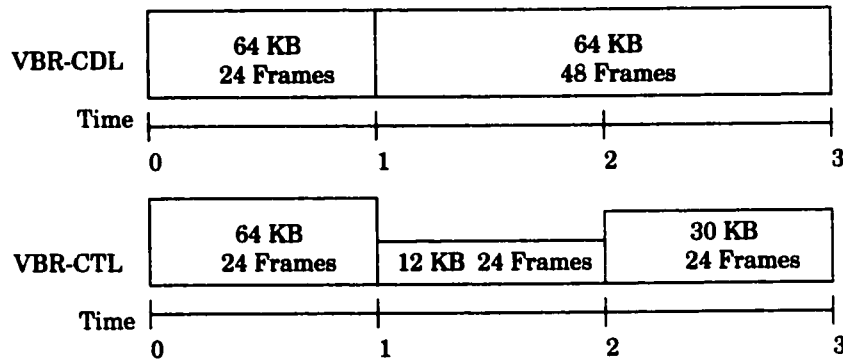


Figure 1.2 : CDL and CTL storage schemes

clients is in general a local area network (LAN) or a wide area network (WAN) based on ATM cell relay or IP packet-switching protocols; it may include other types of common shared mediums for broadband access such as cable and ADSL networks or satellite channels [24]. The server distributes the requested blocks to the clients over the network in real-time in compressed form to save transmission bandwidth, and upon reception the client decompresses and displays the frames in the received datagrams stored in its processing queue. The main goal of a VOD server is to store the large volume of video data from different video clips in its secondary storage and to deliver them to the clients in a timely manner by honoring specific QoS guarantees such as delay, delay jitter and maximum frame-loss probability. Given a fixed set of resources such as disk bandwidth, storage volume, and main memory for buffering, the video server can deliver a limited number of streams at the specified QoS guarantees. A design goal is to maximize the number of clients (streams) that can be concurrently served with the guaranteed QoS.

A VOD system incorporates several subsystems: a high-volume secondary storage subsystem for storage of video data, main memory for buffering, high-speed processors at the server, a high-speed network connecting the server and the clients, and client-end computing and display devices. The storage subsystem is in general the bottleneck for real-time delivery of video due to the high I/O latencies incurred by secondary storage devices. The use of multiple



disks to build a parallel I/O subsystem has been advocated to increase I/O performance. However harnessing the raw increased disk bandwidth afforded by multiple disks to decrease the latency seen by the individual blocks requires sophisticated prefetching and caching techniques [1, 18, 19, 32]. The problem is compounded by the necessity for real-time guarantees in a VOD server. In a parallel I/O system, idle disks can be used to prefetch video blocks concurrently with demand I/Os from other disks. These prefetched blocks are held in the buffer until their deadlines, when they can be transferred to the client. However, deciding which blocks to prefetch is a non-trivial task. Blocks that are prefetched long before their deadline occupy buffer space wastefully, reducing the effective buffer size, and decreasing the number of clients that the server can handle; on the other hand delaying a prefetch potentially wastes disk bandwidth, causing blocks downstream to miss their deadlines and violating the QoS guarantees. To use the increased disk bandwidth effectively, the design of prefetching and buffer management algorithms that ensure the most useful blocks are fetched and retained in the buffer is crucial.

### **1.0.1 Continuous Media Server Overview**

Continuous media is distinguished from traditional textual and record-based media in two ways. First the retrieval and display of continuous media are subject to real-time constraints. These real-time constraints affect not only the storage, retrieval scheduling and delivery of data through the interconnection network, but also the manner in which multiple users may share resources. If the real-time constraints are not satisfied, then the display may suffer from disruptions and delays. Second, objects of continuous media type are large in size. For instance, the size of the three minutes of uncompressed CD quality audio with a 1,4 Megabits-per-second bandwidth requirement, is 31.5 Megabytes. A two hour MPEG-2 encoded video with a 4 Mbps bandwidth requirement is 3.6 Gigabytes in size. During the past few years, several studies have investigated the design issues involved in a continuous media server [31, 33, 9, 7, 11]. Some of these studies have focused on a single-disk continuous-media server

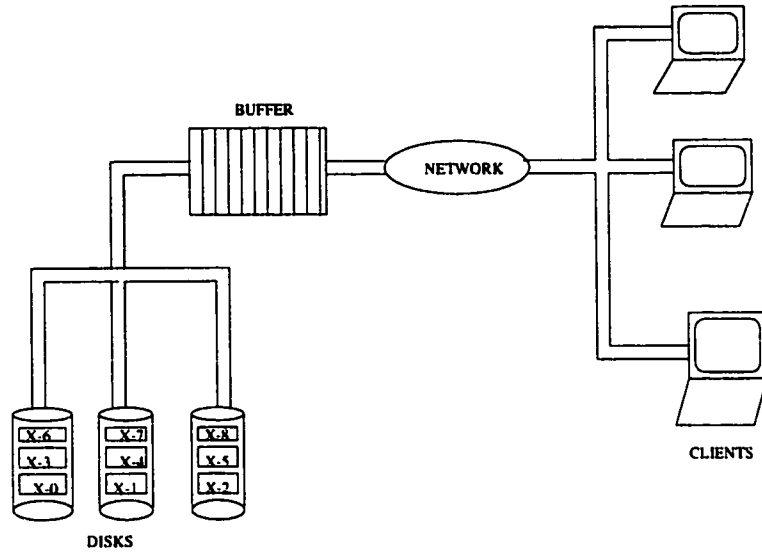


Figure 1.3 : Focused Continuous Media Server Architecture

architecture to support a high number of simultaneous displays accessing the server.

Figure 1.3 shows the continuous media server architecture that we focus on. As shown in the figure, a multi-disk architecture with round-robin striping of data blocks [26, 31] is a common approach that is used in media servers. We adopt the same data placement technique as well.

Scheduling in a continuous media server can be divided into two categories; deterministic and statistical. In traditional round-based deterministic scheduling, an object  $X$  is partitioned into  $n$  blocks  $X_0, X_1, \dots, X_{n-1}$  (Figure 1.3). A block is assumed to be the unit of transfer from the disk to main memory. The display time of a block is termed a time period. To support simultaneous display of several objects, a time period is divided into slots, with each slot corresponding to the retrieval time of a block from a disk. When a request references object  $X$ , the server stages  $X_0$  from its corresponding disk into memory and initiates its display. Prior to completion of a time period, it initiates the retrieval of  $X_1$  from the next disk into memory in order to ensure a continuous display. This process is repeated until all blocks of an object have been displayed.

In statistical scheduling, each client submits its block retrieval request to the operating system with a deadline that must be respected in order to support a hiccup-free display. Continuous media servers which use this kind of scheme, make probabilistic guarantees with regard to meeting the deadline. Disks in this scheme are not synchronized at the end of each time period. In this thesis, we focus on deterministic scheduling of variable bit rate continuous media with the design goal of maximizing the number of concurrent streams supported. However, unlike traditional round-based schemes, we globally schedule the streams, so that high demands for a client at some time can be traded off against low demand from another client dynamically.

### **1.0.2 Related Work**

There have been a number of different approaches for storage and retrieval of real-time video data. Disk-head scheduling algorithms for real-time multimedia scheduling of single-disk systems were analyzed in [22, 23]. Three algorithms SCAN, EDF and a hybrid of the two were evaluated for different buffer sizes. Non-contiguous disk allocation of streams was proposed and admission control based on constrained layout designs were analyzed in [20]. A disk-head scheduling algorithm, GSS (group sweeping scheme), with the objective of minimizing buffer space and access time was presented in [36]. For the SPIFFI video-server, [14] proposed a real-time priority-based disk scheduling algorithm that partitions requests into priority classes based on the nearness of their deadlines, and compares it with GSS. These algorithms are concerned with the scheduling of a single disk. For multiple-disk parallel disk arrays, simple reading of an entire stripe is proposed as a means of obtaining high parallelism. However, it is well known that such an approach is potentially wasteful of buffer memory, and an approach that schedules the multiple disks globally can perform significantly better [1, 18, 19, 32, 34]. However the scheduling algorithms proposed in these studies have no notion of real-time and cannot account for deadlines imposed by real-time requirements.

In [2, 3, 6, 5] all admission and resource allocation algorithms for VBR-

CDL video streams are studied in a single-disk system based on the concept of fetching in rounds. Based on desired QoS guarantees and the characteristics of the video streams, the buffer is partitioned among the different clients. The partitioning of the buffer memory simplifies the round-based admission control algorithms, but under utilizes the buffer by failing to multiplex the buffer dynamically. By assuming that the worst-case buffer requirements of the clients occur simultaneously, the number of clients that can be supported is greatly reduced. In [21], caching methods to reuse blocks of popular movies were proposed and analyzed. None of these works attempts to optimize the use of buffer in a multiple-disk situation among multiple disks nor, with the exception of [21], dynamically among multiple clients.

### **1.0.3 Contribution**

The main novelty of the work reported in this thesis is that in our framework, the buffer is dynamically multiplexed among the clients and disks to maximize its utilization. Such dynamic sharing greatly improves the performance of VBR-CDL systems since there is wide variance in the deadlines for different blocks. This variance causes the load to dynamically shift from one client to another and from one disk to some other. For any given amount of buffer memory and choice of low-level disk-arm scheduling on a single disk, our scheduler maximizes the number of client streams that can be serviced. We create an abstract model in which to pose the real-time scheduling problem precisely. Although our abstract problem may superficially resemble many problems solved in the real-time system community, none of these were found to apply to the particular problem of parallel disk scheduling. The specific differences in our requirements are the use of multiple resources (disk and buffer) and the binding of blocks to specific disks.

#### **1.0.4 Thesis Outline**

The rest of the thesis is organized as follows. Chapter 2 concentrates on hiccup-free continuous display of VBR video clips. Chapter 3 describes our real-time model for prefetching and buffer management within the parallel disk model. Chapter 4 describes the algorithm RT-OPT within our real-time framework for prefetching and buffer management. Chapter 5 provides comparative simulation studies of RT-OPT with other intuitive but suboptimal algorithms such as GREED-EDF. The simulation uses public-domain MPEG traces from several popular movies, and uses a Zipfian distribution to model viewer preference for different movie clips. Chapter 6 presents the conclusions.

