Design of Multi-user Editing Servers for Continuous Media *

Shahram Ghandeharizadeh and Seon Ho Kim

Department of Computer Science
University of Southern California
Los Angeles, California 90089

Abstract

Based on a fifteen month investigation of a post production facilities for both the entertainment industry and broadcasters, we identified a number of challenges with the design and implementation of a server in support of multiple editing stations. These include, how to: share the same content among multiple editors, support continuous display of the edited content for each editor, support complex editing operations, compute the set of changes (deltas) proposed by an editor, compare the deltas proposed by multiple editors, etc. It is beyond the focus of this paper to report on each challenge and its related solutions. Instead, we focus on one challenge, namely how to support continuous display of the edited content for each editors, and techniques for physical organization of data to address this challenge.

1 Introduction

Video editing systems have been around for several decades now. They are used extensively by the content providers, e.g., the entertainment industry, broadcasters, etc. Over time and with technical advances, they have evolved to utilize different storage mediums. The original mechanical devises, named Moviola, manipulated film (a positive print and NOT the negative itself). The mid 1970s witnessed editing systems based on analog tape. In the 1990s, editing systems that employed digital tape and magnetic disks became common place. The tape-based systems are termed *linear* editing systems because they incurs a noticeable latency when a user manipulates content stored in a noncontiguous manner. The disk-based systems are termed *nonlinear* editing systems because they can quickly re-position the disk head to a new cylinder without scanning all cylinders, resulting in a lower latency when the user jumps from one video segment to another.

*This research was supported in part by a Hewlett-Packard unrestricted cash/equipment gift, and the National Science Foundation under grants IRI-9203389, IRI-9258362 (NYI award), and ERC grant EEC-9529152.

