Design and Implementation of Scalable Continuous Media Servers[*]

Shahram Ghandeharizadeh
Computer Science Department
University of Southern California
Los Angeles, California 90089
shahram@perspolis.usc.edu

Richard Muntz
Computer Science Department
University of California-Los Angeles
Los Angeles, California 90024
muntz@cs.ucla.edu

June 13, 1997

Abstract

During the past decade, the information technology has evolved to store and retrieve continuous media data types, e.g., audio and video clips. Unlike the traditional data types (e.g., text), continuous media consists of a sequence of quanta, either audio samples or video frames, that convey meaning when presented at a pre-specified rate. A challenging task when implementing these servers is to guarantee retrieval and delivery of a clip at its pre-specified rate. Continuous media servers are expected to play a major role in many application domains including library information systems, entertainment technology, educational applications, etc.

The focus of this study is on the disk subsystem of a server and techniques that both enhance its performance and ensure its scalability. By scalable, we mean that the system can grow incrementally as the requirements of an application grows in order to avoid a degradation in performance. This study details data placement techniques across both (1) the zones of a disk in order to trade bandwidth for storage, and (2) multiple disks to enable the system to scale. In addition, we describe scheduling techniques to (1) trade memory for disk bandwidth and vice versa, and (2) merge multiple requests referencing a single clip in order to increase the number of simultaneous displays supported by the system. We present performance results from Mitra, an experimental prototype that realizes an implementation of the presented techniques. These results demonstrate the feasibility of a scalable server.

1 Introduction

Advances in computer processing and storage performance and in high speed communications has made it feasible to consider continuous media (e.g., audio and video) servers that scale to thousands of concurrently active clients. The principle characteristics of continuous media is their sustained bit rate requirement. If a system delivers a clip at a rate lower than its pre-specified rate without special precautions (e.g., pre-fetching), the user might observe frequent disruptions and delays with video and random noises with audio. These artifacts are collectively termed *hiccups*. For example, CD-quality audio (2 channels with 16 bit samples at 44 kHz) requires 1.4 Megabits per second (Mbps). Digital component video based on the CCIR 601 standard requires 270 Mbps for its continuous

[*]This research was supported in part by grants from Hewlett-Packard, IBM grant SJ92488, and NSF grants IRI-9110522, IRI-9258362 (NYI award), CDA-9216321, and ERC grant EEC-9529152.

display. These bandwidths can be reduced using compression techniques due to redundancy in data. Compression techniques are categorized into lossy and lossless. Lossy compression techniques encode data into a format that, when decompressed, yields something similar to the original. With lossless techniques, decompression yields the original. Lossy compression techniques are more effective in reducing both the size and bandwidth requirements of a clip. For example, with the MPEG standard, the bandwidth requirement of CD-quality audio can be reduced to 384 Kilobits per second. MPEG-1 reduces the bandwidth requirement of a video clip to 1.5 Mbps. With some compression techniques such as MPEG-2, one can control the compression ratio by specifying the final bandwidth of the encoded stream (ranging from 3 to 15 Mbps). However, there are applications that cannot tolerate the use of lossy compression techniques, e.g., video signals collected from space by NASA [Doz92].

Most of the compression schemes are Constant Bit Rate (CBR) but some are Variable Bit Rate (VBR). With both techniques, the data must be delivered at a pre-specified rate. Typically CBR schemes allow some bounded variation of this rate based on some amount of memory at the display. With VBR, this variation is not bounded. The VBR schemes have the advantage that, for the same average bandwidth as CBR, they can maintain a more constant quality in the delivered image by utilizing more megabits per second when needed, e.g., when there is more action in a scene.

The size of a compressed video clip is quite large by most current standards. For example, a two hour MPEG-2 encoded video clip, requiring 3 Mbps, is 2.6 Gigabytes in size. (In this paper, we focus on video due to its significant size and bandwidth requirements that are higher than audio.) To reduce cost of storage, a typical architecture for a video server employs a hierarchical storage structure consisting of DRAM, magnetic disks, and one or more tertiary storage devices. As the different levels of the hierarchy are traversed starting with memory, the density of the medium and its latency increases while its cost per megabyte decreases. It is assumed that all video clips reside on the tertiary storage device. The disk space is used as a temporary staging area for the frequently accessed clips in order to minimize the number of references to tertiary. This enhances the performance of the system. Once the delivery of a video clip has been initiated, the system is responsible for delivering the data to the settop box of the client at the required rate so that there is no interruption in service. The settop box is assumed to have little memory so that it is incumbent on the server and network to deliver the data in a "just in time" fashion. (We will specify this requirement more precisely and formally later.) Note that the system will have some form of admission control and while it is reasonable to make a request wait to be initiated, once started, delivery in support of a hiccup-free display is required. An important concept is that of a *stream* which is one continuous sequence of data from one clip. At any point in time a stream is associated with a specific offset within the clip. Note that two requests for the same clip that are offset by some time are two different streams.

In this paper we are concerned mainly with the disk storage subsystem; the layout of data and scheduling of disk I/O to ensure continuous delivery of object streams. We focus on neither the management of tertiary storage nor the networking component of the system (see [And93, GDS95, GS94]). We assume a set of videos stored on disk and describe techniques to stage data from disk to memory buffers. The models are described assuming CBR encoded data. (Section 3.3 details modifications of this work in support of VBR.) We assume therefore that the communication network will transmit a stream at a constant bit rate[1]. In this context, the disk subsystem is responsible for delivering data from the disk image of a clip to buffers such that, if R_C is the play-out rate of the

[1]This is a simplifying assumption. There will be some amount of variation in the network transmission. To account for this, some amount of buffering is required in the network interface as well as the settop box. We do not consider this further. See [And93] for a detailed treatment of buffer requirements to mask delays in end to end delivery of continuous media streams.

Term	Definition
$\mathcal{N}$	Maximum number of simultaneous displays (throughput)
ℓ	Maximum latency time assuming fewer than $\mathcal{N}$ active displays
R_C	Display bandwidth requirement (Consumption rate)
R_D	Bandwidth of a disk
g	Number of groups with GSS
T_p	Duration of a time period
m	Number of zones in a disk drive
Z_i	Zone i of a disk drive, where $0 \leq i < m$
$\mathcal{R}(Z_i)$	Transfer rate of Z_i, in MB/s
$\mathcal{B}(Z_i)$	Block size of Z_i with VARB
$\mathcal{B}$	Fixed block size with FIXB
T_{scan}	Time to perform one sweep of m zones

Table 1: List of terms used repeatedly and their respective definitions

object, in Mbps, then at τ seconds after the start of the stream, at least $R_C \times \tau$ megabits must have been delivered to the network interface. This ensures that the network never *starves* for data.

There are a number of ways of scheduling streams depending on the frequency of access to each video. The simplest would be to define a fixed schedule a priori of start times. At the other extreme, one could start a new stream for each request at the earliest possible time (consistent with available resources). Video-on-demand implies the latter policy but there are variations that will be described in Section 3.2. With n videos, we define $f_i(t)$ to be the frequency of access to the i-th video as a function of time. This notation emphasizes that the frequency of access to videos varies over time. Some variations might be periodic over the course of a day, or some other period, while other variations will not be repetitive (e.g., a video becoming old over a couple of months). It is important therefore to design a system that can respond effectively to both variations and do so automatically. As we shall see, striping of objects across disks in an array alleviates this problem.

The following are the main performance metrics that constitute the focus of this study.

1. Throughput: Number of simultaneous displays supported by the system.
2. Startup latency: Amount of time elapsed from when a request arrives referencing a clip until the time the server starts to retrieve data on behalf of this request.

Startup latency corresponds roughly to the usual basic measure of response time. Alternative applications might sacrifice either startup latency in favor of throughput or vice versa. For example, a service provider that supports video-on-demand might strive to maximize its throughput by delaying a user by tens of seconds to initiate the display. If the same provider decides to support VCR functionalities, e.g., pause, resume, fast forward and rewind with scan, then the startup latency starts to become important. This is because a client might not tolerate tens of seconds of latency for a fast forward scan. The impact of startup latency becomes more profound with non-linear digital editing systems. To explain this, consider the role of such a system in a news organization, e.g., CNN. With sports, a producer tailors a presentation together based on highlights of different clips, e.g., different events at the Olympics. Moreover, the producer might add an audio narration to logically tie these highlights together. Upon the display of the presentation, the system is required to display: (1) the audio narration in synchrony with video, and (2) the highlights in sequence, one after another, with no delay between two consecutive highlight. If the system treats the display of each highlight as a request for a stream then the startup latency must be minimized between each stream to provide the desired effect, see [CGS95] for a discussion of this and other techniques.

The rest of this paper is organized as follows. In Section 2, we provide an overview of the current disk technology. Subsequently, Section 3 describes scheduling techniques in support of a continuous display. In Section 4, we focus on a multi-disk architecture and striping techniques. Section 5 describe an evaluation of the proposed techniques in an experimental prototype, Mitra. Brief conclusions are offered in Section 6.

2 Overview of Magnetic Disks

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, see [RW94] for details. A single disk contains a number of platters, as many as sixteen at the time of this writing. Each platter surface has an associated disk head responsible for reading and writing data. Each platter stores data in a series of tracks. A single stack of tracks at a common distance from the spindle is termed a cylinder. To access the data stored in a track, the disk head must be positioned over it. The operation to reposition the head from the current track to the desired track is termed *seek*. Next, the disk must wait for the desired data to rotate under the head. This time is termed *rotational latency*.

The seek time is a function of the distance traveled by the disk arm [BG88, GHW90, RW94]. Several studies have introduced analytical models to estimate seek time as a function of this distance. To be independent from any specific equation, this study assumes a general seek function. Thus, let Seek(c) denote the time required for the disk arm to travel c cylinders to reposition itself from cylinder i to cylinder $i + c$ (or $i - c$). Hence, Seek(1) denotes the time required to reposition the disk arm between two adjacent cylinders, while Seek($\#cyl$) denotes a complete stroke from the first to the last cylinder of a disk that consists of $\#cyl$ cylinders. Typically, seek increases linearly distance except for small number of cylinders [BG88, RW94]. For example, the model used to describe the seek characteristic of Seagate ST31200W disk, consisting of 2697 cylinders, is:

$$Seek(c) = \begin{cases} 1.5 + (0.510276 \times \sqrt{c}) & \text{if } c < 108 \\ 6.5 + (0.004709 \times c) & \text{otherwise} \end{cases} \tag{1}$$

A trend in the area of magnetic disk technology is the concept of *zoning*. Zoning increases the storage capacity of a disk by storing more data on the tracks that constitute the outer regions of the disk drive. With a fixed rotational speed for the disk platters, this results in a disk with variable transfer rate where the data in the outermost region if produced at a faster rate. Figure 1 shows the transfer rate of the 23 different zones that constitute a Seagate disk drive. (Techniques employed to gather these numbers are reported in [GSZ95].)

3 Continuous Display

We start by describing the continuous display of an object with a system that treats the d available disks as a single disk drive. When we state that a block is assigned to the disk, we imply that the

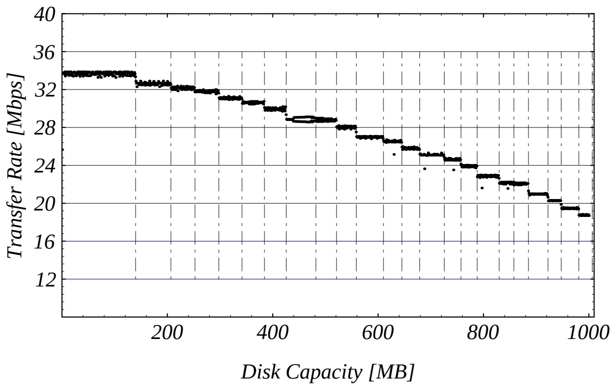


Figure 1: Zone characteristics of the Seagate ST31200W magnetic disk

Mbps	Megabits per second.
Block	Amount of data retrieved per time period on behalf of a display.
Fragment	Fraction of a block assigned to one disk of a cluster that contains the block.
Time period	Amount of time required to display a block.
Startup latency	Amount of time elapsed from when a client requests the display of an object to the onset of the display. It is denoted as ℓ.
Stride	Number of disk drives between the first fragment of block X_i and the first fragment of block X_{i+1}. It is denoted as k.

Table 2: Defining terms

block is declustered [GRAQ91, BGMJ94] across the d disks. Each piece of this block is termed a fragment and is laid out contiguously on the disk. When we state that a block is read from the disk, we imply that d disks are activated simultaneously to produce the fragments that constitute the block. In the following, assume that a disk provides a constant bandwidth R_D; this assumption is relaxed in Section 3.1. Moreover, assume CBR encoded clips; Section 3.3 describes modifications in support of VBR.