## Chapter 2

# Fundamentals of Continuous Media Display and Magnetic Disk Drives

## 2.1 Continuous Media Display Overview

To support continuous display of a video object  $X$ , several studies have proposed partitioning  $X$  into  $n$  equal-sized blocks:  $X_0, X_1, \dots, X_{n-1}$  [31, 7]. The display time of a block and its transfer time from the disk are a fixed function of the display requirements of an object and the transfer rate of the disk, respectively. Using this information, the system stages block  $X_1$  from the disk into main memory prior to completion of the display of  $X_0$ . This ensures a smooth transition between the two blocks in order to support a continuous display. This process is repeated until all blocks of  $X$  have been retrieved and displayed. The system needs to estimate the disk service time in order to stage a block into memory in a timely manner to avoid starvation of data i.e, hiccups. Section 2.4 describes techniques to model the characteristics of disk drives to obtain the necessary service time estimates. Most multi-disk designs utilize striping to assign data blocks of each CM file to individual disks. With striping, a file is broken into (fixed) striping units which are assigned to the disks in a round-robin manner. There are two basic ways to retrieve striped data (a) in parallel to utilize the aggregate bandwidth of all the disks (this is typically done in RAID systems), or (b) in a cyclic fashion to reduce the buffer requirements (this method is sometimes referred to as simple striping or RAID level 0). Both scheduling techniques can also be combined in a hierarchical fashion by forming several clusters of disks. Data retrieval proceeds in parallel within a cluster and in cycles across the clusters. When identical disks are used, all the above techniques feature perfect load-balancing during data retrieval and

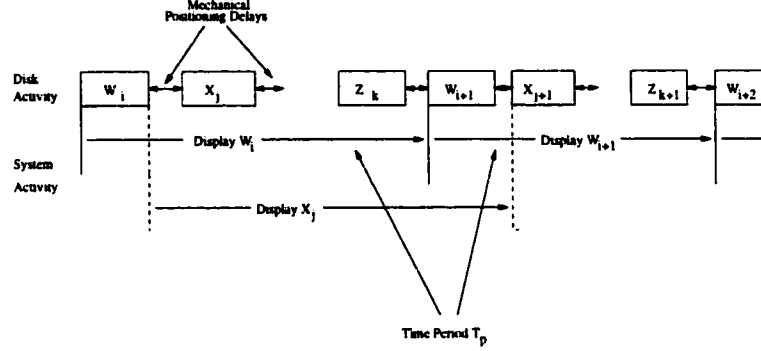


Figure 2.1 : Continuous Display of Multiple Objects

on average equal amount of data is stored on every disk. Level 1 and above of RAID systems have only been analyzed for homogeneous disk arrays, since their performance depends critically on the slowest disk drive (parity information must be calculated from the data of all disks, and read or written for each I/O operation to complete [15]). Note that display time of a block is in general significantly longer than its transfer time from the disk drive assuming a compressed video object. Thus the bandwidth of a disk drive can be multiplexed among several displays referencing different objects (Figure 2.1).

A magnetic disk drive is a mechanical device. Multiplexing it among several displays causes it to incur mechanical positioning delays. The source of these delays is described in Section 2.3, which provides an overview of the internal operation of disk drives. Such overhead is wasteful and it reduces the number of simultaneous displays supported by the system. Section 2.2 details how advanced scheduling policies can minimize the impact of these wasteful operations.

### 2.1.1 Target Environment

An illustration of our target hardware platform is provided in Figure 2.2. A high performance system bus with nanosecond latency and transfer rates in excess of 100 Mbytes per second, neglecting the arbitration overhead, is considered. It connects all the major components of the computer system; the

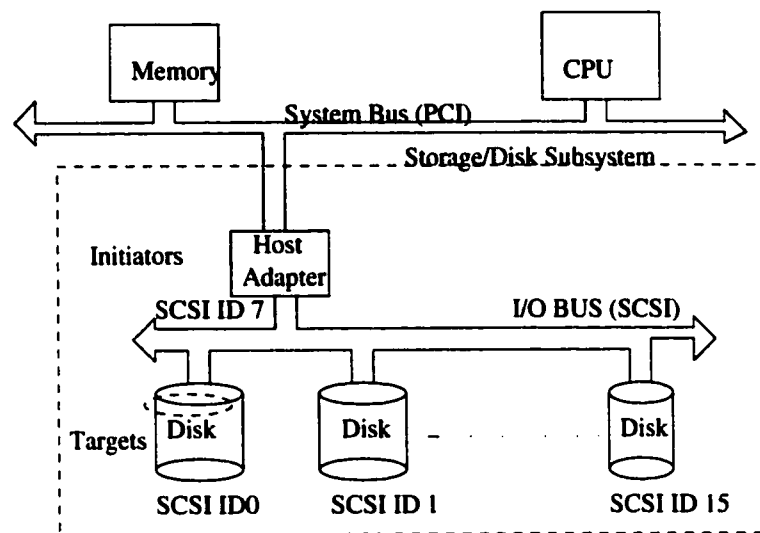


Figure 2.2 : Storage Subsystem Architecture

memory, the CPU (central processing unit), and any attached devices, such as display, network and storage subsystems. Within the storage subsystem each individual device e.g., a disk or a tape, is attached to the I/O bus which in turn is connected to the system bus through a *host adapter*. The host adapter translates the I/O bus protocol into the system bus protocol and it may improve the performance of the overall system by providing caching, and off-loading functions from the main processor. The disk subsystem is central to this study and is detailed in the next section.

### 2.1.2 Modern Disk Drives

Magnetic disk-drive technology has benefited from more than two decades of research and development. It has evolved to provide storage with relatively low latency (in the order of milliseconds) and a low cost per MByte of storage (few cents per MByte). Magnetic disk drives are commonly used for a wide variety of storage purposes in almost every computer system. To facilitate their integration and compatibility with a wide range of host hardware and operating systems, the interface that they present to the rest of the system is



well defined and hides a lot of complexities of the actual internal operation. For example, the popular SCSI (Small Computer System Interface) standard presents a magnetic disk drive to the host system as a linear vector of storage blocks (usually of size 512 bytes each). When an application requests the retrieval of one or several blocks, the data will be returned after some time but there is no explicit mechanism to inform the application exactly how long such an operation will take. In many circumstances such a best effort approach is reasonable because it simplifies program development by allowing the programmer to focus on the task at hand instead of the physical attributes of the disk drive. However, for a number of data intensive applications, for example continuous media servers, exact timing information is crucial to satisfy the real-time constraints imposed by the requirement for a jitter-free delivery of audio and video streams. Fortunately, with a model that imitates the internal operation of a magnetic disk drive, it is possible to predict service times at the level of accuracy that is needed to design and configure CM server storage systems.

### **2.1.3 Internal Operation**

A magnetic disk drive is a mechanical device, operated by its controlling electronics. The mechanical parts of the device consist of a stack of platters that rotate in unison on a central spindle. Presently, a single disk contains one, two or as many as fourteen platters (see Figure 2.3). Each platter surface has an associated disk head responsible for reading and writing data. The minimum storage unit on a surface is a sector (which commonly holds 512 bytes of user data). The sectors are arranged in multiple, concentric circles termed tracks. A single stack of tracks across all the surfaces at a common distance from the spindle is termed a cylinder. To access the data stored in a series of sectors, the disk head must first be positioned over the correct track. The operation to reposition the head from the current track to the target track is termed a seek. Next, the disk must wait for the desired data to rotate under the head. This time is termed the rotational latency. In their quest to improve performance, disk manufac-

turers have introduced methods such as zone-bit recording, partial-response maximum likelihood, and high spindle speeds. Some manufacturers also redesigned disk internal algorithms to provide uninterrupted data transfer (e.g. to avoid lengthy thermal recalibrations) specifically for CM applications. Many of these technological improvements have been introduced gradually with new disk generations entering the market place every year. To date, disk drives of many different performance levels with capacities ranging from 1 up to 18 gigabytes are commonly available. The next section details disk modeling techniques to identify the physical characteristics of a magnetic disk in order to estimate its service time.

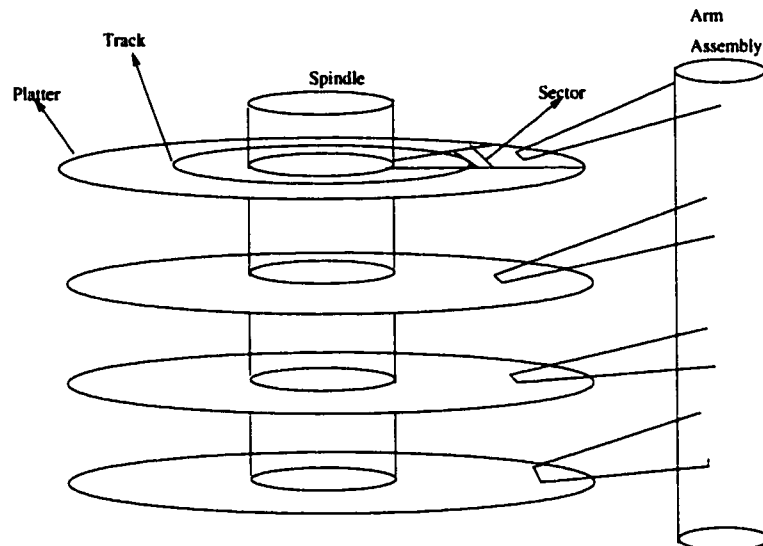


Figure 2.3 : Disk Drive Internals

#### 2.1.4 Disk Drive Modeling

Disk drive simulation models can be extremely helpful to investigate performance enhancements or trade-offs in storage subsystems. Hence a number of techniques to estimate disk service times have been proposed in the literature [35, 4, 10]. These studies differ in the level of detail that they incorporate

into their models. The level of detail should depend largely on the desired accuracy of the results. More detailed models are generally more accurate, but they require more implementation effort and more computational power. Simple models may assume a fixed time for I/O or they may select times from a uniform distribution. However, to yield realistic results in simulations that imitate real-time behavior more precise models are necessary. The most detailed models include all the mechanical positioning delays (seeks, rotational latency), as well as on-board disk block caching, I/O bus arbitration, controller overhead and defect management.