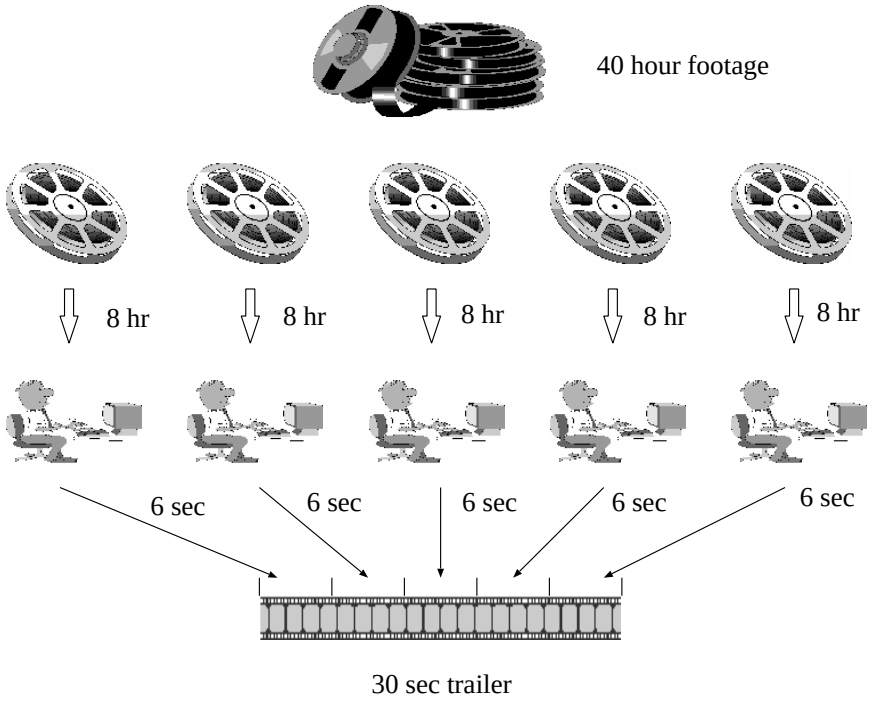


Figure 1: An example of cooperation among editors

Linear editing systems are appropriate for small operations where a few editors work on different projects. They become inefficient for large projects with tight deadlines that require the collaboration of several editors. To illustrate and without loss of generality, we focus on one application from the entertainment industry, namely, the advertisement departments of major studios in Hollywood. (The requirements of this department are similar to other applications in both the entertainment and broadcasting industries.) The advertisement departments are required to generate 30 to 60 second trailers in a short period of time. The original version of the movie might be anywhere from 20 to 40 hours long[1]. The previewing of this content requires more than a day. At times, the deadlines are so tight[2], that an executive decision is made to partition the work among multiple editors. For example, with five editors and a 30 second trailer, the 40 hour footage is shown once to all the editors. Next, each editor is assigned an 8 hour segment and asked to contribute 6 seconds worth of material to the final trailer (see Figure 1). Obviously, the flow of content in the resulting trailer depends on how closely the editors collaborate, communicate, and share their content with one another. With extremely tight deadlines, the 40 hour previewing step might be eliminated all together. Typically, this results in trailers that do not hint at a story line; instead, they are action packed segments. This

[1]The movie is subsequently edited by the director to a shorter version. Since the editors are simultaneously manipulating a copy, scenes shown in a trailer may not appear in the final cut.

[2]Because air-time for showing of the trailer is purchased in advance. For broadcasters that carry with special events (e.g., SuperBowl), air-time might have been purchased several months in advance.

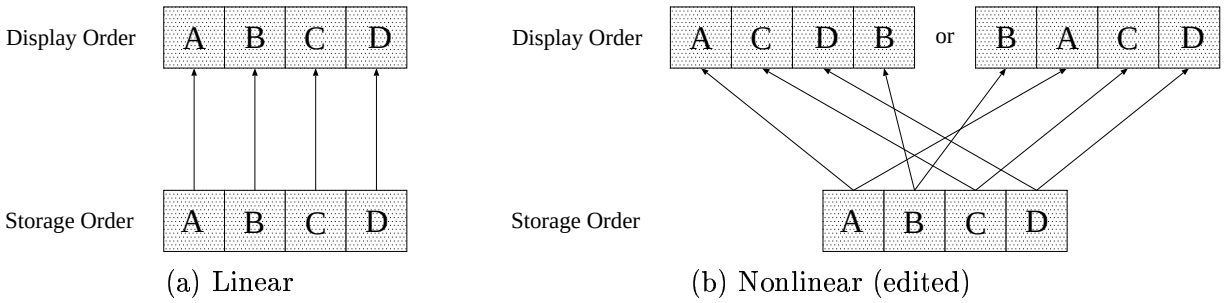


Figure 2: Continuous media presentation

might also happen when editors fail to collaborate closely when authoring their 6 second segment (even when the editors are provided with a preview of the 40 hour footage).

With servers in support of nonlinear editing stations, content can be shared among multiple editors in support of an effective collaboration paradigm. This paradigm empowers several editors to share the 40 hour footage, preview each other's (in progress) work, and make decisions that improve the quality of the final content. With relaxed deadlines, multiple editors can introduce different trailers that are readily available for previewing. In this case, it is desirable to have additional capabilities to reason about the set of changes proposed by each editor (deltas [6]), the difference between two or more deltas, and how to combine aspects of different deltas in order to introduce a new trailer.

This environment can be compared with a client-server software development environment that consists of workstations connected to a file server. The latency for sequentially browsing a file (video) should be less than 100 milliseconds [3]. Similarly, jumping from one line (frame) of the file (video) to another and referencing different files (video clips) should not be noticeable. The client-server nature should not impact the correctness or behavior of a program (previewing of an edited clip). Otherwise, it is difficult to determine whether the program (edited clip) or the system is at fault. To elaborate further, editors are artists who pay attention to details. Once they introduce a special effect and request the system to display the edited clip, they do not want to incur *system hiccups* (disruptions and delays caused by either the disk subsystem or the client-server architecture). This is because a hiccup might be interpreted as a side-effect of their work. This study investigates the physical design of data in support of a hiccup-free display in nonlinear editing servers.