To support continuous display of an object X that requires a constant playout rate R_C, the system partitions X into n equi-sized blocks: X_0, X_1, ..., X_{n-1}, where n is a function of the block size ($\mathcal{B}$) and the size of X. We assume that a block is the unit of transfer from disk to main memory. The display time of a block is termed a time period and defined as:

$$T_p = \frac{\mathcal{B}}{R_C} \tag{2}$$

A simple technique to display object X is as follows. Upon the arrival of a request referencing object X, the system stages X_0 in memory and initiates its display. Prior to completion of a time period, the system initiates the retrieval of X_1 into memory in order to ensure a continuous display. This process is repeated until all blocks of an object have been retrieved and displayed. Assuming that all objects have the same bandwidth requirement as X, a disk can support $\mathcal{N}$ simultaneous object displays:

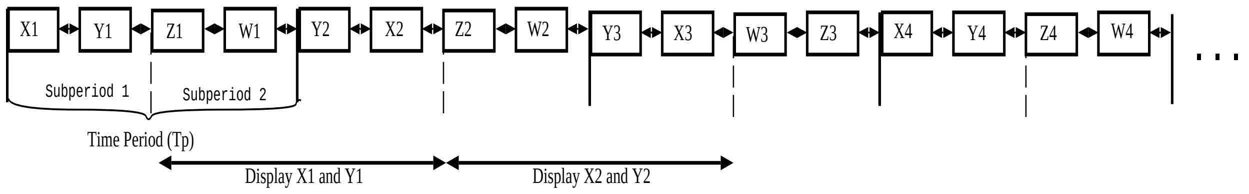


Figure 2: Continuous display with GSS

$$\mathcal{N} = \lfloor \frac{T_p}{seek(\#cyl) + latency + \frac{\mathcal{B}}{\mathcal{R}_D}} \rfloor \tag{3}$$

For simplicity, we assume the maximum seek time, i.e., seek($\#cyl$), in order to guarantee a continuous display. Moreover, we ignore the track switching time incurred when a block spans multiple physical tracks. It is trivial to extend the presented equations to account for this overhead.

Seek time is an overhead that, if not controlled, can greatly reduce the number of simultaneous displays supported by the disk. In the simple technique just described, the disk performs $\mathcal{N}$ seeks per time period. Hence the percentage of time reserved for seeking in order to guarantee no hiccups is: $\frac{\mathcal{N} \times Seek(\#cyl)}{T_p} \times 100$. By substituting T_p from Equation 2, we obtain the percentage of wasted disk bandwidth:

$$wasteful = \frac{\mathcal{N} \times Seek(\#cyl) \times R_C}{\mathcal{B}} \times 100 \tag{4}$$

By reducing this percentage, the system can support a higher number of simultaneous displays. Two variables can be manipulated to reduce this percentage: 1) decrease the distance traversed by a seek, and/or 2) increase the block size ($\mathcal{B}$). A larger block size results in a higher memory requirement per stream. Below, we describe a technique that minimizes the impact of seeks. We analyze the role of a larger block size in the context of this technique.

Group Sweeping Scheme [YCK92, YCK93, TPBG93, BGM95], GSS, reduces the impact of seeks by minimizing the number of cylinders traversed by each seek operation. In its simplest form, the data that is read during one time period is delivered for display during the next time period. During each time period, data for each active stream is read from the disk into main memory. Simultaneously, the data read during the previous time period is transmitted over the network to display stations. Since no data block is required to be present in memory until the end of the time period in which it is fetched, the disk schedule is able to optimize the retrieval of the blocks for each time period (using disk scheduling policies such as SCAN).

In a more general form, GSS partitions $\mathcal{N}$ active requests of a time period into g groups. This divides a time period into g subperiods, each corresponding to the retrieval of $\frac{\mathcal{N}}{g}$ blocks. (For simplicity, we assume $\mathcal{N}$ is a multiple of g. It is straightforward to modify the presented equations when this assumption is violated.) The movement of the disk head to retrieve the blocks within a group abides by the SCAN algorithm, in order to reduce the incurred seek time in a group. To illustrate, consider Figure 2 where g=2 and $\mathcal{N}$=4. The blocks X_1 and Y_1 are retrieved during the first subperiod. Their display is initiated at the beginning of subperiod 2 and lasts for two subperiods. As demonstrated, while it is important to preserve the order of groups across the time periods, it is not necessary to maintain the order of block retrievals in a group. The simple technique introduced

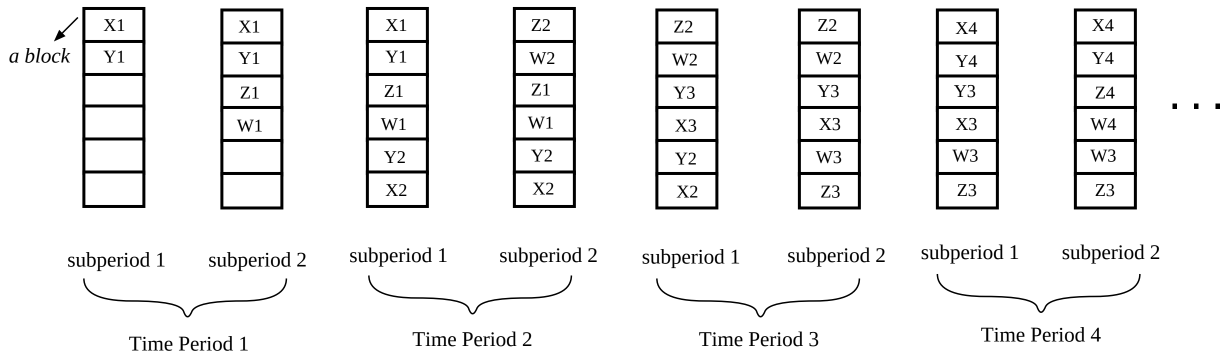


Figure 3: Memory requirement per subperiod

at the beginning of this section can be viewed as a special case of GSS with g=$\mathcal{N}$.

To compute the block size with GSS, we first compute the total duration of time contributed to seek times during a time period. Assuming the $\frac{\mathcal{N}}{g}$ blocks retrieved during a subperiod are distributed uniformly across the disk surface, the disk incurs a seek time of $Seek(\frac{\#cyl}{\frac{\mathcal{N}}{g}})$ between every two consecutive block retrievals. This assumption maximizes the seek time according to the seek model of Equation 1, capturing the worst case scenario. Since $\mathcal{N}$ blocks are retrieved during a time period, the system incurs $\mathcal{N}$ seek times in addition to $\mathcal{N}$ block retrievals during a period, i.e., $T_p = \frac{\mathcal{N}\mathcal{B}}{R_D} + \mathcal{N} \times Seek(\frac{\#cyl \times g}{\mathcal{N}})$. By substituting T_p from Equation 2 and solving for $\mathcal{B}$, we obtain:

$$\mathcal{B}_{gss} = \frac{R_C \times R_D}{R_D - \mathcal{N} \times R_C} \times \mathcal{N} \times Seek(\frac{\#cyl \times g}{\mathcal{N}}) \tag{5}$$

The bound on the distance between two blocks retrieved consecutively is reduced by a factor of $\frac{\mathcal{N}}{g}$, noting that $g \leq \mathcal{N}$.

The amount of memory required to support $\mathcal{N}$ displays depends on the technique used to manage memory. One approach might maintain a shared pool of free blocks among active displays. Each display allocates blocks from the shared pool on demand. With this scheme, assuming the memory is allocated and deallocated in units of size $\mathcal{B}$, the total amount of memory required by the system is:

$$(\mathcal{N} + \frac{\mathcal{N}}{g}) \times \mathcal{B} \tag{6}$$

The $\mathcal{N}$ active displays require $\mathcal{N}$ blocks and, during the subperiod that a group is reading blocks, $\frac{\mathcal{N}}{g}$ additional memory frames are required. To illustrate, recall our example with g=2 and $\mathcal{N}$=4. From Equation 6, this paradigm requires six blocks of memory, see Figure 3.

Assuming fewer than $\mathcal{N}$ active displays, the maximum possible startup latency occurs when the request arrives when the subperiod with the empty slot has just started. In this case, the delay is the summation of one time period and the duration of a subperiod ($\frac{T_p}{g}$):

$$\ell_{gss} = T_p + \frac{T_p}{g} \tag{7}$$

It may appear that GSS with g=1 results in a higher startup latency than simple (g=$\mathcal{N}$). However, this is not necessarily true because the duration of the time period is different with these two techniques for a desired throughput (value of $\mathcal{N}$). This is due to a choice of different block size.

Block size	$\mathcal{N}$	memory required	Max ℓ Seconds	Wasted disk bandwidth (%)	Cost ($) per Stream
8K	5	80K	0.012	88.9	50.48
16K	10	320K	0.167	77.7	25.96
32K	16	1M	0.333	64.7	17.50
64K	24	3M	0.667	47.5	14.17
128K	32	8M	1.333	30.0	15.31
256K	37	18.5M	2.667	19.1	21.76
512K	41	41M	5.333	10.3	36.10
1M	43	96M	10.666	6.0	72.79

Table 3: Continuous display with GSS, g=1

Block size	$\mathcal{N}$	memory required	Max ℓ Seconds	Wasted disk bandwidth (%)	Cost ($) per Stream
8K	1	16K	0.006	97.6	250.48
16K	3	64K	0.080	93.2	83.97
32K	6	224K	0.167	86.7	42.79
64K	11	768K	0.333	75.9	24.82
128K	18	2.4M	0.667	60.6	17.89
256K	26	6.8M	1.333	43.1	17.46
512K	33	17M	2.667	27.8	23.03
1M	38	39M	5.333	16.9	37.37

Table 4: Continuous display with simple (GSS, g=$\mathcal{N}$)

To illustrate, assume a database that consists of a single media type that requires 1.5 Mbps for its display. The target disk has a transfer rate of 68.6 Mbps, maximum seek time of 17 milliseconds, minimum seek time of 2 milliseconds, and a maximum rotational latency of 8.33 milliseconds. Then, for example, GSS with g=1 supports 37 simultaneous displays, requiring 18.5 megabyte of memory while incurring a maximum latency of 2.667 seconds, see Table 3. Simple (GSS with g=$\mathcal{N}$) requires twice as much memory and incurs twice as high a latency to support 38 simultaneous displays, see Table 4. In both tables, we compute the cost per stream assuming $30 per megabyte of memory and a disk drive that costs $250. The cost per stream drops as a function of block size to a point beyond which disk efficiency increases are marginal when compared with the cost of the increased memory required.

3.1 Multi-Zone Disks

To guarantee a continuous display, the techniques described in the previous section must assume the transfer rate of the innermost zone. An alternative is to constrain the layout of data across the disk surface and the schedule for its retrieval [Bir95, HMM93, GKSZ96]. This section describes three such techniques. With all techniques, there is a tradeoff between throughput, amount of required memory and the incurred startup latency. We illustrate these tradeoffs starting with a brief overview of two techniques and then discussing the third approach in detail.

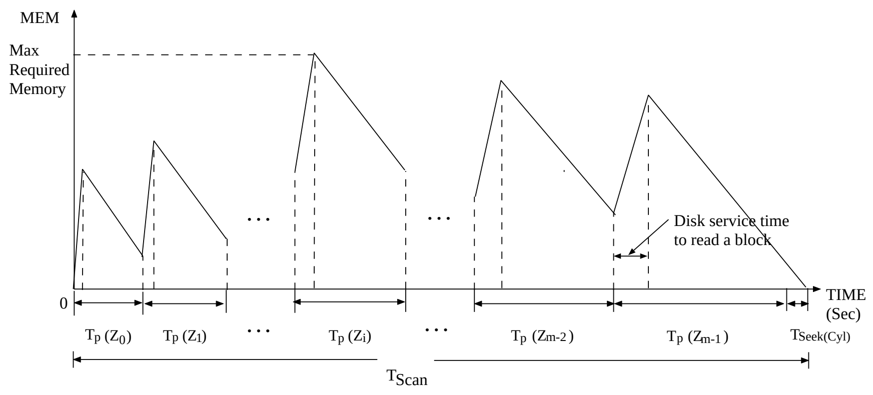


Figure 4: Memory required on behalf of a display with FIXB

Track pairing [Bir95], organizes all tracks into pairs such that the total capacity of all pairs is identical. When displaying a clip, a track-pair is retrieved on its behalf per time period (alternative schedules are detailed in [Bir95]). Similar to the GSS discussions of Section 3, the system can manipulate the retrieval of physical tracks on behalf of multiple active displays to minimize the impact of seeks.