## Chapter 3

### Real-Time Model

The system model of a video server that we consider in a server-network-client architecture is shown in Figure 3.1. The server consists of an array of  $D$  parallel disks and a main memory of size  $M$  blocks to buffer the prefetched blocks. A block is the unit of storage and access from the disks. Video streams are VBR encoded and stored in CDL format so that blocks are of fixed size, and therefore the number of frames per block is variable. Each video clip is assumed to be striped across all the disks in a round-robin fashion, a common data placement method used in video servers. (For other data-placement schemes based on random assignment or partitioning of the disks into groups see [36, 30]). Our framework places no restriction on the placement of the blocks; for specificity and for empirical simulation we assumed round-robin striping at the granularity of 64KB blocks. As the number of frames per block is different due to VBR-CDL encoding, for a given playback rate (assumed to be 24 frames per second) the playback times of blocks at a client are different. The clients are assumed to have buffers to hold the current block being decoded and played back, and to mask network delays.

To take into account the real-time dynamics in which the video server operates, several delay components must be considered. By incorporating the time spent on the different subsystems, the model forms a basis for real-time scheduling of parallel I/O. Each block has associated with it a playback deadline which is the time at which the first frame in the block must be played back. Based on the time required by the client for decompression and display preprocessing, and for transmission delay in the network, this places a deadline by which the block must be transmitted by the server into the network; in turn

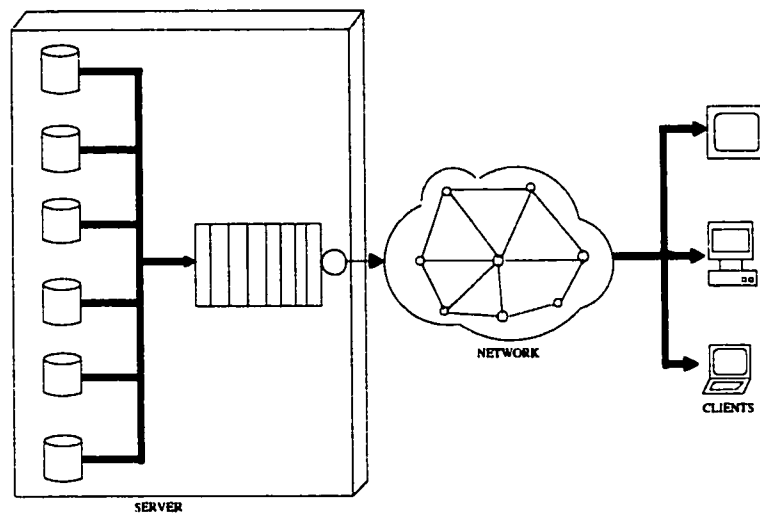


Figure 3.1 : Server Model

this places a deadline on the latest time an I/O for the block can be initiated by the disk system. Due to the VBR encoding into fixed-size blocks the spacing between playback deadlines of individual blocks is variable. The I/O times of individual blocks are influenced by variable seek and rotational latencies and the disk-head scheduling employed at individual disks, and are also variable. The network delay for the transfer of each block and the processing delay for decompression at the client may also vary significantly.

The temporal evolution of each block in our real-time framework is modeled by a *time line*. The lifetime of a block is represented by its time line and indicates the state of the block between the start of its I/O and the end of its playback as shown in Figure 3.2 (a). The time line consists of six adjacent time intervals, respectively: disk I/O, server-buffer residence, network transmission, client-buffer residence, processing-decompression, and playback. The playback deadline for each block, PB Start, is fixed and set by the frame rate and encoding of the previous blocks. The decompression and playback interval is also fixed and depends on the frame rate and encoding of that block. The network transmission interval in general depends on network load and characteristics; for our purpose it is assumed to be known for every client. That

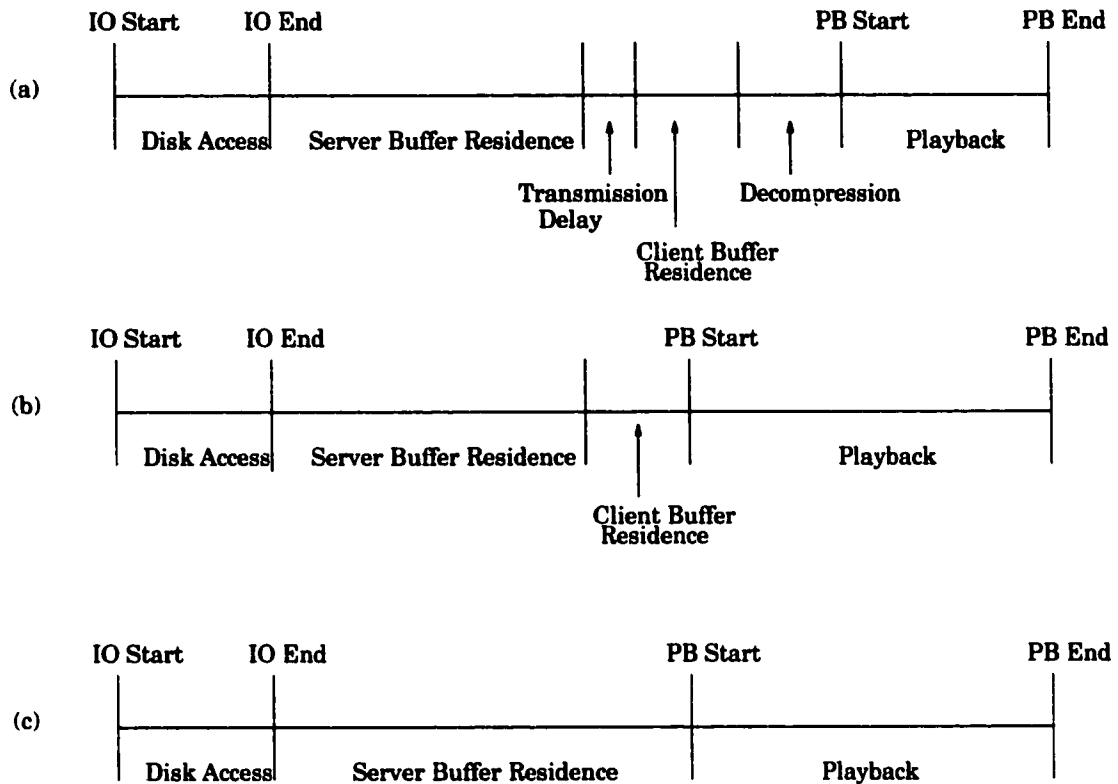


Figure 3.2 : Time line of a Block

is the network provides a guaranteed upper-bound on the latency of any block delivered to a specific client. The I/O interval is determined by the disk-head scheduling policy for that disk and the disk characteristics.

From the viewpoint of the server the model can be simplified by incorporating the network delay and client-side decompression and processing into the playback interval, by shifting the PB Start time earlier by the sum of the times of the two intervals. Figure 3.2 (b) shows the four states of a block with the modified PB Start time. PB Start sets a deadline by which time the I/O for that block should complete; that is IO End cannot be delayed past PB Start. In turn, based on the disk I/O time for the block this sets a latest deadline for IO Start, by which time the I/O of the block must begin. The I/O for a block may begin (and end) earlier than its deadline at which time it is said to be

prefetched. A prefetched block stays in the server buffer until it is transmitted to the network, a variable-length interval depending on how early it was prefetched. Similarly there is an interval, of possibly zero length, during which the block resides in the client buffer while previous blocks are being played out. To stress the server we assume minimal client features; a client only buffers the frames of the current block, so all prefetching is buffered at the server memory. Hence, the time line is further qualified to make the client-buffer residence time zero. Figure 3.2 (c) shows the time line with only three distinct states: disk I/O, Server Buffer Residence, and Playback.

The input to the scheduling algorithm RT-OPT is a string of blocks  $B_i$ , each with its associated disk access time  $L_i$  (IO End - IO Start), and deadline  $D_i$ , the latest time at which the block can complete its I/O. Note that  $D_i$  is PB Start for that block. A block for which IO End is less than  $D_i$  stays in the server buffer till  $D_i$  at which time it is transferred to the client. Each disk is assumed to be serviced using some fixed disk-head scheduling policy (say SCAN, EDF or SCAN-EDF [23] for instance) so that disk access times  $L_i$  for each block can be estimated a-priori. The scheduling algorithm RT-OPT then determines the time at which each block should begin its I/O (IO Start). If RT-OPT fails to find a schedule in which all blocks meet their deadlines  $D_i$ , then it will be shown (in chapter 4) that no other algorithm using the same individual disk-head scheduling can meet the deadlines. As an admission control method the algorithm RT-OPT will determine whether a schedule exists in which each block meets its deadline  $D_i$ . If so, the new client is admitted, else it is refused.

To illustrate some of the issues involved we first consider a simple, intuitive algorithm GREED-EDF to schedule the I/Os. This algorithm tries to maximize disk concurrency by fetching a block from each disk (whenever possible) on an I/O as done by algorithms based on disk striping. However, while disk striping fetches consecutive blocks from the same data stream from all disks in a parallel I/O, GREED-EDF gives priority to the block on a disk with the earliest deadline. Thus a striped I/O in GREED-EDF consists of the earliest-required block from each disk, and the entire stripe is read in parallel. If the available

### Algorithm GREED-EDF

At any time  $t$

For each disk  $i$  in  $I(t)$  let  $B_i$  denote the block from disk  $i$  with the earliest deadline  $D_i$ .

Initiate I/Os to fetch  $F(t)$  blocks  
from  $\{B_1, B_2, \dots\}$  with the earliest deadlines.