For the purpose of this study, it is worthwhile to differentiate between two kinds of presentations: linear and nonlinear. With a linear presentation, the order of display is identical to the order of original data stored on the disk drives. For example, if video frames in a clip are stored as A, B,

C, D, then a linear presentation of the clip entails a sequential retrieval and display of these frames (Figure 2.a). However, with a nonlinear presentation, the order of display can be different from the order of storage. Moreover, there could be multiple ways of ordering data for different presentations (Figure 2.b). A nonlinear editing server should support displays whose frames are stored in an arbitrary order (logical display orders). Changing the storage order (physical order) or duplicating data is not appropriate due to the noticeable latency incurred by the transfer of large volume of data. To support frame level nonlinear editing, each frame should be uniquely identified in the system. The entertainment industry employs a standard SMPTE timecode (*hour:minute:second:frame*) [15] for this purpose.

Due to the large size of continuous media, transmission of data is based on a just-in-time schedule: data is retrieved from disks and transmitted to the display in a timely manner that prevents hiccups. A cycle-based data retrieval technique [11, 16, 21] is one such approach. We describe this technique with a running example. Assume a video clip A with 4 Mb/s of bandwidth requirement is partitioned into equi-sized blocks of 512 KB. The time to display a block is termed a *time period* and denoted as $T_p = B/R_C$, where B is the size of a block and R_C is the bandwidth requirement of a continuous media type. In our example, a time period is one second long. Assuming constant-bit-rate(CBR) media, when A (starting with A_i) is referenced, the system retrieves A_i and initiates its display. Prior to completion of a time period (its display), it initiates the retrieval of A_{i+1} in order to ensure a continuous display. This process is repeated in a cyclic manner until all blocks of A have been displayed. If the time to retrieve a block, termed *block retrieval time*, is smaller than or equal to the time period then the display will be hiccup free. The time duration between the arrival of a request for A and the start of A_i's display is termed *startup latency*.

Using the concepts from both cycle-based and deadline-driven scheduling techniques, this study proposes data placement, buffering, and scheduling techniques in support of multi-user editing servers for continuous media. First, in Section 2, we investigate alternative retrieval techniques for a single disk system and propose a hybrid technique between frame and block based retrieval. By comparing round-robin and random placement, we determine random placement as the choice of data placement technique across multiple disk drives because random provides a shorter startup latency. However, random might introduce hiccups due to the variance of load balance. In Section 3, we propose several techniques to minimize both hiccup probability and startup latency with random, namely, N buffering with bulk request scheduling and data replication. Our simulation results demonstrate that our proposed techniques minimize hiccup probability to less than one in a million and startup latency to less than a few hundreds of milliseconds even in the worst scenario. Our conclusion and future research directions are contained in Section 4.

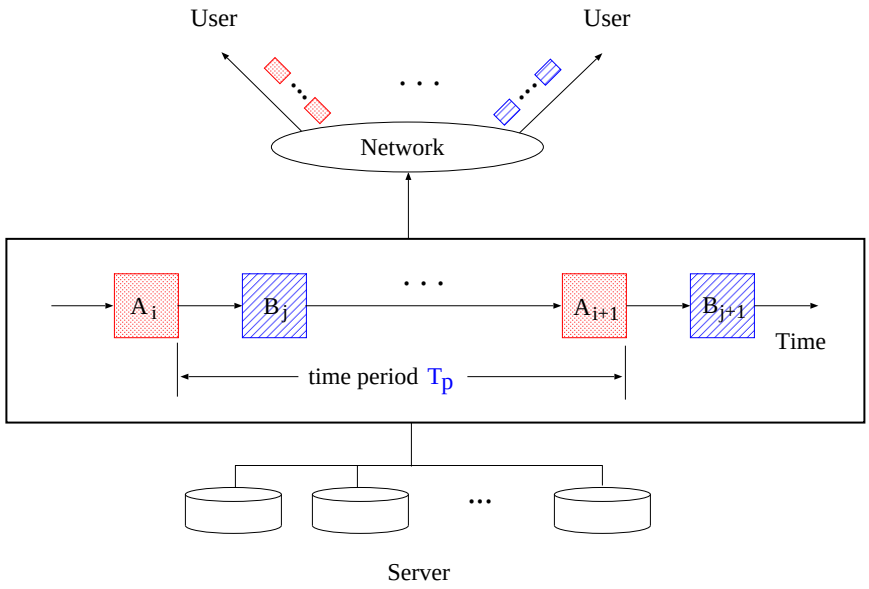


Figure 3: A hiccup free display

2 Data Placement

We start with a discussion of the storage granularity on a single disk and compare three approaches, frame-based, block-based, and a hybrid of these two techniques. Next, Section 2.2 discusses data placement within a multi-zone disk. We describe a placement strategy that is insensitive to the frequency of access to blocks and argue why this strategy is superior. Finally, Section 2.3 analyzes the placement of data across multiple disks.