Assuming that the number of tracks in every zone is a multiple of some fixed number, [HMM93] constructs Logical Tracks (LT) from the same numbered physical track of the different zones. The order of tracks in a LT is by zone number. When displaying a clip, the system reads a LT on its behalf per time period. An application observes a constant disk transfer rate for each LT retrieval. This forces the disk to retrieve data from the constituting physical tracks in immediate succession by zone order. Recall that we assumed a logical disk drive that consists of d physical disks. If d equals the number of zones (m), the physical tracks of a LT can be assigned to a different disk. This facilitates concurrent retrieval of physical tracks that constitute a LT. Similarly, to facilitate concurrent retrieval with track pairing, if d is an even number then the disks can be paired such that track i of one disk is paired with track $\#cyl - i$ track of another disk.

The third approach as detailed in [GKSZ96] organizes a clip at the granularity of blocks (instead of tracks). We describe two variations of this approach that guarantee continuous display while harnessing the average transfer rate of m zones, namely, FIXed Block size (FIXB) and VARiable Block size (VARB) [GKSZ96]. These two techniques assign the blocks of an object to the available zones in a round-robin manner, starting with an arbitrary zone. With both techniques, there are a family of scheduling techniques that ensure a continuous display. One might require the disk to retrieve m blocks of an object assigned to m different zones in one sweep. Assuming that $\mathcal{N}$ displays are active, this scheduling paradigm would result in $\mathcal{N}$ disk sweeps per time period and substantial amount of buffer space. (With this scheduling paradigm, VARB would be similar to LT [HMM93].) An alternative scheduling paradigm might multiplex the bandwidth of each zone among the $\mathcal{N}$ displays and visit them in a round-robin manner. It requires the amount of data retrieved during one scan of the disks on behalf of a display (m time periods that visit all the zones, starting with the outermost zone) to equal that consumed by the display. This reduces the amount of required memory. However, it results in a higher startup latency. We will focus on this scheduling paradigm for the rest of this section.

To describe the chosen scheduling technique assume the simple display scheme (GSS with $g = \mathcal{N}$) of Section 3. We choose the block size (the unit of transfer from a zone on behalf of an active display) to be a function of transfer rate of each zone such that the higher transfer rate of fast

$\mathcal{N}$	*FIXB*				*VARB*			
	Mem. (MB)	Max ℓ (Sec)	% wasted disk space	% Avg wasted band.	Mem. (MB)	Max ℓ (Sec)	% wasted disk space	% Avg wasted band.
1	0.007	0.44	58.0	94.1	0.011	0.44	40.4	94.3
2	0.023	0.92	58.0	88.2	0.044	0.92	40.4	88.6
4	0.107	2.10	58.0	76.5	0.192	2.08	40.4	77.3
8	0.745	6.03	58.1	53.1	1.059	5.88	40.4	54.6
10	1.642	9.66	58.1	41.4	2.078	9.26	40.5	43.2
12	3.601	16.15	58.2	29.7	4.040	15.06	40.6	31.9
13	5.488	21.77	58.3	23.9	5.759	19.84	40.4	26.2
14	8.780	31.05	58.0	18.0	8.511	27.26	40.6	20.5
15	15.515	49.23	58.4	12.1	13.481	40.34	41.0	14.8
16	35.259	100.9	58.3	6.3	24.766	69.53	41.3	9.2
17	536.12	1392.5	73.8	0.4	72.782	192.4	42.2	3.5

Table 5: Seagate ST31200W disk

zones compensates for that of the slower zones. The display time of the block that is retrieved from the outermost zone (Z_0) on behalf of a display exceeds the duration of a time period ($T_P(Z_0)$), see Figure 4. Thus, a portion of the block remains memory resident in the available buffer space. During the time period when the innermost zone is active, the amount of data displayed exceeds the block size, consuming the buffered data. In essence, the display of data is synchronized relative to the retrieval from the outermost zone. This implies that if the first block of an object is assigned to the zones starting with a zone other than the innermost zone, then its display might be delayed relative to its retrieval in order to avoid hiccups.

Both FIXB and VARB waste disk space due to: (1) a round-robin assignment of blocks to zones, and (2) a non-uniform distribution of the available storage space among the zones (in general, the outermost zones have a higher storage capacity relative to the innermost zones). Once the storage capacity of a zone is exhausted, no additional blocks can be assigned to the remaining zones because it would violate the round-robin assignment of blocks. Table 5 compares FIXB with VARB by reporting on the amount of required memory, the maximum incurred startup latency assuming fewer than $\mathcal{N}$ active displays, and the percentage of wasted disk space and disk bandwidth with the Seagate ST31200W disk. The percentage of wasted disk space with both FIXB and VARB is dependent on the physical characteristics of the zones. While VARB wastes a lower percentage of the Seagate disk space, it wastes a higher percentage of another analyzed disk space (HP C2247) [GKSZ96].

If we assumed the transfer rate of the innermost zone as the transfer rate of the entire disk and employed the discussion of Section 3 to guarantee a continuous display, the system would support twelve simultaneous displays, require 16.09 megabyte of memory, and incur a maximum startup latency of 7.76 seconds. A system that employs either FIXB or VARB supports twelve displays by requiring less memory and incurring a higher startup latency, see sixth row of Table 5. For a high number of simultaneous displays (16 and 17, see last two rows of Table 5), when compared with FIXB, VARB requires a lower amount of memory and incurs a lower maximum startup latency. This is because VARB determines the block size as a function of the transfer rate of each zone while FIXB determines the block size as a function of the average transfer rate of the zones, i.e., one fix-sized block for all zones. Thus, the average block size chosen by VARB is smaller than the block size chosen by FIXB for a fixed number of users. This reduces the amount of time required to scan the disk (T_{Scan} in Figure 4) which, in turn, reduces both the amount of required memory and the maximum startup latency, see [GKSZ96] for details.

A system designer is not required to configure either FIXB or VARB using the vendor specified zone characteristics. Instead, one may logically manipulate the number of zones and their specifications by either (1) merging two or more physically adjacent zones into one and assume the transfer rate of this logical zone to equal that of the slowest participating physical zone, or (2) eliminate one or more of the zones. For example, one might merge zones 0 to 4 into one logical zone, zones 5 to 12 in a second logical zone, and eliminate zones 13 to 23 all together. With this logical zone organization, VARB supports 17 simultaneous displays by requiring 10 megabytes of memory, and observing a maximum startup latency of 6.5 Seconds[2]. Compared with a system that assumes the transfer rate of the innermost zone as the transfer rate of the entire disk, the throughput of the system is increased by 40% at the expense of wasting 35% of the available disk space. A more intelligent arrangement might even outperform this one. The definition of outperform is application dependent. A configuration planner that employs heuristics to strike a compromise between the conflicting factors is described in [GKSZ96].

3.2 Batching of Requests

The areal density of disks is inreasing much more rapidly than the tranmission bandwidth or the reduction in seek times. This trend is predicted to continue in the forseeable future and thus disk bandwidth, as opposed to disk storage capacity, is the more scarce resource. It is reasonable then to consider ways to reduce the bandwidth required. There are three main approaches to reduction of disk bandwidth:

- *batching of requests [DSST94, DSS94, OBRS94, WSY95]:* in this method, requests are delayed until they can be merged with other requests for the same video. These merged streams then form one physical stream from the disk and consume only one set of buffers. Only on the network will the streams split at some point for delivery to the individual display stations.
- *buffering [DD$^+$95, KRT94, KRT95] :* the idea here is that if one stream for a video lags another stream for the same video by only a short time interval, then the system could retain the portion of the video between the two in buffers. The lagging stream would read from the buffers and not have to read from disk.
- *adaptive piggybacking [GLM95]:* in this approach streams for the same video are adjusted to go slower or faster by a few percent, such that it is imperceptable to the viewer, and the streams eventually merge and form one physical stream from the disks.

Batching comes in several flavors distinguished by the rule used to trigger the start of a new stream (shared by the waiting customers). One method is to start a new stream when a certain number of customers accumlate in the waiting queue. The problem with this scheme is that the waiting time for a customer can be arbitrarily long. Another simple scheme is to start a new stream for a video at scheduled intervals, e.g., every 5 minutes. This scheme has the advantage of determining a priori all of the streams for a given video and this perfect predictability can be used to enhance the layout and scheduling of the streams [OBRS94]. A potential disadvantage is that streams that have no associated customers are scheduled and allocated resources. Such streams could be eliminated but then the advantage of layout and a priori scheduling are reduced as well. The fixed stream start

[2]The amount of required memory is lower than that required by the approach that assumes the transfer rate of the innermost zone for the entire disk, see the previous paragraph.

times are very inflexible and can be wasteful in some cases. For example, suppose at a scheduled start time, there is only one customor waiting which arrived only a few seconds previously. It is tempting to skip that scheduled stream and make that one customor wait for the following scheduled initiation time. A more flexible scheme is time based in which each customer is guaranteed a maximum wait time, T, for the start of a video. When a customer arrives and it is the only one waiting for a particular video, a clock is started and when it reaches the bound, T, a stream starts. Any requests for the same video that arrive in the meantime share that same stream. In this manner, customers have bounded wait time and useless streams are eliminated.

Bridging of streams using buffers is a tempting idea. If one stream lags another of the same video, the frames read into memory for the first stream can be held in buffers for the lagging stream. The problem with this approach is the amount of memory required to hold even a small segment of a video. For example, just 30 seconds of MPEG1 video would require approximately 5.5 MB of buffer space. (This number should be compared with the 3 MB required for all 24 streams in the optimal design described in Section 3, Table 3.) There have been several studies of how to design buffer replacement strategies for video servers which claim to achieve significant reduction in the amount of buffer space required to support a given number of streams, e.g., [KRT94].

Adaptive piggybacking is the name given in [GLM95] to a method of merging streams by slowing down a leading stream and/or speeding up a lagging stream until they merge. The magnitude of the variation in playout rate is to be kept small enough that it is not perceived by the viewer. While this proposal is often met with some skepticism, it turns out that 5% or more variation in playout rate is quite feasible and is done routinely for a number of reasons. For example, for a movie to fit within the time constraints of a TV program it will sometimes be stretched or shortened. Further evidence that a 5% change in rate is not perceptable is that 24 frame per second american movies are shown in Europe at 25 frames per second with no alteration or compensation.

To apply this technique requires that a copy (*trick file* as it is called) with the faster rate version and slower rate version be available to choose from to serve a particular stream. Then a scheduling algorithm is required to determine which streams should be delivered at which rate. The objective being to have streams merge and reduce the I/O bandwidth consumed as much as possible. In [GLM95] several scheduling algorithms are described. One of the simplest is as follows. When a request arrives, if there is a request for the same video that started less than T units of time earlier and is consuming data from the slow trick file, then this video is scheduled to consume data from the trick file that corresponds to a fast display rate until the two streams merge. If no such stream was started within T seconds of the current time, then this video is scheduled to consume data from the slow trick file (so that possibly another stream for the same video will start within T seconds and it can be merged with that stream). Of course the maximum gain is obtained from merging streams early in the video and it is reasonable therefore to consider only having the fast and slow versions of a video for the first m minutes. Analysis is [GLM95] shows that for 100 minute movies, having the replicated versions on disk for the first 10 minutes can reduce disk bandwidth by 30 to 40 percent. It is also possible to combine piggybacking with other schemes such as batching. For example, one could batch with a timeout of 1 minute and still merge streams that start one minute apart. An optimal scheduling algorithm for adaptive piggybacking was reported in [AWY96].

3.3 Variable bit-rate media

Scheduling schemes of the previous Sections assume a constant bandwidth to display each object. A class of compression techniques, termed variable bit-rate (VBR) encoding schemes, may result in bandwidth requirement that varies as a function of time. To illustrate, Figure 5(b) depicts the number of bytes that must be displayed as a function of time for a clip (Star Wars) in both uncompressed and VBR compressed format.