Figure 3.3 : Algorithm GREED-EDF

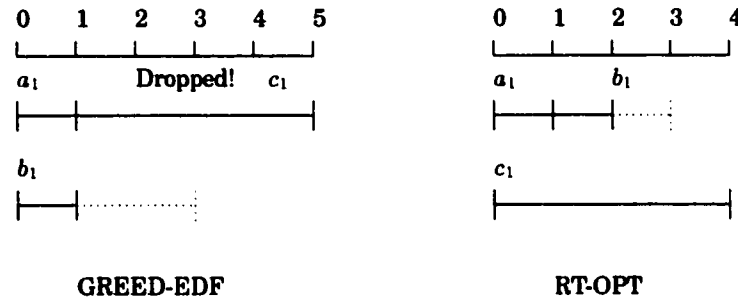


Figure 3.4 : Illustration of GREED-EDF and RT-OPT

buffer at any time is less than the number of disks so that an entire stripe cannot be fetched, it fetches from the disks containing blocks with the earliest deadlines. The blocks that cannot meet their deadline are dropped.

Figure 3.3 shows the pseudo-code of GREED-EDF. In the description, at any time  $t$ ,  $I(t)$  represents the set of idle disks, and  $F(t) = \min\{N(t), |I(t)|\}$ , where  $N(t)$  is the number of free blocks in the server buffer.

Figure 3.4 illustrates the working of GREED-EDF. Let the I/O system consist of 3 disks and buffer of capacity 2 blocks. Let 3 blocks be requested by a client, one from each disk,  $a_1$ ,  $b_1$ , and  $c_1$  with deadlines 1, 3 and 4 respectively. Let the I/O time for blocks  $a_1$  and  $b_1$  be 1 each while that of  $c_1$  be 4. Since the



deadline of  $b_1$  (3) is earlier than that of  $c_1$  (4), GREED-EDF fetches blocks  $a_1$  and  $b_1$  initially. Thus an I/O for  $c_1$  cannot start before time 1, which causes it to miss its deadline. On the other hand we could initiate an I/O for  $c_1$  at time 0 and start fetching  $b_1$  only at time 1 which can allow all deadlines to be met. This schedule is constructed by RT-OPT.

## Chapter 4

### Algorithm RT-OPT

Algorithm RT-OPT is used to schedule a set of blocks  $\{B_i, i = 1, \dots, N\}$ . Each block  $B_i$  has a disk access time  $L_i$  and deadline for I/O completion  $D_i$  associated with it. The disk access time are obtained by assuming a particular order of accessing the blocks on a single disk. RT-OPT will schedule each disk in accordance with that policy. As shown by the example of GREED-EDF in chapter 4 it is non-trivial to decide when to initiate the I/Os on the different disks. RT-OPT provides an optimal scheduling method: if it fails to find a schedule whereby all blocks meet their deadlines then no such schedule exists with the given buffer size and access ordering of individual disks.

RT-OPT operates as follows. Initially the server buffer residence times are set to zero for all blocks. The blocks are first tentatively scheduled so that each block's IO End time coincides with its deadline. Figure 4.1 illustrates an example of scheduling four blocks A, B, C and D using 2 disks and 2 clients. Blocks A and B are consecutive blocks of a stream destined for client 1, and C and D are consecutive blocks destined for client 2. Blocks A and C are on disk 1 and blocks B and D on disk 2. The deadlines for blocks A, B, C and D are 3, 5, 4 and 7 respectively. Note that consecutive blocks of a client have a small overlap in playback durations due to network delay and client-side processing. The disk access times for the blocks are as shown in the figure; blocks A, C and D requires 2 time units, block B requires 4 units. The shared buffer size is assumed to be of size 2 blocks. When an I/O is initiated for a block, space is also reserved for it in the shared buffer.

Next RT-OPT iteratively runs through the set of blocks ordered in time in order to resolve conflicts. The first type of conflict occurs due to overlapping

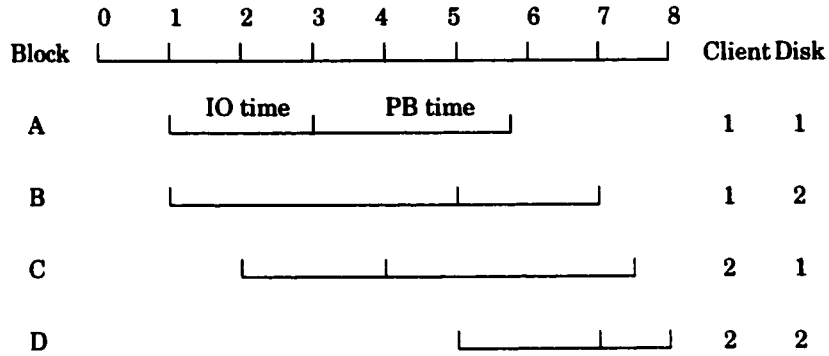


Figure 4.1 : Initial step of Algorithm RT-OPT

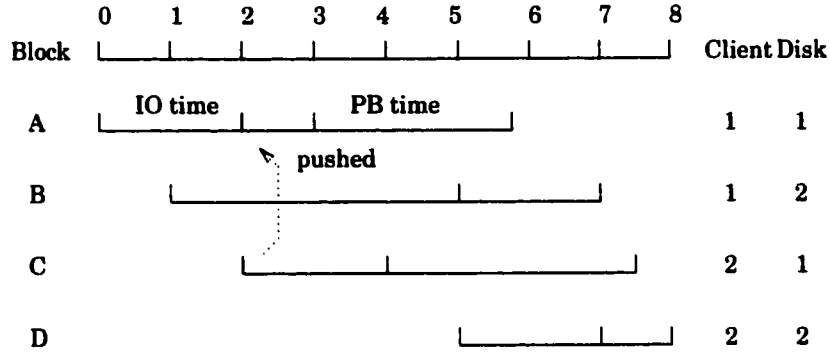


Figure 4.2 : RT-OPT following the removal of disk-access conflicts

disk access intervals on a single disk. Since only one I/O can be in progress from a single disk at any time, the disk access interval of one of these blocks must be moved earlier in time to remove the overlap. The chosen order of access from the disk determines which will stay and which will be prefetched by pushing it back. The pushing back of the disk access interval for a block, creates a server-buffer residency for the block. The block is pushed back the minimal amount, so that its IO End coincides with the IO Start of the next block to be fetched from that disk. In Figure 4.1, the disk accesses for blocks A and C from disk 1 overlap between times 2 and 3. The IO Start time for block A is pushed back so that its IO End time coincides with the IO Start time of block C. Since the blocks on disk 2, B and D, already have disjoint disk access intervals, these are not changed. Figure 4.2 shows the modified schedule after

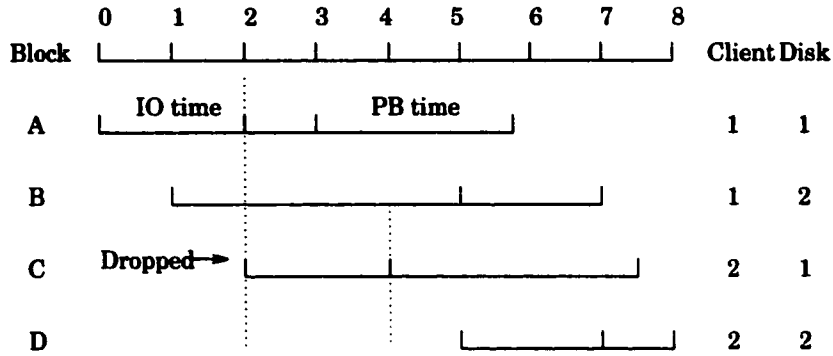


Figure 4.3 : RT-OPT following the removal of buffer conflicts

the IO Start time of block A is pushed forward to 0. Between times 2 and 3, the block remains in the server buffer.

Finally RT-OPT checks for buffer violations in the schedule. At any time the total number of blocks being fetched or which are in the server-buffer-residence state cannot exceed  $M$ , the size of the server buffer. The blocks are scanned in increasing order of time, and blocks whose I/O initiation would violate the buffer constraint are dropped. For instance in Figure 4.3, at time 2, I/Os for blocks A and B are already in progress, and assuming  $M = 2$ , the I/O for block C cannot be accommodated. The I/O for C cannot be deferred without violating its deadline, and hence the block is dropped. The next I/O initiation for block D at time 5 is permitted since there is buffer available.

The pseudo-code for algorithm RT-OPT is shown in Figure 4.4. In the description,  $\text{StartPB}(b)$  denotes the playback start time PB Start for block  $b$ ; similarly  $\text{StartIO}(b)$  and  $\text{EndIO}(b)$  denote IO Start and IO End times for block  $b$ . As used previously, at any time  $t$  let  $I(t)$  represent the set of idle disks, and  $F(t) = \min\{N(t), |I(t)|\}$ , where  $N(t)$  is the number of free blocks in the server buffer.

RT-OPT's policy of fetching blocks as late as possible to meet the deadlines minimizes the buffer occupancy of each block and consequently the probability of buffer overflow for a given buffer size. We prove below that RT-OPT is optimal in the sense that if it drops some blocks then every other algorithm must

### Algorithm RT-OPT

```

for  $d=1$  to Number of disks /* Initialization */
  for  $b = 1$  to last block on disk  $d$ 
     $\text{StartIO}(b) \leftarrow \text{StartPB}(b) - \text{IOTime}(b)$ ;  $\text{EndIO}(b) \leftarrow \text{StartPB}(b)$ 
    call PushDisk( $d, b$ )

At any time  $t$ 
  For each disk  $i$  in  $I(t)$  let  $B_i$  denote the block with
    the earliest I/O start time such that  $t + \text{IOTime}(B_i) \leq \text{StartPB}(B_i)$ 
  Initiate I/Os to fetch  $F(t)$  blocks from
     $\{B_1, B_2, \dots\}$  with earliest IO Start times.

Procedure PushDisk(disk  $d$ , block  $b$ )
if  $b \neq 1$  and  $\text{EndIO}(b-1) > \text{StartIO}(b)$  then
   $\text{EndIO}(b-1) \leftarrow \text{StartIO}(b)$ 
   $\text{StartIO}(b-1) \leftarrow \text{EndIO}(b-1) - \text{IOTime}(b-1)$ 
  call PushDisk( $d, b-1$ )

```

Figure 4.4 : Algorithm RT-OPT

also drop at least one block for the same data placement, buffer size and single disk-head scheduling policy. By minimizing the buffer occupancy of each block and consequently the probability of buffer overflow, RT-OPT provides maximal opportunity for other blocks to schedule their I/Os, increase I/O parallelism and meet their deadlines.