2.1 Granularity of Storage and Retrieval

A simple and naive approach might store and retrieve data at the granularity of a frame (*FF*). Data in a frame is contiguously stored in a disk drive. This approach suffers from the following limitation. If the size of a frame is too small, the portion of wasteful work attributed to disk arm movements, i.e., seeks, becomes significant, resulting in a poor utilization of disk bandwidth [1, 9].

An alternative is to store and retrieve data at the granularity of a block, termed *BB*. A block consists of a number of frames and data in a block is contiguously stored on a disk platter. This approach is flexible because one can determine the size of a block and the number of frames it contains. For example, if the size of a block is small enough to accommodate only a single frame then it becomes identical to *FF*. With linear presentations such as movies, increasing block size

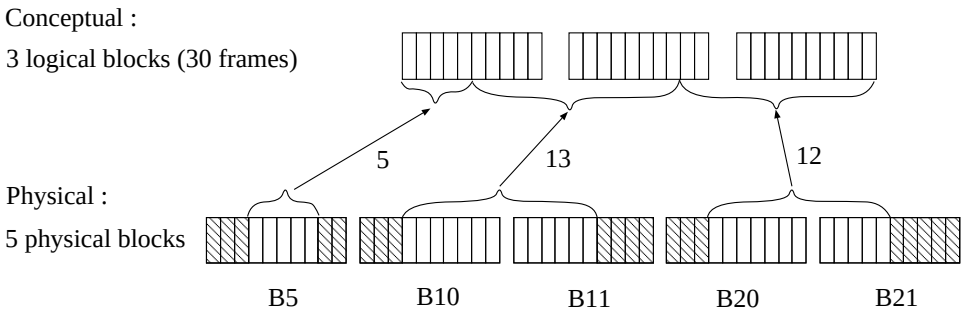


Figure 4: Disk bandwidth waste in BB ($N_T = 2$)

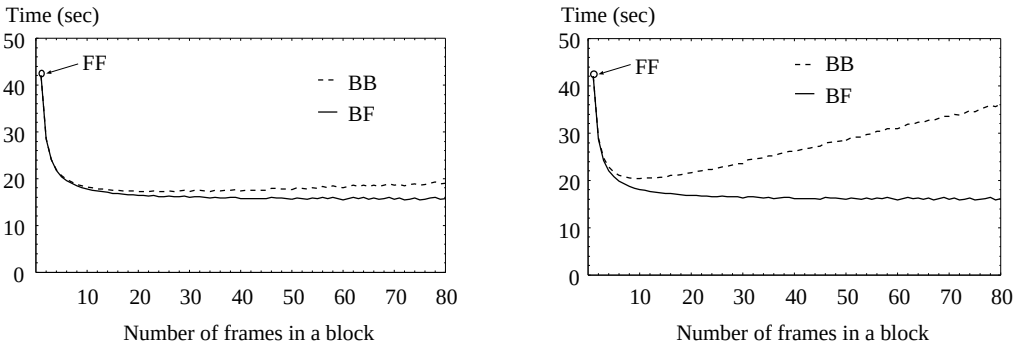
maximizes the utilization of disk bandwidth. However, with nonlinear editing systems, large block sizes introduce a new limitation that wastes disk bandwidth. An edited clip produced by an editor is a sequence of shots where a shot is a sequence of frames. This sequence may consist of either one or multiple frames. Different shots might have been obtained from different parts of original clips. A transition from one shot to another may waste disk bandwidth because it might result in retrieval of irrelevant data. To elaborate, consider an editor who wants to display an edited clip that consists of 3 shots. These shots consist of 30 frames. Conceptually, if 10 frames fit per block then the system should retrieve 3 blocks, termed logical blocks. However, depending on the organization of frames across physical blocks, the number of blocks retrieved may exceed 3. For example, in Figure 4, the system retrieves 5 physical blocks. The portion of data which is retrieved and not displayed constitutes wasteful work performed by the disk subsystem.