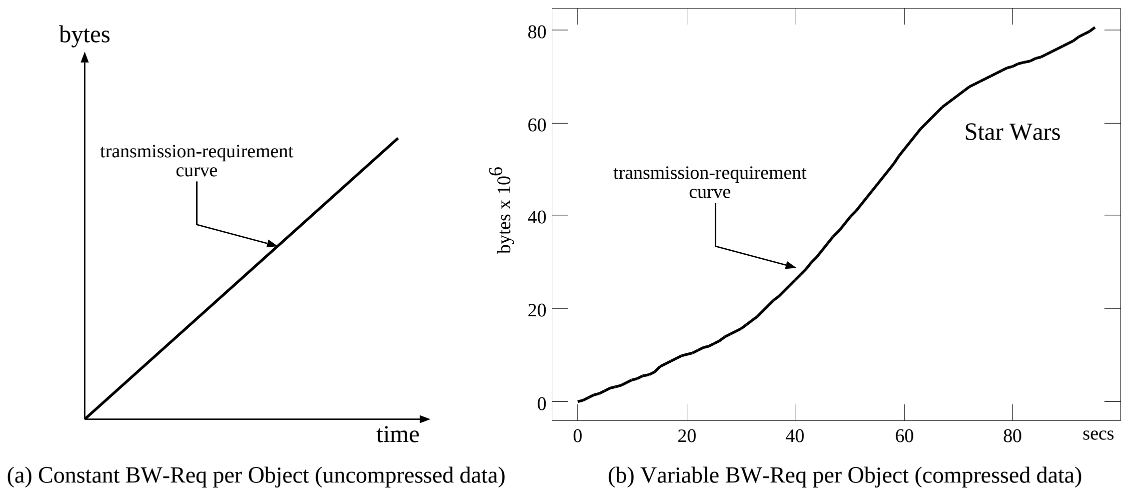


Figure 5: Object Bandwidth Requirements

With a VBR encoded clip, the disk transmission characteristic is not necessarily identical to the bandwidth requirement of an object as long as it is at a sufficiently high rate to stay ahead of the display bandwidth requirement, i.e., prevent hiccups. This is illustrated in Figure 6, where the disk transmission rate only "roughly" matches the object's delivery requirement. The data that has been transmitted by the disks but is not yet due at a display station, is buffered in memory. We

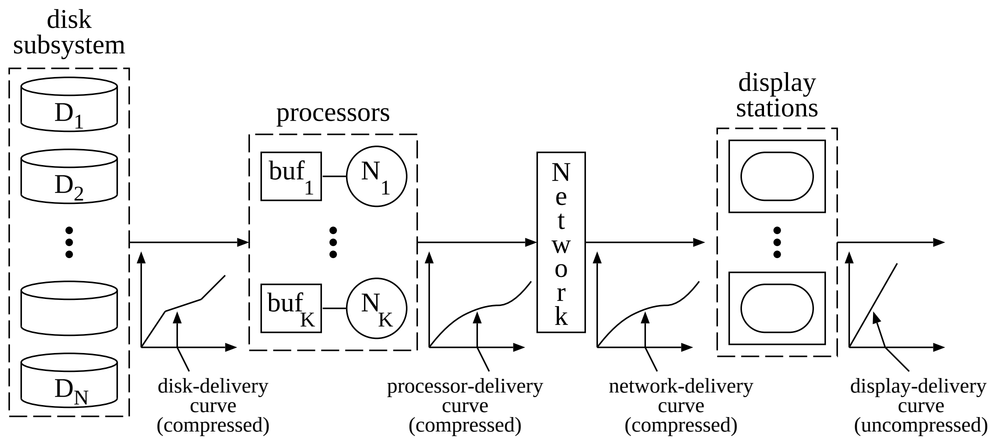

disk subsystem
D1
D2
DN
processors
buf1
N1
bufK
NK
Network
display stations
disk-delivery curve (compressed)
processor-delivery curve (compressed)
network-delivery curve (compressed)
display-delivery curve (uncompressed)

Figure 6: Transmission of Data

assume that the data is transmitted through the network in compressed form to reduce the network bandwidth requirements and is only decompressed at the display station. In addition, we assume that a display station performs the decompression, and has a limited amount of memory for buffering purposes, i.e., any extensive buffering required by the scheduling schemes must be provided by the storage manager. In this section, we focus on the data layout and the corresponding scheduling schemes for *disk* transmission, i.e., on the first step in Figure 6.

A taxonomy of disk scheduling to support delivery of VBR streams is shown in Figure 7 [EMGGM94].

The two basic categories of schemes that can be used with compressed data are:

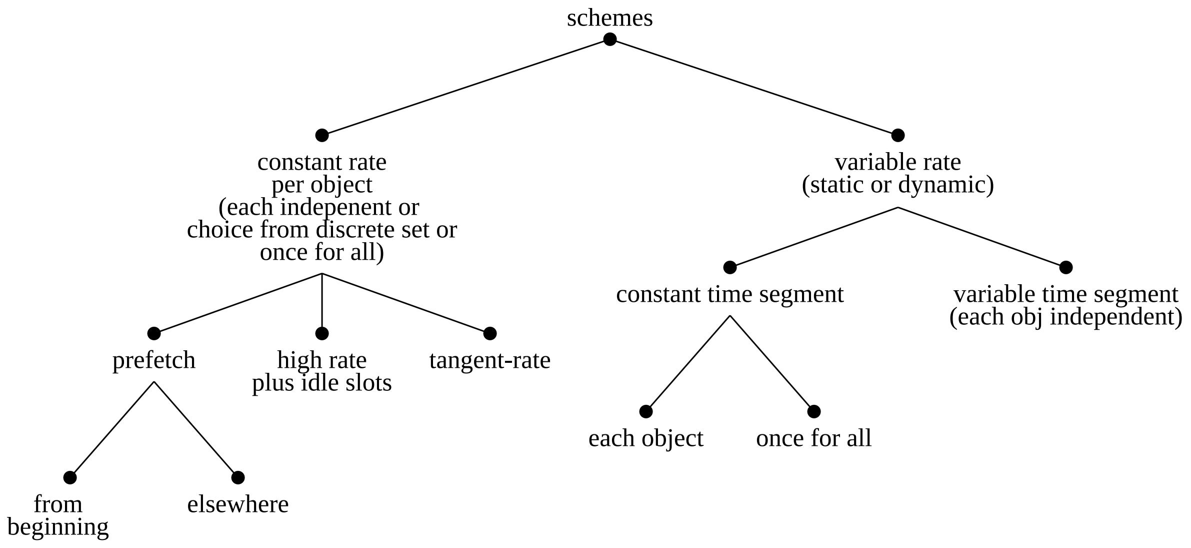


Figure 7: Taxonomy of Schemes

1. *constant-rate* schemes: Each object is delivered at a constant rate, although the rate may vary from one object to another, see Figure 8(a), and
2. *variable-rate* schemes: The amount of data retrieved from the disk on behalf of a display may vary from one time period to another, see Figure 8(b).

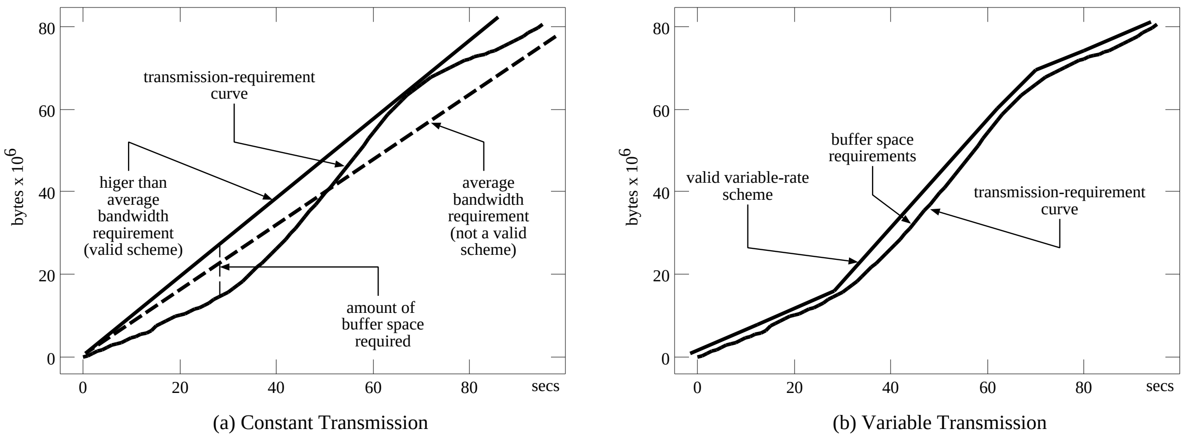


Figure 8: Disk Transmission

Each scheme discussed in [EMGGM94] differs in its construction of the "disk-delivery" curve, i.e., the curve that represents the rate at which the data will actually be transmitted from the disks. (This curve determines the bandwidth requirements, buffer space requirements, etc., discussed in the remainder of this section.) To insure the satisfaction of the real-time constraint, it is necessary for the "disk-delivery" curve to either coincide with or stay above the "transmission-requirement" curve. The difference between the required bandwidth and the bandwidth delivered by the disks defines the amount of required buffer space. The buffering requirements are minimized by closely fitting the two curves together.

An advantage of using a constant-rate scheme is that we can employ the scheduling techniques of previous sections without modifications. A limitation of this approach is that it results in high buffer space requirements when the highest transmission rate required by an object is not representative of the rest of the transmission-requirement curve. Large buffer space requirements by a display might limit the number of simultaneous displays supported by a server (even when sufficient disk bandwidth is available). This is due to a fixed amount of memory at the server.

A variable-rate scheme helps remedy the buffering problem because it provides more control on the buffer usage where a piecewise-linear curve is constructed to better fit the bandwidth requirements of the object, see Figure 8(b). One may construct the piecewise-linear curves in two alternative ways: variable time segment and constant time segment. With the first, the required disk transmission rate for an object varies with time. An example of this approach is the algorithm developed by [SZKT96]. This algorithm assumes the following two input parameters: the amount of memory available at a client, and the bit-rate requirements of a clip as a function of time. Using this input, the algorithm constructs the piecewise-linear curves optimally such that: (1) the peak rate of data retrieval is minimized, and (2) the variability in the rate of data retrieval for the duration of a display is minimized. While this is a very nice contribution, minimizing variability does not necessarily minimize the amount of memory required by a display. There might be other measures of variability that are more appropriate as detailed in the following paragrphs. A limitation of variable time segment approach is that it complicates the disk scheduling algorithms. In particular, when scheduling a new request, it is not sufficient to consider the current available bandwidth. Instead, the availability of the necessary bandwidth for the entire duration of an object's display must be taken into consideration in order to decide when to admit the new request. This can (potentially) increase the latency for starting the service of a request and result in disk bandwidth fragmentation problems, i.e., wasted bandwidth due to inability to use the full bandwidth capacity of the system as a result of future, rather than current, bandwidth requirements of an object.

The second, constant time segment (see Figure 7), constructs "rougher" piecewise-linear curves that are constant in time length. This might result in a higher amount of required memory per display. However, it would result in simpler disk scheduling techniques that would reduce the bandwidth fragmentation problems.

The following metrics should be used when comparing schemes:

- *buffer space*: amount of buffer space required by the scheme, which is a function of how closely the "disk-delivery" curve matches the "transmission-requirement" curve
- *bandwidth*: amount of additional bandwidth, i.e., in addition to the transmission requirements, that is used by the scheme (this is also a function of how closely the "disk-delivery" curve matches the "transmission-requirement" curve)
- *variability of bandwidth requirements*: the number of different bandwidths that an object requires during its display

Note that when comparing schemes, it is important to consider not just the maximum amount of buffer space that a particular schemes requires, but also the *duration* for which these buffers are required, e.g., some schemes might require a lot of buffer space, but only at the beginning of an object's display. A scheme that occupies buffers for a long time, in effect has relatively high buffer space requirements. Similarly, it is important to consider not just the amount of additional disk bandwidth required by a scheme, but also the duration for which it is required, e.g., some schemes might require more bandwidth but for a shorter duration of time. These factors impact both the throughput and the average startup latency of a server.

3.4 VCR functionality

In order to be provide an interface similar to the existing VCRs, a continuous media server might be required to support VCR functions such as Pause, Resume, Fast Forward and Fast Rewind with scan. Pause can be implemented by suspending the retrieval of the data from the disk, while Resume would restart the data retrieval. The design and implementation of scan functionalities are more challenging. One approach, as described in [DSST94], supports a fast-forward scan with five times (ff_{5X}) the normal display rate by allocates five times normal display bandwidth and includes quality-of-service (QoS) guarantees by introducing queuing models. This approach provides almost immediate access to full-resolution fast-forward and fast-rewind bandwidth with a low system load. With a high system load when bandwidth is not available, service is either delayed or provided immediately but with a lower resolution, i.e., QoS. While this approach is conceptually simple, it might require a complex decoder for selective display rates.

An alternative approach, as described in [ORS96], is a matrix-based allocation (phase-constrained allocation). This scheme skips a fixed number of blocks to achieve the desired scan rate by controlling the placement of blocks in a disk drive. Similarly, [CKY94] proposed a storage method, placement and sampling methods for MPEG-like video streams (full VCR functions) without increasing resources. For example, to support a ff_{5X} display, this method retrieves and displays one block out of every five consecutive blocks. The advantage of this approach is that it requires no extra resource to support various scan rates. However, both approaches result in an unusual scan viewing because they skip blocks (segment-skipping) instead of frames.