**Definition 1** The Block-Sequence of disk  $d$  is the ordered sequence of blocks fetched from disk  $d$ . It is determined by the disk-head scheduling algorithm for the disk.

**Theorem 1** *Given the Block-Sequence for each disk, if in the schedule created by RT-OPT any block misses its deadline then there is no schedule in which all blocks meet their deadline.*

**Proof :** If a block is dropped by RT-OPT then there must be some time  $t$  and a block  $b$  such that  $\text{StartIO}(b) = t$ , after initialization, and the number of blocks in the buffer at time  $t$  is  $M$ . Consider the earliest such time  $t'$  and the corresponding block  $b'$ .

For the sake of contradiction, assume that there is a schedule  $S$  such that no blocks are dropped. Let the set of blocks in the I/O buffer at time  $t'$  be  $B(t')$ . By Lemma 1 for each block  $b$  in  $B(t')$  schedule  $S$  fetches it earlier than  $\text{StartIO}(b) < t'$ . Hence, considering schedule  $S$ , the number of blocks in the buffer is at least  $|B(t')| = M$ . Hence block  $b'$  cannot be fetched at time  $t' = \text{StartIO}(b')$  in schedule  $S$ . This contradicts Lemma 1 as  $S$  claims to schedule all requests without dropping any.  $\square$

**Lemma 1** *Given a disk sequence for a disk  $d$ , if after the initialization step in RT-OPT  $\text{StartIO}(b) = t$ , then no other schedule with the same Block-Sequence can fetch  $b$  later than time  $t$  and still service all requests.*

**Proof :** Proof by induction on the requests in the disk sequence. Let there be  $n$  requests in the disk sequence. Consider the last request,  $b_n$ , in the disk sequence.  $t = \text{StartIO}(b_n) = \text{StartPB}(b_n) - \text{IOTime}(b_n)$ . Hence if in any schedule  $b_n$  is fetched after time  $t$  it will clearly miss its deadline. Let us assume that the lemma holds for all requests  $b_i$ ,  $k \leq i \leq n$ .

Consider block  $b_{k-1}$ . By construction,  $\text{EndIO}(b_{k-1}) = \min\{\text{StartIO}(b_k), \text{StartPB}(b_{k-1})\}$ . Two cases are now possible

**Case 1:**  $\text{EndIO}(b_{k-1}) = \text{StartPB}(b_{k-1})$ . Similar to the base case, if in an alternative schedule  $b_{k-1}$  is fetched after time  $\text{StartIO}(b_{k-1})$ , then it will miss its deadline.

**Case 2:**  $\text{EndIO}(b_{k-1}) = \text{StartIO}(b_k)$ . In this case if in some other schedule  $b_{k-1}$  is fetched after time  $\text{StartIO}(b_{k-1})$  then the I/O for block  $b_k$ , which is from the

same disk, cannot start till after  $\text{EndIO}(b_{k-1})$ . By the induction hypothesis, this is not possible without violating the deadline of some other block.  $\square$

## Chapter 5

### Simulations

For the performance evaluation of RT-OPT, a trace-driven simulator in C++ based on our real-time model was developed. We used public-domain video traces described in [25]. These are obtained from real MPEG compressed video clips and form a representative and a realistic set of MPEG compressed VBR (variable bit rate) video data traces. These were preprocessed and stored on the simulated disks using CDL format.

The frame rate for the traces is 24 frames per second and each trace contains 40000 frames corresponding to 28 minutes of video. Using the data on the coded frame sizes for each of the video sequences, we created constant data length blocks of 64 Kbytes from 5 representative streams: Bond, Terminator, Lambs, Dino and Starwars. The blocks were then striped across the disks in a round-robin fashion.

To model the probabilistic access structure of clients we used the Zipf distribution [37], which assigns a discrete probability  $P_i = \frac{C}{i^{(1-\Theta)}}$  to the  $i$ th video clip for any client's access request; here  $\Theta = 0.271$  is the skewness factor that reflects the popularity of a clip. Clients were assumed to access the chosen clip from a randomly chosen starting frame; at the end of the stream, every client views the same clip from the beginning till the end of the simulation which is chosen to be the access of 50000 blocks. We assumed EDF scheduling at individual disks, and the I/O times were uniformly distributed between 16 and 44ms.

For different number of users, we varied the shared buffer size and the number of disks and obtained the maximum number of users that could be supported without dropping any blocks. Both GREED-EDF and RT-OPT schedul-



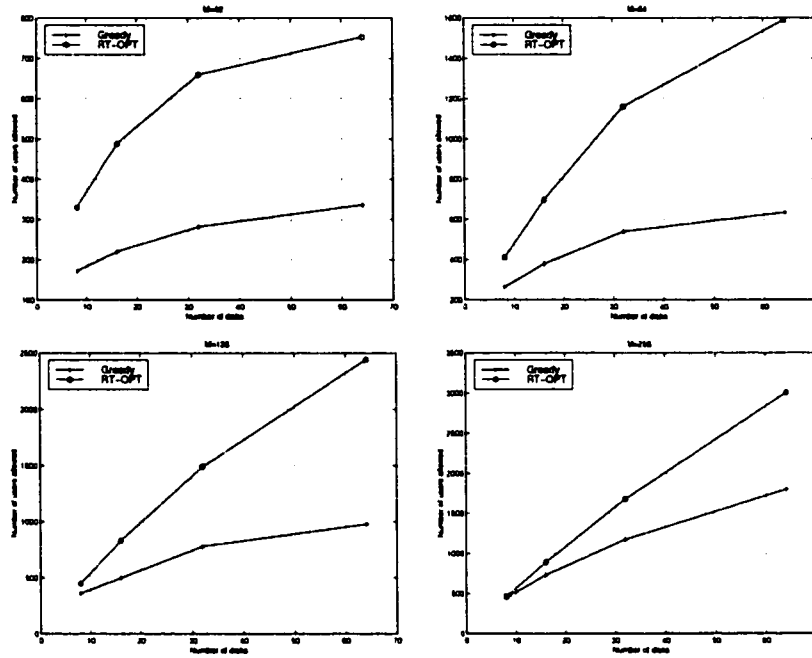
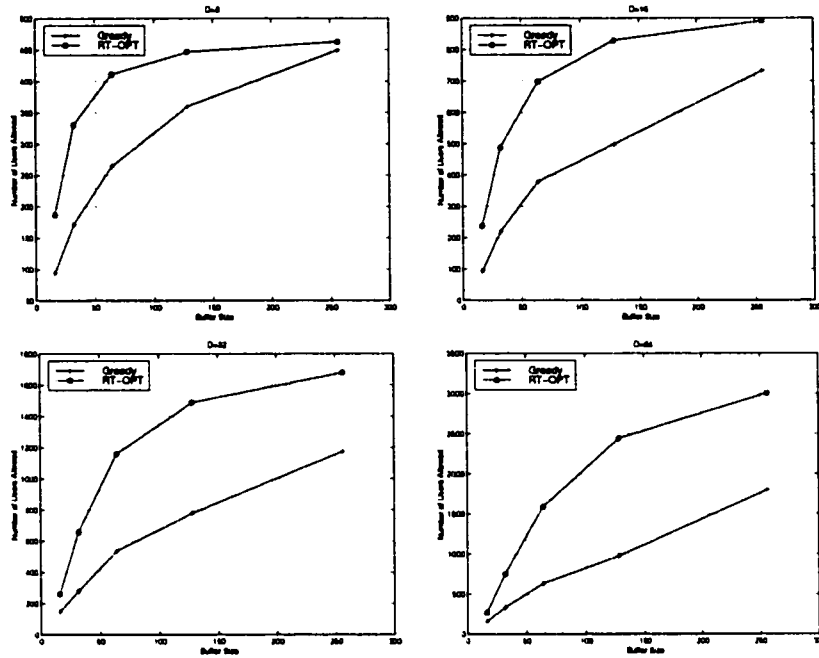


Figure 5.1 : Number of Users Serviced vs. Number of Disks

ing algorithms were simulated. The number of users supported was averaged over several Monte Carlo trials, to smooth out the effects of randomization in the choice of the frame in the formation of the reference string.

The scalability of the algorithms RT-OPT and GREED-EDF with number of disks is illustrated by the graphs in Figure 5.1. The scalability of RT-OPT in the number of disks is superior to that of GREED-EDF and very close to linear at all buffer sizes  $M = 32, 64, 128$  and  $256$ . At  $M = 32$ , the scalability of RT-OPT is saturated at higher number of disks and diverges from linearity due to memory limitations; however it is almost linear in the number of disks with higher buffer sizes of  $M = 64, 128$  and  $256$ , while the scalability of GREED-EDF saturates very quickly. Furthermore, the difference in the number of clients supported by RT-OPT and GREED-EDF is maximal when the buffer size is small at  $M = 32$  due to the optimal fetching policy of RT-OPT; however the difference in the number of clients serviced between RT-OPT and GREED-EDF decreases as the buffer size gets larger such as  $M = 64, 128$  or  $256$ . When the



**Figure 5.2 : Number of Clients Served at Zero Dropping Probability vs. Buffer Size**

buffer size is large and the number of disks is small in which case the fetching policy is not a significant factor in determining the number of clients serviced, the performance of RT-OPT and GREED-EDF are very close. Nonetheless, the trend is evident that for larger parallel disk configurations, RT-OPT supports a substantially larger number of clients with very modest buffer requirements.