The number of transitions in a clip is dependent on its target applications. For example, while some commercials incur more than thirty transitions per minute, documentaries typically incur fewer than five transitions per minute. The degradation in system performance can be quantified as a function of the transitions. Assuming only one transition[3] occurs within a logical block, a transition invokes one extra seek (T_S: the average seek time) and block read on the average. (Appendix A relaxes this simple assumption and outlines accurate calculation based on the exact statistics of a clip.) Let T_F denote the time to read a frame and F_B to denote the number of frames in a block, then the total time (T_T) to read F_T frames with N_T transitions is:

$$T_T = (\lceil \frac{F_T}{F_B} \rceil + N_T)(T_S + F_B T_F) \quad (1)$$

In this equation, $\lceil \frac{F_T}{F_B} \rceil$ is the number of logical blocks (three in Figure 4). This plus N_T (two in Figure 4) defines the number of physical blocks.

[3]We assume that a transition is a switch from the current physical block to a different one.



(a) $N_T = 5$ (b) $N_T = 30$

Figure 5: Retrieval time of a one minute clip with various block sizes ($F_T = 1800$)

One may employ a hybrid approach using both *BB* and *FF* (termed *BF*). The system can use a block-based retrieval when all the frames in a physical block are required, while applying a frame-based retrieval when only a few frames in a physical block are required. *BF* eliminates both the overhead caused by the excessive seeks with *FF* and retrieval of unnecessary data with *BB*. However, additional seeks attributed to transitions are still unavoidable. Hence, Equation 1 can be modified as follows.

$$T_T = \lceil \frac{F_T}{F_B} \rceil \ (T_S + F_B T_F) + N_T T_S \tag{2}$$

Figure 5 shows the time to retrieve a one minute video clip with both the block-based (*BB*) and hybrid (*BF*) approaches. The x-axis of this figure denotes the size of a block with *BB*, measured in terms of the number of frames that fit in a block. The y-axis denotes the time to retrieve a one minute video clip. Figure 5.a presents results for a clip that incurs 5 transitions on the average while Figure 5.b is a clip that incurs 30 transitions. Note that *FF* is a special case of *BB* or *BF* when the number of frames in a block is one. *FF* requires a longer service time when compared with both *BB* and *BF* because of the excessive seeks. With a small number of transitions ($N_T = 5$), the difference between *BB* and *BF* is insignificant. However, with $N_T = 30$, while the performance of *BF* remains steady as a function of block size, the performance of *BB* starts to degrade beyond a certain point. This is because each transition incurs a higher disk transfer time. In sum, *BF* is a superior alternative to both *FF* and *BB*.

Zone #	Size (MB)	Rate (MB/s)
0	506.7	8.79
1	518.3	8.74
2	164.1	8.37
3	134.5	8.03
4	116.4	7.70
5	121.1	7.41
6	119.8	6.99
7	103.2	6.66
8	101.3	6.34
9	92.0	5.94
10	84.6	5.61

(a) Seagate Barracuda 4LP, ST32171W

Zone #	Size (MB)	Rate (MB/s)
0	136.1	7.27
1	168.9	7.07
2	155.0	6.91
3	185.5	6.76
4	105.7	6.62
5	135.9	6.45
6	140.5	6.26
7	128.0	6.09
8	275.8	5.74
9	107.6	5.48
10	114.9	5.26
11	112.6	4.99
12	107.8	4.77
13	76.4	4.54
14	100.0	4.29

(b) Quantum Atlas XP32150

Figure 6: Zone information of two commercial disks

2.2 Data Placement within a Multi-zone Disk

Modern disk drives are produced with multiple zones to meet the demands for a higher storage capacity [18]. A zone is a contiguous collection of disk cylinders whose tracks have the same storage capacity. The outer zones provide a higher transfer rate as compared to the inner ones. This is because tracks are longer towards the outer cylinders of a disk and can store more data as compared to the inner ones. While zoning increases the storage capacity of a disk, it produces a disk that has multiple data transfer rates depending on the location of data within a disk (see Figure 6 for two different disk models). Therefore, the block reading time varies significantly depending on block placement within a disk drive. For example, if a 1 MB block is assigned to the outermost zone of the Seagate Barracuda disk (Figure 6.a), its transfer time is 113.8 msec. If the same block is now assigned to the innermost zone, its transfer time increases to 178.3 msec. Thus, when a server is required to support multiple media types with different bandwidth requirements (i.e., different block sizes), the block reading time will vary widely depending on the block size and its assigned zone. This may result in a higher block retrieval time and consequently a higher hiccup probability. In this section, we describe several data placement techniques and quantify their performance tradeoffs using a simulation study.