Another alternative is to support separate fast-forward and fast-rewind files by selecting and preprocessing frames before compression [BGMJ94]. For example, to support a ff_{5X} scan of a movie, every fifth frame of the movie is selected prior to its compression. The new file, termed a *trick* file, is encoded in the normal fashion and stored in the system. The system maintains an index that maps the frames of the original movie to that of its trick file. When the user requests a fast-forward with scan, the system uses the index to locate the corresponding frame in the trick file. It suspends the current retrieval and initiates the display starting with the computed frame of the trick file. This method requires extra storage for the trick files and their corresponding index structures. This overhead is marginal and typically less than a few percentage of the movie size.

4 Striping and Continuous Display

Assuming a fixed block size and a system with D disks, declustering a block across all D disks might waste a large fraction of the available disk bandwidth if the fragment size on each disk is small. An alternative, termed *simple striping*, is to partition the D disks into M clusters. Each cluster will consist of d disks; $M = \frac{D}{d}$. For now, assume that D is a multiple of d. (This assumption is relaxed in Section 4.1.) The blocks of an object X are assigned to the disk clusters in a round-robin manner, starting with an arbitrarily chosen disk cluster. For example, in Figure 9, a system consisting of six disks (D=6) is partitioned into three clusters (M=3), each consisting of two disk drives. The

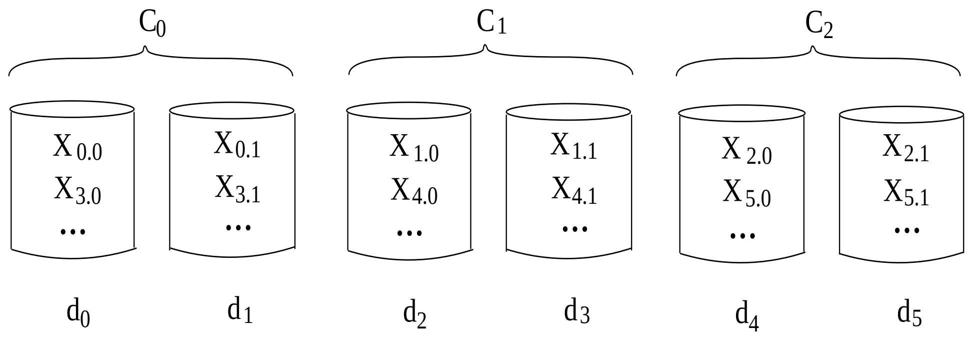
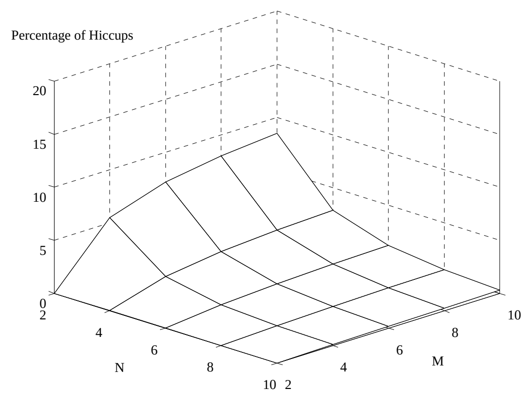
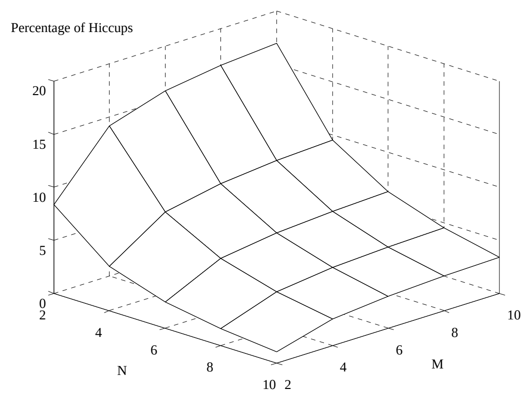


Figure 9: Simple striping



a. 50% system load

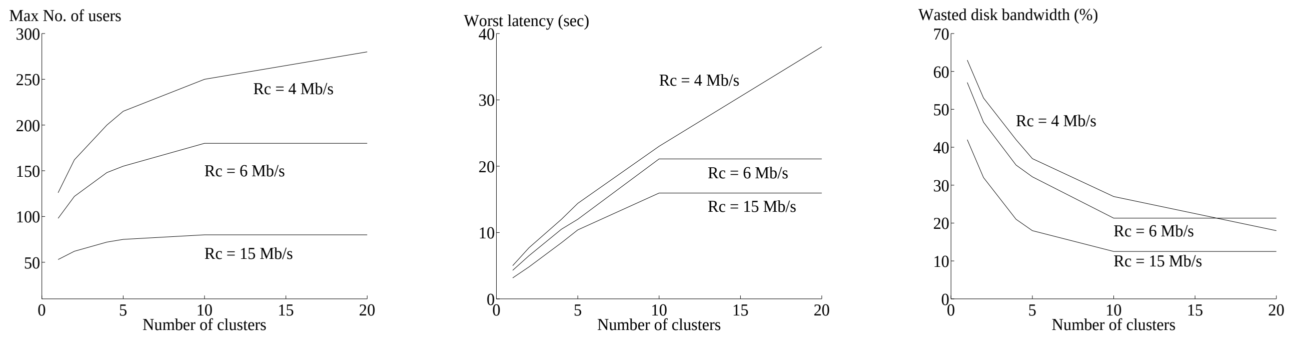


b. 75% system load

Figure 10: Percentage of hiccups with a random data placement

assignment of the blocks of X starts with cluster 0. This block is declustered into two fragments: $X_{0.0}$ and $X_{0.1}$. The retrieval and display of an object is almost identical to the discussion of Section 3. The main difference is as follows. Once a display locates an idle slot on the cluster that contains X_0 (C_i), it retrieves the next block from cluster $C_{(i+1)\ mod\ M}$ during the next time period. This process is repeated until all blocks of X have been retrieved and displayed.

The blocks are dispersed across the disks clusters based in a deterministic pattern (e.g., round-robin) in order to yield a regular schedule of block retrieval. With a regular schedule, once the system locates an idle slot (from the $\mathcal{N}$ slots on the cluster containing X_0) on behalf of a display referencing X, it prevents the possibility of more than $\mathcal{N}$ active requests referencing a single cluster during future time periods. Alternatively, with a random placement of data, in the worst case scenario, $M \times \mathcal{N}$ active displays might reference a single cluster during one time period. In this case, the retrieval of $(M-1)\times\mathcal{N}$ blocks are delayed, resulting in $(M-1)\times\mathcal{N}$ displays to observe hiccups. Using analytical models, one can compute the percentage of displays that would observe hiccups as a function of input parameters: M, $\mathcal{N}$, and a system load (i.e., the number of active requests defined as a percentage of $M \times \mathcal{N}$). The analytical models enumerate the possible ways of distributing the active requests across the clusters. Each distribution is termed a system state. Each state i has a fixed probability of occurrence (p_i,$0 < p_i$) and results in a fixed number of hiccups (h_i, $0 \leq h_i \leq (M-1) \times \mathcal{N}$). Given s possible system states, the percentage of hiccups is: $\sum_{i=1}^{i=s} p_i \times h_i$. Figure 10.a shows the percentage of hiccups as a function of M and $\mathcal{N}$ for a 50% system load. With two disk clusters, each with the



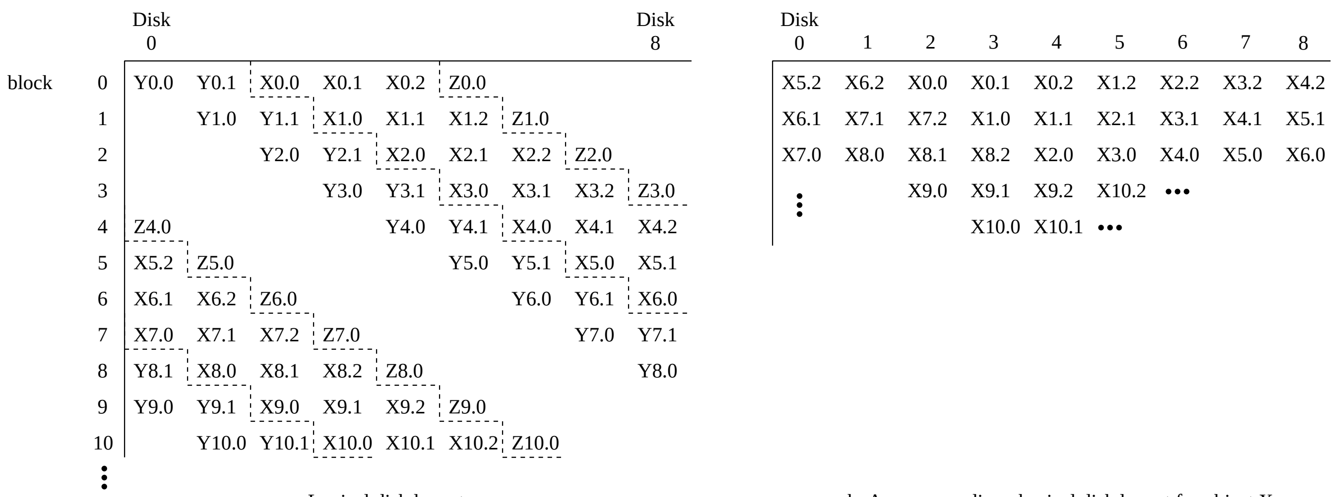
a. Throughput b. Maximum startup latency c. Wasted disk bandwidth

Figure 11: Impact of cluster size

capacity to retrieve $\mathcal{N}$ blocks per time period, the 50% system load yields $\mathcal{N}$ active requests. In this case, the percentage of hiccups is zero. However, as the number of disk clusters increases (M), the percentage of hiccups increases proportionally. With $\mathcal{N} = 2$ and M=10, the percentage of hiccups is 9%. This demonstrates that a random data placement yields a non-scalable system because as M increases, the imposed system load should be reduced to maintain a fixed hiccup percentage.

A round-robin data placement and schedule of block retrieval distributes the work imposed by a request evenly across the available resources, resulting in a scalable server where the number of simultaneous displays supported by the system increases as a linear function of M (as long as sufficient amount of memory is provided). This linear increase is independent of the distribution of access to the different clips. A limitation, however, is a linear increase in the maximum startup latency as a function of M. This is because in the worst case scenario, a single slot might be available on the cluster containing X_0 and a new request referencing X might arrive too late to utilize this slot. In this case, the request must wait for M time periods before the idle slot visits the cluster containing X_0. The percentage of wasted disk bandwidth is a function of d; see Equations 3 4 and recall that these equations assume that a logical disk consists of d disks. To illustrate the tradeoff associated with different values of d, assume a system that consists of 20 disks ($D = 20$) with 320 megabytes of memory. Figure 11 shows the maximum throughput and startup latency of the system as a function of the number of clusters (M) for three different media types requiring 4, 6, and 15 Mbps. In this figure, the x-axis corresponds to M which is a function of d. While with M=20, each cluster consists of one disk (d=1), with M=10 each cluster consists of two disks (d=2). The throughput of the system increases as a function of M because the percentage of wasted disk bandwidth decreases, see Figure 11.c. With R_C=6 and 10 Mbps, the throughput remains unchanged when M changes from 10 to 20 because we assumed a fixed amount of memory.