Another important metric for the video server is the concurrent number of clients it can service at a specific average block dropping probability. Figure 5.2 shows the the number of clients serviced at zero dropping probability versus buffer size for the number of disks 8, 16, 32 and 64. RT-OPT does substantially better with respect to GREED-EDF in terms of the number of clients serviced at zero average block dropping probability for all number of disks and buffer sizes. When the buffer size gets large for a given number of disks, the fetching policy is not as significant performance of both RT-OPT and GREED-EDF converge.

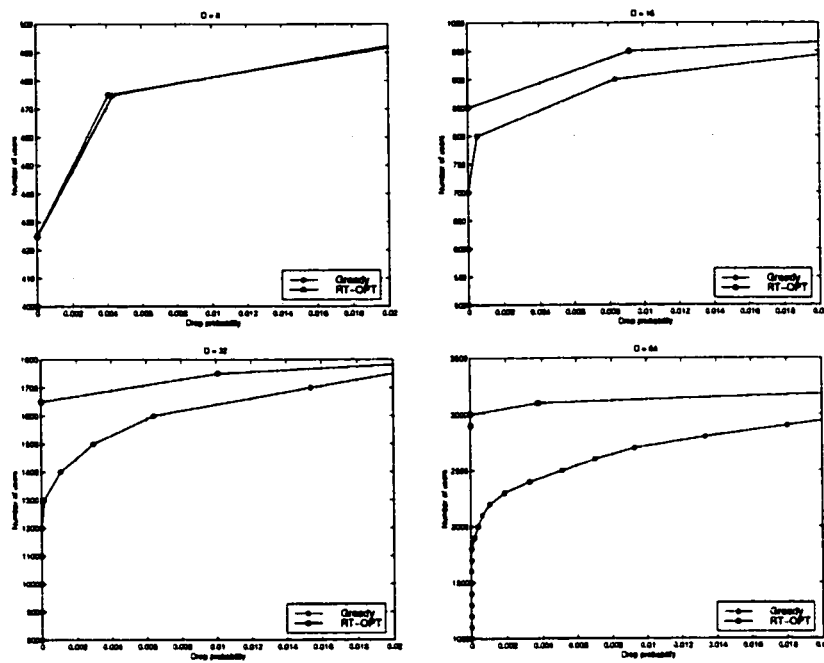


Figure 5.3 : Number of Users Serviced vs. Drop Probability

Finally in Figure 5.3 the two algorithms are compared by determining the number of clients that can be supported with a given block dropping probability. The maximum drop rate permitted was fixed at 2%, and  $M$  was fixed at 256. In all cases the superiority of RT-OPT over GREED-EDF is evident. As expected the improvement is greatest with larger number of disks when the need for efficient buffer management becomes more acute.

## **Chapter 6**

### **Conclusions**

We introduced a framework for incorporating real-time constraints on block accesses in a parallel I/O system. The framework was used to design prefetching and scheduling algorithms for real-time playback of multiple video streams in a multiple-disk, parallel I/O based video server.

Our real-time model is based on a distributed server-network-client architecture that allows one to incorporate the real-time constraints imposed by the delivery requirements of continuous video data to the clients over the networks, and the resource constraints such as CPU/memory/network bandwidth. These can be translated into constraints on the prefetching and I/O scheduling and dealt with in an integrated manner.

We introduced a scheduling algorithm RT-OPT for optimally fetching blocks into the server buffer. We showed that if the schedule created by RT-OPT fails to meet the deadline of block, then no feasible schedule is possible for the same buffer size, data placement and single-disk scheduling policy. Thus RT-OPT is optimal in the sense that if it drops a block then every other algorithm drops at least one block.

Using traces from MPEG compressed video clips we simulated RT-OPT and empirically observed its performance. By dynamically multiplexing the buffer among different clients and disks optimally, RT-OPT was shown to be highly scalable with the number of disks with only modest buffer requirements. The number of clients supported was shown to be uniformly superior to intuitive but suboptimal algorithms like GREED-EDF that attempt to keep the disks busy and fetch in order of deadlines.

## Bibliography

- [1] R. D. Barve, M. Kallahalla, P. J. Varman, and J. S. Vitter. Competitive Parallel Disk Prefetching and Buffer Management. In *Proceedings of the Fifth Workshop on I/O in Parallel and Distributed Systems*, pages 47–56. ACM Press, November 1997.
- [2] E. W. Biersack and F. Thiesse. A New Class of Constant Data Length Retrieval Algorithms for Video Servers with VBR Streams. In C.-C. Jay Kuo, editor, *Multimedia Storage and Archiving Systems*, volume 2916, pages 316–329. SPIE, November 1996.
- [3] E. W. Biersack, F. Thiesse, and C. Bernhardt. Constant Data Length Retrieval for Video Servers with Variable Bit Rate Streams. In *Proceedings of the IEEE Conference on Multimedia*, pages 151–155, June 1996.
- [4] Dina Botton and Jim Gray. Disk Shadowing. In *Proceedings of the International Conference on Very Large Databases*, pages 331–338, September 1988.
- [5] Ed Chang and A. Zakhor. Admission Control and Data Placement for VBR Video Servers. In *Proceedings of 1st International Conference on Image Processing*, pages 316–329, November 1994.
- [6] Ed Chang and A. Zakhor. Cost Analyses for VBR Video Servers. *IEEE Multimedia*, 3(4):56–71, 1994.
- [7] H. J. Chen and T. Little. Physical Storage Organizations for Time-Dependent Multimedia Data. In *Proceedings of the Foundations of Data Organization and Algorithms Conferenc*, October 1993.

- [8] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High Performance Reliable Secondary Storage. *ACM Computing Surveys*, 26(2):145–185, 1994.
- [9] J. Dey, C. Shih, and M. Kumar. Storage subsystem in a large multimedia server for high-speed networks environments. In *IST/SPIE Symposium on Electronic Imaging Science and Technology*, February 1994.
- [10] Bruce Worthington et al. On-Line Extraction of SCSI Disk Drive Parameters. In *Proceedings of the ACM Sigmetrics*, 1995.
- [11] S. Shim et al. An effective data placement scheme to serve popular video-on-demand. Technical report, University of Minnesota, 1995.
- [12] International Organization for Standardization. MPEG-1: Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5Mbit/s, 1996. ISO/IEC JTC1/SC29/WG11 N MPEG96/June1996.
- [13] International Organization for Standardization. MPEG-2: Coding of Moving Pictures and Associated Audio, 1996. ISO/IEC JTC1/SC29/WG11 N MPEG96/June1996.
- [14] C. S. Freedman and D. J. DeWitt. The SPIFFI Scalable Video-on-Demand System. In *Proceedings of SIGMOD'95*, pages 352–363. ACM, 1995.
- [15] Mark B. Friedman. RAID keeps going and going and... *IEEE Spectrum*, 33(4):73–79, April 1996.
- [16] ISO/IEC. ISO/IEC 13818-3 Generic Coding of Moving Pictures and Associated Audio Information: Audio, 1995.
- [17] ISO/IEC. ISO/IEC 13818-1 Generic Coding of Moving Pictures and Associated Audio Information: Systems, 1996.
- [18] M. Kallahalla and P. J. Varman. Improving Competitiveness of Parallel-Disk Buffer Management using Randomized Writeback. In *Proceedings of*

- International Conference on Parallel Processing*, pages 270–277, August 1998.
- [19] M. Kallahalla and P. J. Varman. Optimal Read-Once Parallel Disk Scheduling. In *Proceedings of Seventh Workshop on I/O in Parallel and Distributed Systems*, 1999.
  - [20] P. V. Rangan and H. Vin. Efficient Storage Techniques for Digital Continuous Multimedia. *IEEE Transactions on Knowledge and Data Engineering*, 5(6), August 1993.
  - [21] B. Ozden R. Rastogi and A. Silberschatz. Demand Paging for Video-on-Demand Servers. In *IEEE International Conference on Multimedia Computing and Systems*, pages 264–272, May 1995.
  - [22] A. L. N. Reddy and J. C. Wyllie. Disk Scheduling in Multimedia I/O Systems. In *Proceeding of ACM Conference on Multimedia*, pages 225–233, 1993.
  - [23] A. L. N. Reddy and J. C. Wyllie. I/O issues in a Multimedia System. *Computer*, 27(3):69–74, March 1994.
  - [24] M. Reisslein and K. W. Ross. High-Performance Prefetching Protocols for VBR Prerecorded Video. *IEEE Network*, 12(6):46–55, November 1998.
  - [25] O. Rose. Statistical Properties of MPEG Video Traffic and their Impact on Traffic Modelling in ATM Systems. Research Report Series 101, University of Wurzburg, Institute of Computer Science, February 1995. <ftp://ftp-info3.informatik.uni-wuerzburg.de>.
  - [26] K. Salem and H. Garcia-Molina. Disk striping. In *Proceedings of International Conference on Database Engineering*, February 1986.
  - [27] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 1996.
  - [28] T. Sikora. MPEG Digital Video Coding Standards. *IEEE Signal Processing Magazine*, pages 82–100, September 1997.

- [29] M. Tekalp. *Digital Video Processing*. Prentice Hall, 1995.
- [30] R. Tewari, D. M. Dias, R. Mukherjee, and H. M. Vin. High Availability in Clustered Multimedia Servers. In *Proceedings of the 12th International Conference on Data Engineering*, pages 645–654, February 1996.
- [31] F. A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *First ACM Conference on Multimedia*, 1993.
- [32] P. J. Varman and R. M. Verma. Tight Bounds for Prefetching and Buffer Management Algorithms for Parallel I/O Systems. In *Foundations of Software Technology and Theoretical Computer Science*, volume 16 of *LNCS*. Springer Verlag, December 1996.
- [33] Harrick M. Vin, Prashant Shenoy, and Sriram S. Rao. Analyzing The Performance of Asynchronous Disk Arrays for Multimedia Retrieval. In *Proceedings of the ISMM International Conference for Distributed Multimedia Systems and Applications*, pages 14–17, 1994.
- [34] J. S. Vitter and E. A. M. Shriver. Optimal Algorithms for Parallel Memory, I: Two-Level Memories. *Algorithmica*, 12(2–3):110–147, 1994.
- [35] N. C. Wilhelm. A General Model for The Performance of Disk System. *Journal of the ACM*, 24(1):14–31, 1977.
- [36] P. S. Yu, M-S. Chen, and D. D. Kandlur. Grouped Sweeping Scheduling for DASD-based Multimedia Storage Management. *Multimedia Systems*, 1(1):99–109, January 1993.
- [37] G. K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, Reading, MA, 1949.