Define m as the number of zones in a disk, R_i as the data transfer rate at zone i, n as the number of media types, D_i as the bandwidth requirement of media type i ($1 \leq i \leq n$), and $B_i = T_p \times D_i$

as the block size of an object with media type i assuming a fixed time period (T_p) for all the media types. Because of multiple block sizes and disk transfer rates, the time to read a block (service time) varies depending on the location of a data block within a disk. When the system retrieves a block of type i located in zone j, its transfer time (service time)[4] is $s_{i,j} = \frac{B_i}{R_j}$. When there are b blocks in the system, the average service time is:

$$\bar{s} = \sum_{i=1}^{b} F_i \sum_{j=1}^{n} P_{i,B_j} \sum_{k=1}^{m} \frac{P_{i,Z_k} B_j}{R_k} \tag{3}$$

where F_i is the access frequency of block i, P_{i,B_j} is the probability that the size of block i is B_j, and P_{i,Z_k} is the probability that this block is assigned to zone k. The variance of service time is:

$$\sigma_s^2 = \sum_{i=1}^{b} F_i \sum_{j=1}^{n} \sum_{k=1}^{m} P_{i,B_j} P_{i,Z_k} (s_{j,k} - \bar{s})^2 \tag{4}$$

The block retrieval time (ω) in a multi-user environment consists of the service time and waiting time in a disk queue. It is determined by the service time and its variance when the system load and access frequency are fixed. We compare three block placement approaches in this paper.

- Random placement (RP): This simple technique assigns blocks to the zones in a random manner.

- Maximizing throughput placement (MTP): Several studies attempted to minimize the average disk service time ($\bar{s}$) by assigning large blocks with a high frequency of access to faster zones [7, 19]. MTP sorts blocks based on their size and frequency of access ($F_i \times B_i$ for each block i). Next, blocks are assigned to the zones sequentially starting with the fastest zone. Block i with the highest $F_i \times B_i$ value is assigned to the fastest zone.

- Minimizing variance placement (MVP): The objective of this technique is to reduce both the service time and its variance. With MVP, a block of size B_i is placed on the zone Z_j (with R_j) which has the closest B_i/R_j value to the average block reading time ($\bar{T_B}$):

$$\bar{T_B} = \frac{average\ block\ size}{average\ transfer\ rate} = \frac{\frac{1}{n}\sum_{i=1}^{n} B_i}{\frac{1}{m}\sum_{i=1}^{m} R_i}$$

  Note that MVP is independent of frequency of access to the blocks.

[4]Seek time is not considered in this calculation for a simple discussion. Seek time can be considered as a constant and added to $s_{i,j}$.