When the database consists of a mix of media types each with a different bandwidth requirement, the design of simple striping can be extended to minimize the percentage of wasted disk bandwidth. This is specially true for the high bandwidth media types whose display bandwidth requirement exceeds that of a single disk drive. To illustrate, assume a database that consists of two media types, A and B. The bandwidth requirement of media type A is less than one disk while that of media type B requires four disks. With simple striping, one might organize the D disks into cluster sizes of four disks, i.e., d=4. If the system workload is dominated by requests that reference objects of media type A then a large percentage of disk bandwidth is wasted. Staggered striping is a superior alternative as it minimizes the percentage of wasted disk bandwidth.



block	Disk 0	1	2	3	4	5	6	7	Disk 8
0	Y0.0	Y0.1	X0.0	X0.1	X0.2	Z0.0			
1		Y1.0	Y1.1	X1.0	X1.1	X1.2	Z1.0		
2			Y2.0	Y2.1	X2.0	X2.1	X2.2	Z2.0	
3				Y3.0	Y3.1	X3.0	X3.1	X3.2	Z3.0
4	Z4.0				Y4.0	Y4.1	X4.0	X4.1	X4.2
5	X5.2	Z5.0				Y5.0	Y5.1	X5.0	X5.1
6	X6.1	X6.2	Z6.0				Y6.0	Y6.1	X6.0
7	X7.0	X7.1	X7.2	Z7.0				Y7.0	Y7.1
8	Y8.1	X8.0	X8.1	X8.2	Z8.0				Y8.0
9	Y9.0	Y9.1	X9.0	X9.1	X9.2	Z9.0			
10		Y10.0	Y10.1	X10.0	X10.1	X10.2	Z10.0		
⋮									

a. Logical disk layout

Disk 0	1	2	3	4	5	6	7	8
X5.2	X6.2	X0.0	X0.1	X0.2	X1.2	X2.2	X3.2	X4.2
X6.1	X7.1	X7.2	X1.0	X1.1	X2.1	X3.1	X4.1	X5.1
X7.0	X8.0	X8.1	X8.2	X2.0	X3.0	X4.0	X5.0	X6.0
⋮		X9.0	X9.1	X9.2	X10.2	•••		
			X10.0	X10.1	•••			

b. A corresponding physical disk layout for object X

Figure 12: Staggered striping with 8 disks

4.1 Staggered Striping

Staggered striping is a generalization of simple striping. It constructs the disk clusters by allowing two consecutive blocks of X (say X_i and X_{i+1}) to reside on an overlapping set of disks. The blocks are assigned such that the disk containing the first fragment of X_{i+1} (i.e., $X_{i+1.0}$) is k disks (modulo the total number of disks) apart from the disk drive that contains the first fragment of X_i (i.e., $X_{i.0}$). The distance between $X_{i.0}$ and $X_{i+1.0}$ is termed *stride*. If a database consists of a single media type (with a degree of declustering d_X) and D is a multiple of d_X then staggered striping can implement simple striping by setting the stride to equal to the degree of declustering of an object ($k = d_X$). The system maintains a time period for each disk drive. Assuming an idle system, when a request referencing object X arrives, the system locates d_X idles slots on d_X adjacent disks starting with the disks that contains $X_{0.0}$, disk 2 in Figure 12.

Figure 12 illustrates both the logical and physical assignment of the blocks of X with staggered striping and stride k=1 (for the moment we ignore objects Y and Z). As compared with simple striping, the display of X with staggered striping differs in one way: after each time period, the disks employed by a request shift k to the right (instead of d_X with simple striping). When the database consists of a mix of media types, the objects of each media type are assigned to the disk drives independently but all with the same stride. Figure 12(a) demonstrates the assignment of objects X, Y, and Z that belong to three different media types. In order to maximize the throughput of the system, each media type requires a cluster size of 3, 2, and 1 disks, respectively (i.e., d_X=3, d_Y=2, d_Z=1). The stride for the entire database is 1. In order to display object X, the system locates the d_X logically adjacent disk drives that contain its first block (disks 2, 3, and 4). If an idle slot is available on each of these disk drives, they are employed during the first time period to retrieve X_0. During the second time period, the next d_X disk drives are employed by shifting k disks to the right. When compared with simple striping, the throughput of the system is maximized because the percentage of wasted disk bandwidth attributed to the display of each object is minimized.

With staggered striping, the degree of declustering of objects is dependent on their media type. However, assuming a disk with a single zone, the size of a fragment (i.e., unit of transfer from each disk drive) is the same for all objects, regardless of their media type[3]. Thus, the duration of a time period is constant for all objects. The choice of a value for the stride (k) must be determined at system configuration time. It may vary in value from 1 to D since a value (say i) greater than D is equivalent to i modulo D. The choice of k and D is important as particular combinations of values for k and D can result in very skewed load on the disks, both storage and bandwidth. For example, if $k = D$ then all blocks of X are assigned to the same disk drive. Hence, the number of unique disks employed to display X is d_X, each for the entire display time of X. If X is a popular object and these d_X disks can support 10 such displays then the entire system can support only 10 displays of this object even though its true potential might be thousands of displays. In this case, one might replicate object X on different disks in order to support a higher number of displays for this object [GS93, YCK93]. Simulation results presented in [BGM95] demonstrate that a different value for k is more beneficial when compared with replication. A stride of 1 guarantees no data skew. Similarly, a choice of value for D and k that are relatively prime guarantees no data skew.

Staggered striping may cause a fraction of the disk drives to remain idle even though there are requests waiting to be serviced. This occurs when the idle disk drives are not adjacent due to the display of other objects. This limitation is termed *bandwidth fragmentation*. To illustrate, consider the assignment of X, Y, and Z in Figure 12. Assume that an additional object (say W) with a degree of declustering of 4 is disk resident (d_W=4). Suppose the placement of W_0 starts with disk 2 ($W_{0.0}$ is stored on the same disk drive containing $X_{0.0}$). If the system is busy servicing three requests referencing objects X, Y, and Z, then there are three disks that are idle. Assume that a new request arrives referencing object W. It would have to wait because the number of idle disks (3) is less that that required to display W (4). If the display of object X now completes then there are six idle disks. However, the system is still unable to display object W because the available disk drives are not adjacent to each other (they are in groups of three separated by the display of Y and Z). The system cannot service displays requiring more than three disks until the display of either Y or Z completes.

Bandwidth fragmentation can be alleviated by careful scheduling of jobs, but cannot be completely eliminated. However, with additional memory for buffer space and additional network capacity, the bandwidth fragmentation problem can be solved. To accomplish this, assume that a fragment can be read from the disk into a buffer in one time period and, in a subsequent time period, the same processor node can concurrently transmit to the network both (a) the previously buffered fragment, and (b) a disk resident fragment (using the pipelining scheme outlined earlier). In this section, we show how to use buffers to utilize a set of disks that are not adjacent to deliver an object.

Figure 13 illustrates an example of how our approach works. In the figure, the white regions indicate which disks were available for serving new requests while the shaded regions are disks busy serving other requests. A request arrives at time 0 for an object X with a degree of declustering equal to 2. Further the stride is 1 in this example and the initial block X_0 is stored on disks 0 and 1. There are 2 free disks, but they are not consecutive; there are 2 intervening busy disks. Disk 1 is free and is in position to read fragment $X_{0.1}$, but disk 6 which is also free will not be in position to read fragment $X_{0.0}$ until time period 2. In order to support bandwidth fragmented delivery of object X, disk 1 can keep fragment $X_{0.1}$ in memory until time 2 when it can be delivered along with fragment $X_{0.0}$. Thus at time 2, fragment $X_{0.0}$ is pipelined directly from disk 0 to the network, while node 1 transmits fragment $X_{0.1}$ from its buffers (while disk 1 is concurrently servicing another request). Similarly disk 2 reads fragment $X_{1.1}$ at time 1, and buffers it until time period 3 when both $X_{1.0}$ and $X_{1.1}$ can be delivered.

[3]With a multi-zone disk, the size of a fragment may vary depending on the employed data placement and scheduling techniques.

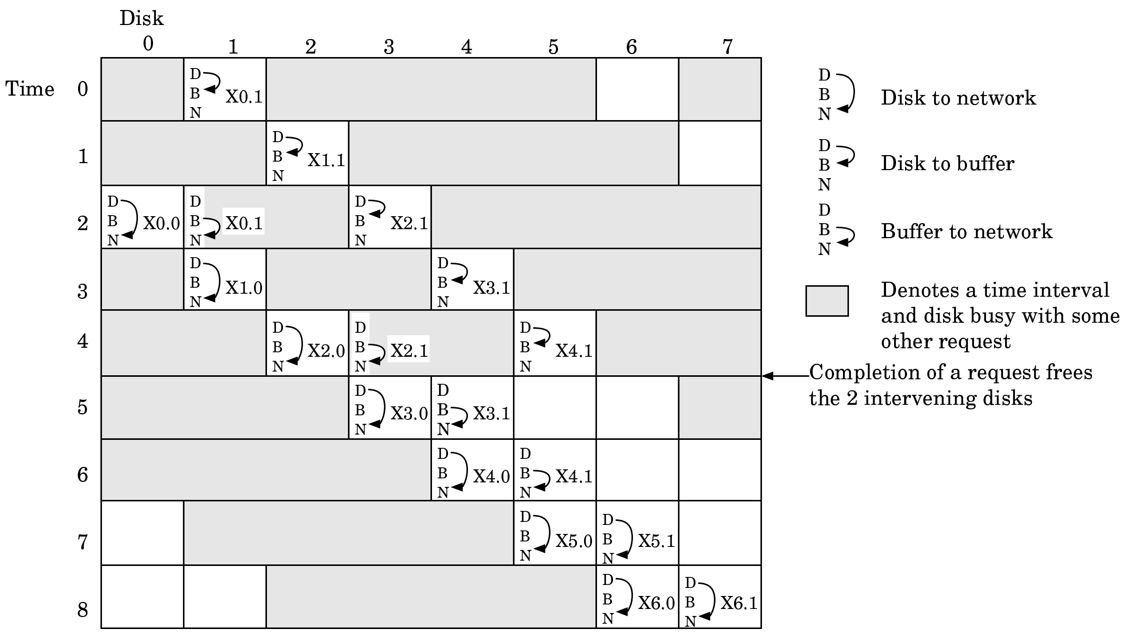


Figure 13: Utilizing fragmented disks

It is interesting that it is also possible to dynamically *coalesce* the use of the disks for the fragmented request as intervening busy disks become available. Figure 13 illustrates how fragmented requests can be dynamically coalesced. Suppose at time period 5, the 2 intervening disks have completed their service and become free. At that point, the bandwidth fragmented request can be completely coalesced so that the disks supporting the transmission of object X are adjacent (depending on how many disks become free, a bandwidth fragmented request may be only partially coalesced). By the start of time period 5, fragments $X_{3.1}$ and $X_{4.1}$ are already buffered, and have to be delivered before reading recommences. During time periods 5 and 6, fragments $X_{3.1}$ and $X_{4.1}$ are delivered from buffers while fragments $X_{3.0}$ and $X_{4.0}$ are delivered directly from disk. Starting at time 7, the coalescing has been completed and the 2 consecutive disks pipeline the fragments directly from the disk to the network.

5 Experimental Results From a Prototype

Mitra [GZS$^+$97] is a realization of the design concepts described in this paper. It is a software based system that can be ported to alternative hardware platforms. It guarantees simultaneous display of a collection of different media types as long as the bandwidth required by the display of each media type is constant (isochronous). The current hardware platform of Mitra is a cluster of HP 9000/735 workstations. An HP Magneto Optical Juke-box serves as the tertiary storage device for the system. Each workstation consists of consists of a 125 MHz PA-RISC CPU, 80 MByte of memory, and four Seagate ST31200W magnetic disks. While 15 disks can be attached to the fast and wide SCSI-2 bus of each workstation, we attached four disks to this chain because additional disks would exhaust the bandwidth of this bus. It is undesirable to exhaust the bandwidth of the SCSI-2 bus for several reasons. First, it would cause the underlying hardware platform to not scale as a function of additional disks. Second, it renders the service time of each disk less predictable, increasing the memory requirement to avoid hiccups.