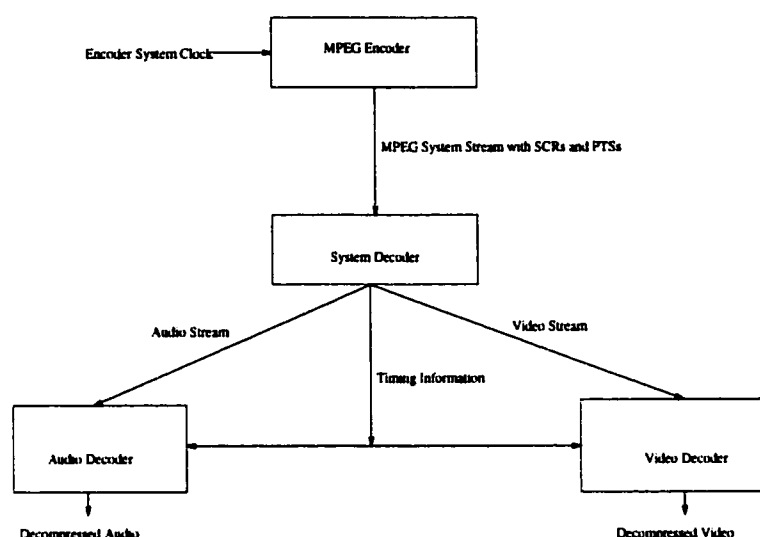
## **Appendix A**

### **MPEG Overview**

The Moving Pictures Experts Group (MPEG) committee was founded in late 1988. It is a joint committee of the International Standards Organization (ISO) and International Electrotechnical Commission (IEC). MPEG standards are generic and universal in the sense that they only specify a compressed bit-stream syntax. MPEG has developed MPEG-1 and MPEG-2 standards [12, 13]. MPEG-1 is used to represent the progressive (non-interlaced) signals. MPEG-2 is designed to represent the progressive signals and the interlaced (broadcast signals). While MPEG-1 can only directly represent two channels of sound, MPEG-2 is designed to provide the extension to 3/2 multichannel audio (3 front/2 surround loudspeakers channels) and an optimal low frequency enhancement channel [16]. MPEG specifications consists of three main parts, systems, audio, and video [17].

#### **A.0.5 Systems**

The systems part of the MPEG specifications resolves the problem of multiplexing the video and audio streams into a single system stream [17]. This is achieved by providing a syntax for transporting packets of video and audio bit-streams over digital channels and storage mediums and by providing a syntax for synchronizing video and audio streams. The synchronization of video and audio is achieved by timing information which is important to synchronize the playback of the stream by the decoder without any overflow or underflow of the decoder buffers. The synchronization is done in the MPEG standard through two parameters: the System Clock Reference (SCR) and the Presentation Timestamp (PTS). An SCR is a snapshot of the encoder system clock. The



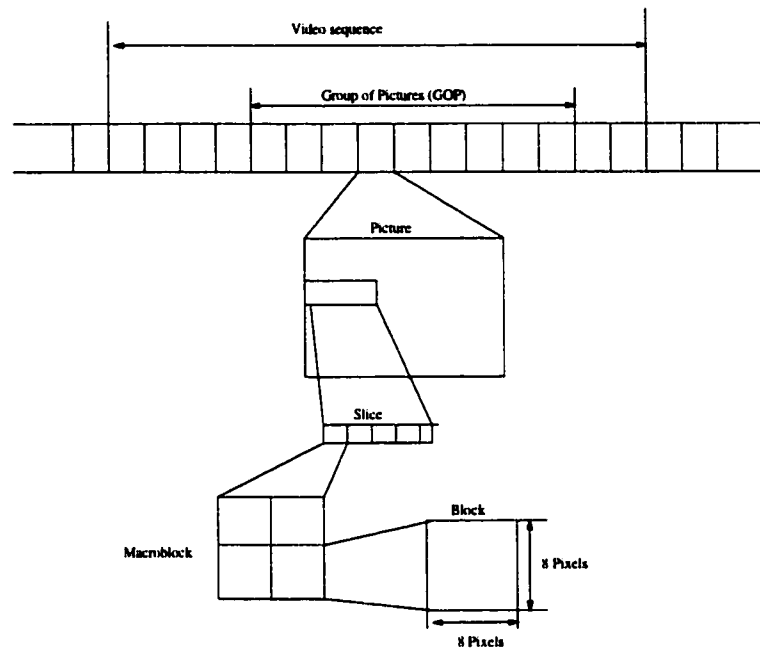
**Figure A.1 : Flow of the MPEG System Stream**

time snapshot is put into the system layer of the bit-stream which is used by the decoder to update its system clock counter.

A PTS is a snapshot of the encoder system clock that is associated with audio and video presentation units. A presentation unit refers to a decoded video picture or a decoded audio time sequence. The PTS is used to denote the time at which the video picture is to be displayed or the beginning playback time for the audio time sequence. Figure A.1 depicts the flow of the MPEG system stream.

## **Audio**

The audio part explains the syntax and semantics for three coding schemes, termed Layer 1, Layer 2, and Layer 3 coding [16, 27]. The MPEG standard defines an MPEG audio stream as series of packets. Each audio packet consists of an audio packet header and one or more audio frames. These frames are coded based on one of the tree layers. Each layer is more complex than the previous and provides better compression. These layers are based in part on subband coding and quantization and are upward compatible. Layer 1 is



**Figure A.2 : Video Hierarchical Structure**

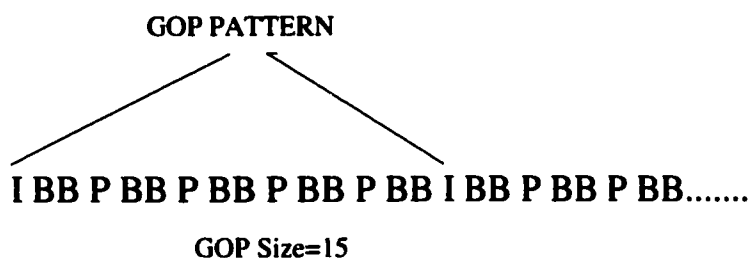
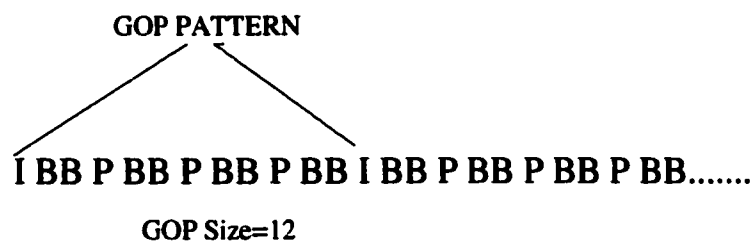
applied in Sony MiniDisc and Philips Digital Compact Cassette (DCC). Layer 2 is used in satellite broadcasting and compact disc video. Layer 3 is applied in ISDN, satellite, and Internet audio applications.

## **Video**

In this part, the syntax and semantics of the video streams are defined. The MPEG standard defines a video stream, like an MPEG audio stream, as a series of packets. Each video packet consists of a video packet header and one or more video frames. Video hierarchical structure and video compression algorithm are described next.

### **A.0.6 Video Hierarchical Structure**

An MPEG standard defines the video as a series of nested layers, which include video sequence, group of pictures (GOP), picture, slice, macroblock, and block. Figure A.2 shows this hierarchy.



**Figure A.3 : Common Pattern of Group of Pictures**

### **Video Sequence**

A video sequence starts with a sequence header, may contain additional sequence headers, and includes one or more group of pictures. The sequence ends with an end of sequence code, and the sequence may contain information about quantization tables which are needed to decode all the following group of pictures

### **Group of Pictures(GOP)**

A group of pictures consists of header and a series of one or more pictures and is intended to allow random access into the video sequence. Group of pictures size refers to the total number of pictures that are contained in a GOP. A GOP is considered an independently decodable sequence of frames if the frames are closed, the B frames start and end with either an I or P frame. For example, a GOP with the pattern IBBPBBP is closed, but with the pattern IBBPBB is not. An MPEG video stream consists of one or more GOPs. GOPs within

Display Order: I B B P B B P B B P B B I B B P .....

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Storage Order: I P B B P B B P B B I B B .....

1 4 2 3 7 5 6 10 8 9 13 11 12

**Figure A.4 : Group of Pictures Order**

a video stream do not have to follow a fixed pattern. Each frame can be of any type as will be explained next. However, a static pattern is often used throughout the entire video stream for simplicity. The most commonly used patterns of GOPs are shown in Figure A.3 and are of GOP size of 12 and 15. Figure A.4 depicts the order of the display and the storage order. In the storage order, frames within a GOP are rearranged in a way that an MPEG decoder can decompress them with minimum frame buffering and minimum processing overheads during the display. The choice of a GOP pattern is provided by the encoder and is based on the applications need for random accessibility and location of scene cuts in the video stream. The choice is also based on factors such as the amount of memory needed in the encoder and the characteristics of the movie being encoded.

### **Picture(Frame)**

A picture is considered the basic unit of display and corresponds to a single frame in a video sequence. It is divided into 16x16 macroblocks. There are three types of pictures, termed intra-pictures, predicted pictures, and bidirectional pictures. Each GOP must start with an intra-picture. Intra-pictures usually have up to 3 times as many bits as predicted pictures. Predicted pictures have about 5 times as many bits as bidirectional pictures. Within a GOP, similar picture types may not have same sizes. Even though GOPs have the same number of frames, they may not have exact number of bits.