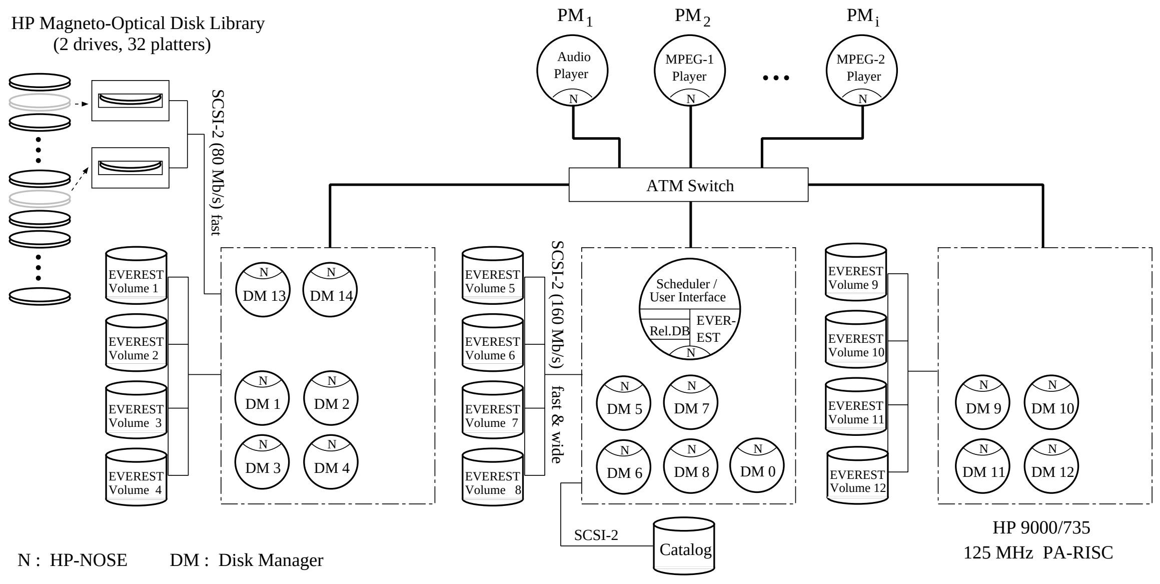


Figure 14: Hardware and software organization of Mitra

Mitra consists of three software components:

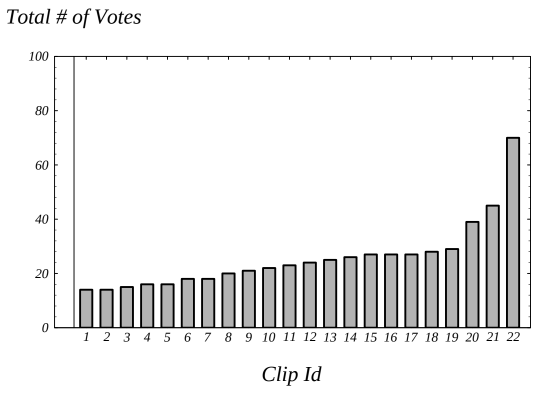
1. Scheduler: this component schedules the retrieval of the blocks of a referenced object in support of a hiccup-free display at a Presentation Manager (PM). In addition, it manages the disk bandwidth and performs admission control. Currently, the Scheduler includes an implementation of the EVEREST file system [GIZ96], staggered striping, and techniques to manage the tertiary storage device [GDS95]. It also has a simple relational storage manager to insert, and retrieve information from a *catalog*. For each media type, the catalog contains the bandwidth requirement of that media type and its block size. For each presentation, the catalog contains its name, whether it is disk resident (if so, the name of EVEREST files that represent this clip), the cluster and zone that contains its first block, and its media type.

2. mass storage Device Manager (DM): Performs either disk or tertiary read/write operations.

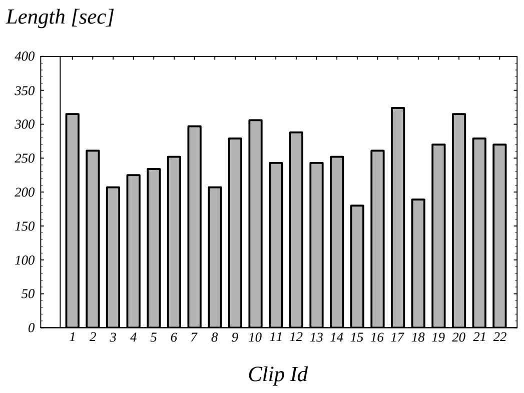
3. Presentation Manager (PM): Displays either a video or an audio clip. It might interface with hardware components to minimize the CPU requirement of a display. For example, to display an MPEG-1 clip, the PM might employ either a program or a hardware-card to decode and display the clip.

For a given configuration, the following processes are active: one Scheduler process, a DM process per mass storage read/write device, and one PM process per active client. For example, in our twelve disk configuration with a magneto optical juke box, there are sixteen active processes: fifteen DM processes, and one Scheduler process (see Figure 14). There are two active DM processes for the magneto juke-box because it consists of two read/write devices (and 32 optical platters that might be swapped in and out of these two devices).

The combination of the Scheduler with DM processes implements asynchronous read/write operations on a mass storage device (that is otherwise unavailable with HP-UX 9.07). This is achieved as follows. When the Scheduler intends to read a block from a device (say a disk), it sends a message to the DM that manages this disk to read the block. Moreover, it requests the DM to transmit its



(a) Number of votes for each clips



(b) Length [in seconds] of each clip

Figure 15: Characteristics of the CD audio clips

block to a destination port address (e.g., the destination might correspond to the PM process that displays this block) and issue a done message to the Scheduler. There are several reasons for not routing data blocks to active PMs using the Scheduler. First, it would waste the network bandwidth with multiple transmissions of a block. Second, it would cause the CPU of the workstation that supports the Scheduler process to become a bottleneck with a large number of disks. This is because a transmitted data block would be copied many times by different layers of software that implement the Scheduler process.

5.1 Experimental Design

Mitra is a rich platform to demonstrate and verify the tradeoff associated with the alternative design decisions described in this paper. We describe a subset of these by: 1) ignoring the role of tertiary and focusing on the performance of the system when all the referenced objects are disk resident, and 2) focusing on a single media type. The second design decision is a simplifying one because it eliminates a dimension that raises questions such as: What is a typical mix of media type? What are the bandwidth requirement of each media type? What is the size distribution of objects for a given media type? etc.

For this evaluation, we used a database based on a WWW page that ranks the top fifty songs every week[4]. We chose the top 22 songs of January 1995 to construct both the benchmark database and its workload. (We could not use all fifty because the total size of the top 22 audio clips exhausted the storage capacity of one disk Mitra configuration.) Figure 15.a and b shows the frequency of access to the clips and the size of each clip in seconds, respectively. The size of the database was fixed for all experiments.

We employed a closed evaluation model with zero think time for our evaluation. With this model, a workload generator process is aware of the number of simultaneous displays supported by a configuration of Mitra (say $\mathcal{N}$). It dispatches $\mathcal{N}$ requests for object displays to Mitra. (Two or more requests may reference the same object, see below.) As soon as Mitra is done with the display of a request, the workload generator issues another request to the Scheduler (zero think time). In

[4]This web site is maintained by Daniel Tobias (http://www.softdisk.com/comp/hits/). The ranking of the clips is determined through voting by the Internet community, via E-mail.

Seagate ST31200W	
Capacity	1.0 gigabyte
Revolutions per minute	5400
Maximum seek time	21.2 millisecond
Maximum rotational latency	11.1 millisecond
Number of zones	23 (see Figure 1)
Database Characteristics	
CD Quality Audio	
Sampling rate	44,100 per second
Resolution	16 bits
Channels	2 (stereo)
Bandwidth requirement	1.3458 Mbps

Table 6: Fixed Parameters

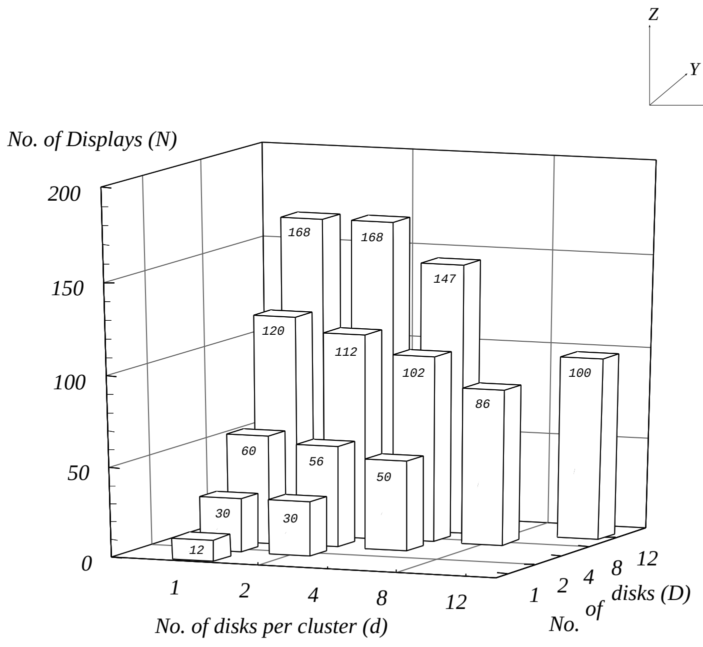
effect, this is a heavy load model that maintains the maximum number of streams in the system. The distribution of request references to clips is based on Figure 15.a. This is as follows. We normalized the number of votes to the 22 clips as a function of the total number of vote for these objects. The workload generator employs this distribution to construct a queue of requests that reference the 22 clips. This queue of requests is randomized to result in a non-deterministic reference pattern. However, it might be the case that two or more requests reference the same clip (e.g., the popular clip) at the same time. Mitra was NOT configured to multiplex a single stream to service these requests.

For the purposes of the workload generator, the initial value of $\mathcal{N}$ is determined using the analytical models developed in Section 3. In practice, Mitra cannot support these analytical expectations, resulting in many hiccups. Using a trail and error method, we reduce the value of $\mathcal{N}$ until there were no hiccups. Section 5.2 reports on the difference between the analytical expectations of $\mathcal{N}$ and experimental value of $\mathcal{N}$. Moreover, it provides our intuition behind this difference.

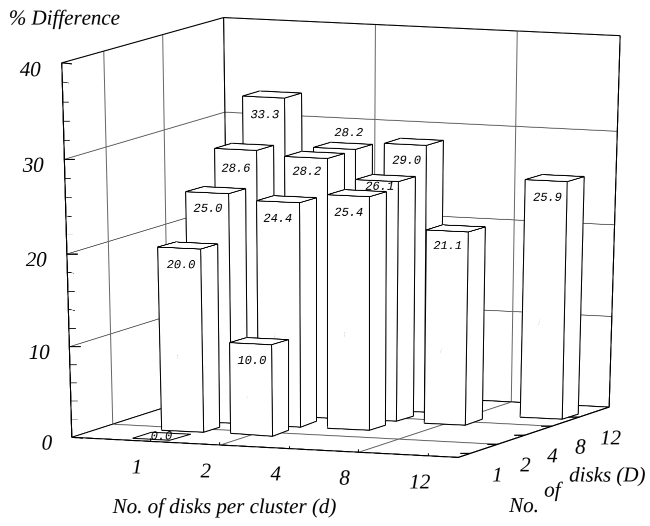
This experimental design consists of three states: warmup, steady state, and shutdown. During the system warmup (shutdown), Mitra starts to become fully utilized (idle). In our experiments, we focused on the performance of Mitra during steady state by ignoring the statistics collected during both system warmup and shutdown.

5.2 Experimental Results

In these experiments, the following system parameters are fixed: block size is 384 KByte, GSS is configured with a single group (g=1), and a single logical zone spans all 23 physical zones of each disk. We analyzed the performance of Mitra as a function of additional disks by varying D from 1 to 2, 4, 8, and 12. For each configuration, we analyzed the performance of Mitra as a function of the number of disks that constitute a cluster (i.e., d). In all experiments, the stride (k) equals to d, yielding a data placement based on simple striping. Figure 16.a presents the number of simultaneous displays supported by Mitra as a function of D and d. In this figure, the number of disks available to Mitra is varied on the y-axis, the number of disks that constitute a cluster is varied on the x-axis, and the throughput of the system is reported on the z-axis.



(a) Number of simultaneous displays



(b) Percentage difference between theoretical expectations and obtained results from Mitra

Figure 16: Performance of Mitra as a function of D and d (k=d)

As the number of disks in the system (D) increases from 1 to 12 with d=k=1, the throughput of the system increases super linearly (the throughput of Mitra with D=12 is fourteen times higher than that with D=1). This is because the average transfer rate of each disk increases as a function of D. The explanation for this is as follows. In this experiment, the size of the database is fixed and the EVEREST file system organizes files on a disk starting with the outermost zone, i.e., fastest zone. The amount of data assigned to each disk shrinks as D increases. With D=1, the innermost zone of the disk contains data, while with D=12, only the three outermost zones contain data. The average transfer rate of the three outermost zones is higher than the average transfer rate of all 23 zones of a disk (see Figure 1).

In Figure 16.a, for a given hardware platform (fixed D), the throughput of Mitra drops as d increases. For example with D=12, Mitra's throughput drops from 168 streams to 100 as d increases from 1 to 12. This is because the percentage of wasted disk bandwidth increases as d increases in value, see the discussions of Section 4. A limitation associated with values of d smaller than D is the constrained placement of data. This results in a higher average startup latency (see Figure 17), see the discussions of Section 4.

For each choice of value for D, we located the slowest participating zone of the disks that contains data. This zone is the same for all disks due to the round-robin assignment of blocks of each object to disks. We computed expected performance of Mitra as a function of this zone's transfer rate for each configuration (using the analytical models of Section 3). Next, we examined how closely Mitra approximates these theoretical expectations. Figure 16.b presents the percentage difference between the measured results and theoretical expectations. Each value of this figure is computed based on: $1 - \frac{Measured}{Theory}$. With $D = 1$, the system approximates the theoretical expectation with 100% accuracy. With $D > 1$, Mitra's performance is anywhere from 10% to 35% lower than its theoretical expectations. These results raise two questions: 1) When compared with $D = 1$, why does the system

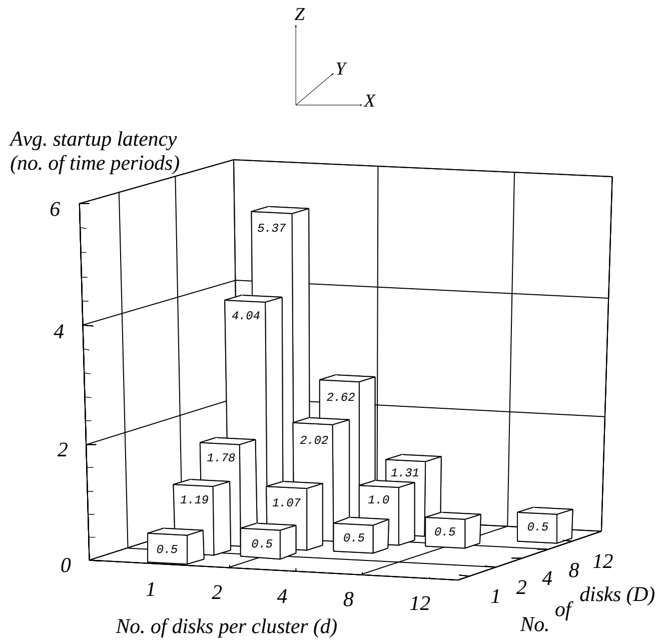


Figure 17: Average Startup latency of a PM with Mitra

demonstrate a higher percentage difference with $D > 1$? and 2) What implementation overheads result in a performance lower than the theoretical expectations? Consider each question in turn. The answer to the first question is that the overhead of implementation becomes more profound with $D > 1$. To explain this, note that Mitra is observing the average transfer rate of the zones (even though configured to assume the transfer rate of slowest zone) because the data is dispersed across all the participating zones. This average transfer rate is higher than the transfer rate of the slowest participating zone assumed by the analytical models. As one increases D, the difference between these two numbers becomes insignificant. This explains why the overhead of implementation becomes more profound with $D > 1$.

The answer to the second question is more difficult. We can confidently state that part of the overhead is attributed to loss of network packets and their retransmission. With our ATM switch, we transmitted a 384 KByte disk block as a sequence of 8 KByte network packets. With larger packet sizes, the frequency of packet loss was very high, diminishing the overall system throughput. Moreover, we believe there are other factors (e.g., SCSI-2 bus, software overhead, system bus arbitration, HP-UX scheduling of processes, etc.,) that might contribute to the overhead. These delays are expected with a software based system (based on previous experience with Gamma [DGS$^+$90] and Omega [GCKL92]) because the system does not have complete control of the underlying hardware.

The implementation of Mitra has provided us with the following lessons:

1. The performance trends expected by the analytical models for the disk subsystem (as a function of D and d) are realistic; compare the discussions of Figure 11(a) with those of Figure 16(a).
2. The performance of Mitra is lower than the theoretical expectations because it is a software solution that does not have complete control of the hardware. Moreover, the focus of the system has been on the disk subsystem. Our current networking solutions require further studies. Mitra currently employs UDP/IP protocol with ATM. Both are not necessarily the

best implementation decisions. In order to enhance the overall system performance, we should consider other networking protocols (e.g., RSVP [ZDE+93]). In addition, our use of the ATM switch requires further evaluation. ATM was employed because it was available to us at no cost.

3. A centralized scheduler is necessary to implement: (a) round-robin activation of disk clusters on behalf of a display, and (b) admission control policies. With a large system consisting of thousands of disks, this scheduler might become a bottleneck.

6 Conclusion and Future Research Directions

In this study, we have focused on the disk subsystem of a continuous media server and described disk scheduling and data placement techniques in support of high performance, scalability, and continuous display. For the proposed techniques, we have described analytically models to quantify the tradeoff between memory and disk bandwidth, disk bandwidth and disk storage, throughput and startup latency. These trends have been validated using Mitra.

There are several research directions that are currently in progress. We list two of these activities in this section. First, while a single disk is quite reliable, as one increases the number of disks in a scalable server, the probability of *some* disk failure increases. For example, with a mean time to failure of once very 500,000 hours (57 years) for a disk, a system that consists of one thousand disks would observe a disk failure once every 500 hours (20 days). When such a failure occurs, the system might be required to terminate some streams since there is less system capacity available. While it is possible for admission control to hold back resources against the possibility of component failure, this will imply a smaller maximum number of concurrent streams and possibly larger waiting times for initiation of streams [BGM95]. The system designer has to decide whether using all or nearly all of the system capacity during normal operation, with the possibility of having to drop streams in the case of failure, is better than denying or delaying the initiation of service. Clearly there is a range of possibilities here determined by the amount of capacity held back against failures and the number of failures that can be sustained before dropping streams. Another method of compensating for reduced capacity in these systems is to maintain all streams but with some or all at a reduced bit rate. This assumes that a reduced resolution versions of objects are available.

Second, the news broadcasters have produced a strong demand for digital non-linear editing systems. A specific application of these systems to sports was detailed in Section 1. In addition to the synchronization and retrieval issues [CGS95], these systems must be able to: 1) accept satellite input from remote sources to populate the server with new video clips, and 2) be able to broadcast data [AKNG97]. With a satellite input, a clip might be written to the server at four times the regular display rates. In addition, the server must manage the multiplexing of disk bandwidth between: 1) the different editors accessing the server, and 2) the broadcasted stream that has the highest priority. It must also manage the disk space to prevent its fragmentation and ensure availability of enough space (by writing old data to disk). These requirements require further extensions to the solutions described in this paper.

References

[AKNG97] W. Aref, I. Kamel, S. Niranjan, and S. Ghandeharizadeh. Disk scheduling for Displaying and Recording Video in Non-Linear News Editing Systems. *Multimedia Computing and Networking*, February 1997.

[And93] D. P. Anderson. Metascheduling for Continuous Media. *ACM Transactions on Computer Systems*, 11(3):226–252, August 1993.

[AWY96] C. Aggarawal, J. Wolf, and P. S. Yu. On Optimal Piggyback Merging Policies for Video-on-Demand Systems. In *Proceedings of the 1996 ACM Sigmetrics Conference*, pages 200–209, 1996.

[BG88] D. Bitton and J. Gray. Disk shadowing. In *Proceedings of the International Conference on Very Large Databases*, September 1988.

[BGM95] S. Berson, L. Golubchik, and R. Muntz. Fault Tolerant Design in Multimedia Servers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1995.

[BGMJ94] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 79–89, 1994.

[Bir95] Y. Birk. Track-Pairing: A Novel Data Layout for VOD Servers with Multi-Zone-Recording Disks. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 248–255, May 1995.

[CGS95] S. Chaudhuri, S. Ghandeharizadeh, and C. Shahabi. Retrieval of Composite Multimedia Objects. In *Proceedings of the International Conference on Very Large Databases*, September 1995.

[CKY94] M. Chen, D.D. Kandlur, and P.S. Yu. Support for Fully Interactive Playout in a Disk-Array-Based Video Server. In *Proceedings of the ACM Multimedia*, 1994.

[DD$^+$95] A. Dan, , D. Dias, R. Mukherjee, D. Sitaram, and R Tewari. Buffering and Caching in Large-Scale Video Servers. In *Proc. of COMPCON*, 1995.

[DGS$^+$90] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 1(2), March 1990.

[Doz92] J. Dozier. Access to data in NASA's Earth observing system (Keynote Address). In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 1992.

[DSS94] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. In *Proc. of the 2nd ACM International Conference on Multimedia*, October 1994.

[DSST94] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley. Channel Allocation under Batching and VCR Control in Mov ie-On-Demand Servers. Technical Report RC19588, IBM Research Report, Yorktown Height, NY, 1994.

[EMGGM94] M. L. Escobar-Molano, S. Ghandeharizadeh, L. Golubchik, and R. R. Muntz. A framework for the study of delivery of compressed data in multimedia information systems. Technical report, UCLA, 1994.

[GCKL92] S. Ghandeharizadeh, V. Choi, C. Ker, and K. Lin. Omega: A Parallel Object-based System. In *Proceedings of the International Conference on Parallel and Distributed Information Systems*, Dec 1992.

[GDS95] S. Ghandeharizadeh, A. Dashti, and C. Shahabi. A Pipelining Mechanism to Minimize the Latency Time in Hierarchical Multimedia Storage Managers. *Computer Communications*, March 1995.

[GHW90] J. Gray, B. Host, and M. Walker. Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput. In *Proceedings of the International Conference on Very Large Databases*, August 1990.

[GIZ96] S. Ghandeharizadeh, D. Ierardi, and R. Zimmermann. An Algorithm for Disk Space Management to Minimize Seeks. *Information Processing Letters*, 57:75–81, 1996.

[GKSZ96] S. Ghandeharizadeh, S. H. Kim, C. Shahabi, and R. Zimmermann. Placement of Continuous Media in Multi-Zone Disks. In S. Chung, editor, *Multimedia Information Storage and Management.* Kluwer, 1996.

[GLM95] L. Golubchik, J. C.-S Lui, and R. R. Muntz. Reducing i/o demand in video-on-demand storage servers. In *Proceedings of the ACM SIGMETRICS and Performance '95*, pages 25–36, 1995.

[GRAQ91] S. Ghandeharizadeh, L. Ramos, Z. Asad, and W. Qureshi. Object Placement in Parallel Hypermedia Systems. In *Proceedings of the International Conference on Very Large Databases*, 1991.

[GS93] S. Ghandeharizadeh and C. Shahabi. Management of Physical Replicas in Parallel Multimedia Information Systems. In *Proceedings of the Foundations of Data Organization and Algorithms (FODO) Conference*, October 1993.

[GS94] S. Ghandeharizadeh and C. Shahabi. On Multimedia Repositories, Personal Ccomputers, and Hierarchical Storage Systems. In *Proceedings of ACM Multimedia Conference*, 1994.

[GSZ95] S. Ghandeharizadeh, J. Stone, and R. Zimmermann. Techniques to Quantify SCSI-2 Disk Subsystem Specifications for Multimedia. Technical Report USC-CS-TR95-610, USC, 1995.

[GZS$^+$97] S. Ghandeharizadeh, R. Zimmermann, W. Shi, R. Rejaie, D. Ierardi, and T. Li. Mitra: A Scalable Continuous Media Server. *Kluwer Multimedia Tools and Applications*, January 1997.

[HMM93] S. R. Heltzer, J. M. Menon, and M. F. Mitoma. Logical Data Tracks Extending Among a Plurality of Zones of Physical Tracks of one or More Disk Devices. U.S Patent No. 5,202,799, April 1993.

[KRT94] M. Kamath, K. Ramamritham, and D. Towsley. Buffer Management for Continuous Media Sharing in Multim edia Database Systems. Technical Report 94-11, University of Massachusetts, Feb. 1994.

[KRT95] M. Kamath, K. Ramamritham, and D. Towsley. Continuous Media Sharing in Multimedia Database Systems. In *Proc. of the 4th Intl. Conference on Database Systems for Advanc ed Applications (DASFAA '95)*, Singapore, April 10-13, 1995.

[OBRS94] B. Ozden, A. Biliris, R. Rastogi, and A. Silberschatz. A Low-Cost Storage Server for Movie on Demand Databa ses. *Proc. of the 20th Intl. Conf. on Very Large Data Bases*, Sept. 1994.

[ORS96] B. Ozden, R. Rastogi, and A. Silberschatz. The Storage and Retrieval of Continuous Media Data. In *Multimedia database systems*, 1996.

[RW94] C. Ruemmler and J. Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer*, March 1994.

[SZKT96] J. D. Salehi, Z. Zhang, J. F. Kurose, and D. Towsley. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. In *Proceedings of the 1996 ACM Sigmetrics Conference*, May 1996.

[TPBG93] 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*, August 1993.

[WSY95] J. Wolf, H. Shachnai, and P. Yu. DASD Dancing: A Disk Load Balancing Optimization Sc heme for Video-on-Demand Computer Systems. In *Proceedings of the ACM SIGMETRICS and Performance*, May 1995.

[YCK92] P. S. Yu, M. S. Chen, and D. D. Kandlur. Design and Analysis of a Grouped Sweeping Scheme for Multimedia Storage Management. In *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1992.

[YCK93] 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.

[ZDE$^+$93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP A New Resource Reservation Protocol. *IEEE Network*, September 1993.