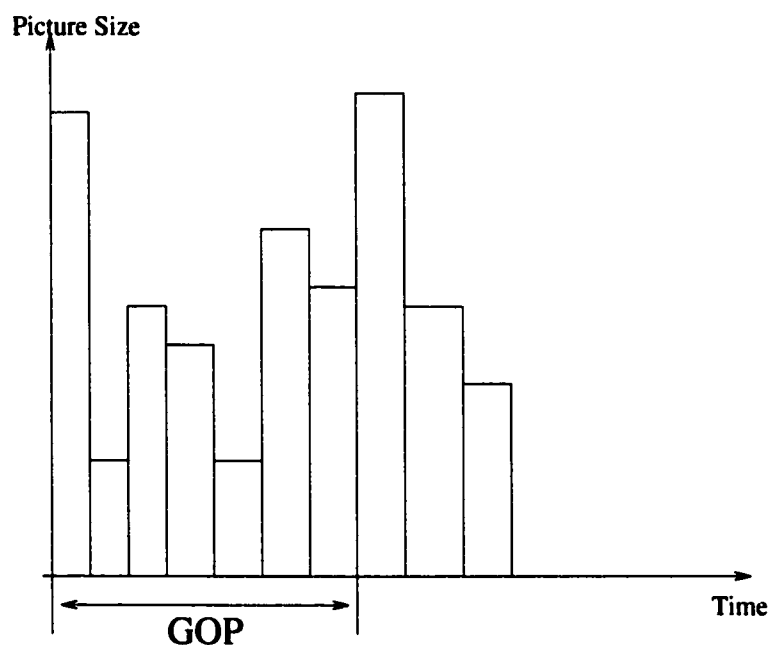


Figure A.5 : An Example of MPEG Picture Sizes

### **Intra-Pictures**

Intra-pictures, called I-pictures or I-frames, are encoded utilizing only the information existing within the picture itself. Thus, I-pictures can be displayed by itself. The main purpose of I-pictures is to provide random access points into the compressed video stream. I-pictures provide moderate compression since they are encoded independently of other pictures.

### **Predicted-Pictures**

Predicted pictures, called P-pictures or P-frames, are encoded with respect to the nearest previous I-picture or P-picture. This process is known as forward prediction and is shown in Figure. I-pictures and P-pictures are called reference pictures since they can be referenced by predicted pictures and bidirectional pictures. A P-picture cannot be displayed without the existence of the previous reference frame. P-pictures provide better compression than I-pictures, but can propagate coding errors since they are predicted from previ-

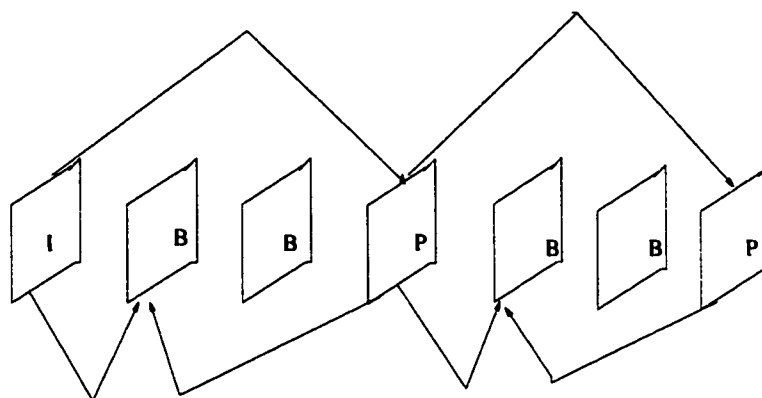


Figure A.6 : Prediction Processes

ously referenced picture.

### **Bidirectional-Pictures**

Bidirectional pictures, called B-pictures or B-frames, are encoded with respect to both the previous and the following I-picture and P-picture. This process is called bidirectional prediction and is shown in Figure A.6. To be able to decode a B-picture, the previous and the following reference frames are required. B-pictures provide the most compression, when compared to I-pictures and P-pictures. Unlike P-pictures, B-pictures do not propagate errors since they are not referenced. B-pictures also minimize the effect of noise by taking the average of the two referenced pictures.

### **Slice**

A slice consists of one or more contiguous macroblocks and can be as big as an entire picture. Each slice is considered as an independently decodable unit. The macroblocks inside a slice are ordered from left to right and top to bottom. Slices are increased for noisy transmission so that errors are more recoverable; however, increasing the number of slices might slightly worsen the compression.

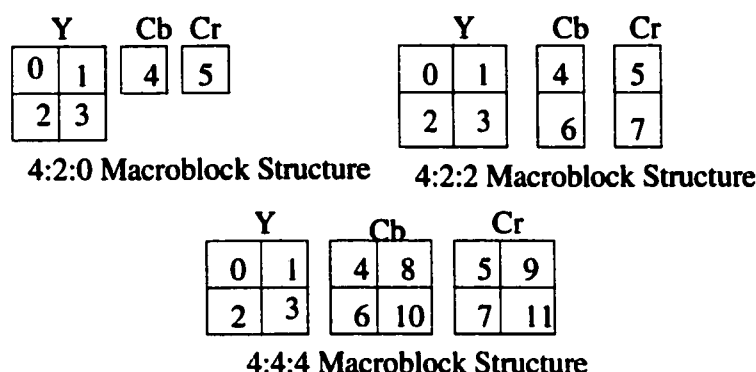


Figure A.7 : Chrominance Format for Macroblocks

### Macroblock

A macroblock consists of 16x16 pixels in a picture. It contains a luminance (Y) component, a red chrominance (Cr) component and a blue chrominance (Cb) component. There are three chrominance formats for a macroblock, namely the 4:2:0, 4:2:2, 4:4:4 formats as shown in Figure A.7. The numbers represent the order of the blocks in a video stream.

### Block

A block is an 8x8 pixel matrix in a picture and is the basic coding unit in the MPEG standard. It can be either luminance, red chrominance, or blue chrominance.

### A.0.7 Video Compression Algorithm

The MPEG video compression algorithm provides a high-degree of compression by exploiting the large temporal and spatial redundancies which exist within an image. Figure A.8 depicts the major steps of transform coding and motion compensation involved in the MPEG video compression algorithm.



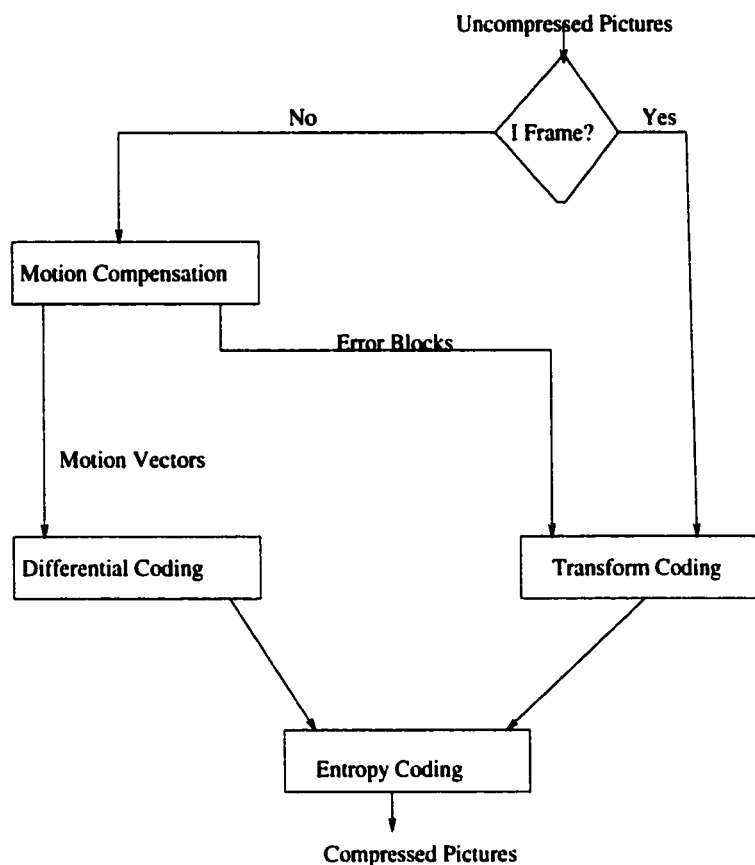


Figure A.8 : MPEG Video Compression Algorithm

### Transform Coding

Transform coding consists of three main steps: discrete cosine transform (DCT), quantization, and run-length encoding. Blocks are the units of transform coding. All I-pictures blocks and error blocks which come from the motion compensation step are fed into the transform coding as shown in Figure A.8. The blocks are first transformed from a spatial domain into a frequency domain using DCT. The coefficient in the upper left corner of a block, called the direct current (DC) coefficient, which captures the average brightness of the block, is encoded relative to the DC coefficients of the previous block. Differential coding is used to encode the DC coefficients of successive blocks. The remaining 63 coefficients in the block, called alternating current (AC), which

capture the detail within the block, are quantized to eliminate high frequency coefficients. They are then scanned in a zig-zag manner to produce long-runs of zeros. Finally, they are run-length and entropy coded.

### **Motion Compensation**

Motion compensation is a technique for improving video compression of P-pictures and B-pictures by removing temporal redundancy. Macroblocks are the units for motion compensation. Each macroblock of P-pictures and B-pictures is temporally interpolated from the corresponding reference pictures. An error macroblock, which contains only the difference between the matching macroblock and the macroblock to be encoded, is produced. The motion compensation process computes up to two motion vectors for each macroblock, which indicate the positions of the encoded macroblock in the matching reference picture. Each error macroblock is partitioned into six blocks, four luminance and two chrominance, and passed to the transform coding step. However, motion vectors are encoded using differential and entropy coding. If a macroblock in P-pictures or B-pictures cannot be efficiently encoded by motion compensation, the macroblock is encoded in the same way as a macroblock in an I-picture, using transform coding.