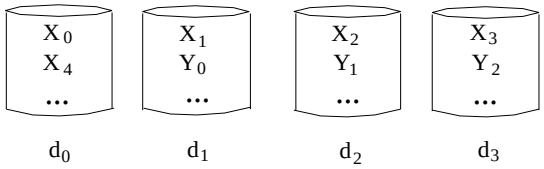


Figure 7: Round-robin placement

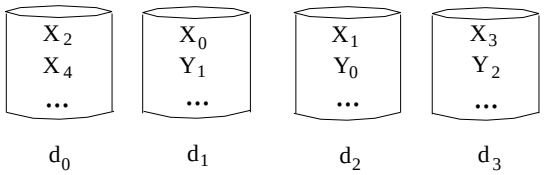


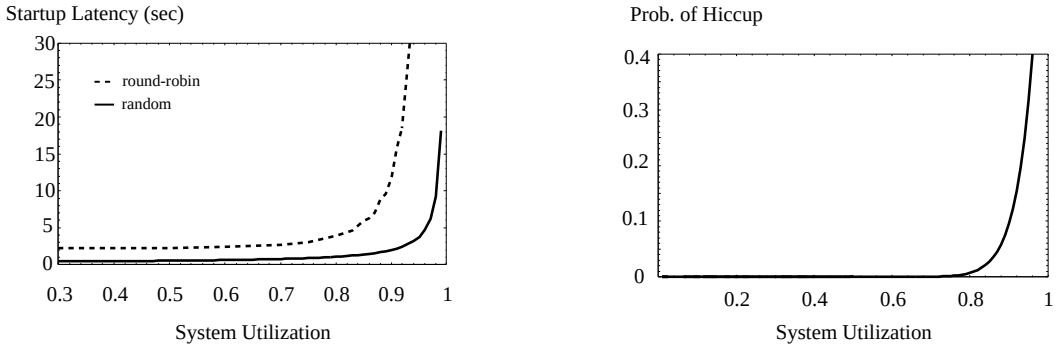
Figure 8: Random placement

We have compared these techniques using a simulation model, see Appendix B. Our performance results indicate that both MTP and MVP are superior to RP. MTP does outperform MVP at times. However, in these cases, MVP's variance in retrieval time is lower than MTP's (approximately 50% lower). One advantage of MVP is that it is not sensitive to the access frequency of objects. With MTP, if the access frequency is either inaccurate or changes over time, the placement of data could result in a low system performance. (Reorganization of data with MTP is a non-trivial research topic with a number of open-ended questions, such as When to trigger reorganization ? What objects to migrate ? Where to migrate these objects ? etc.)

2.3 Data Placement across Disk Drives

Assuming a system with d homogeneous disks, the data is striped [1, 10, 16] across the disks in order to distribute the load of a display evenly across the disks. There are a number of ways to assign blocks of an object to the d disks. We consider two, namely, *round-robin* and *random*. With round-robin, blocks are assigned to disks in a round-robin manner starting with an arbitrarily chosen disk [11, 16, 1]. For example, Figure 7 depicts a system with four disk drives where the assignment of X starts with disk d_0. Another way to distribute the system load across the disks is to employ a random placement of data [14, 20], see Figure 8.

Figure 9.a shows the average startup latency of these two alternatives as a function of system load (utilization). Results from various configurations showed similar trends (see details in [8]). In all cases, the average startup latency with round-robin is higher than with random. With a high system utilization this difference becomes more profound. In sum, random is more appropriate for



(a) Average startup latency (b) Prob. of hiccup with random

Figure 9: round-robin vs. random

latency-sensitive applications than round-robin. Round-robin cannot support any arbitrary display order in editing application because the pattern of access to the data might changed arbitrarily when an editor changes shot sequence. However, random incurs hiccups that are attributed to the statistical variation of the number of block requests per disk drive. In particular, a single disk could become a potential bottleneck with an uneven distribution of block references on it. Figure 9.b shows an example of the probability of hiccups with random as a function of the system utilization. With a utilization higher than 0.8, many displays may suffer due to a high probability of hiccup. This probability must be minimized in order for random to be suitable for editing applications.

With random, when a disk receives more than its fair share of block requests at an instance in time, it become a bottleneck for the entire system, increasing block retrieval time (ω) which consists of service time (block reading time) and waiting time in a disk queue. In this section, we measure block retrieval time with a traditional double buffering technique and describe the relation between block retrieval time and hiccup probability.

Traditionally, double buffering (Figure 12.a) has been widely used to absorb the variance of block retrieval time[24, 10, 22]. The idea is as follows: while a buffer is being consumed from memory, the system fills up another memory frame with data. The system can initiate display after the first buffer is filled and a request for the next one is issued.

Figure 10 presents some simulation results of block retrieval time distributions with a random placement. In these simulations, we assumed that: 1) a block consisted of 30 frames and the duration of a time period was one second long ($T_p = 1$), 2) requests followed the Poisson arrival pattern, and 3) a constant service time of 100 msec based on a simple single zone disk model with a fixed disk transfer rate and a constant-bit-rate media type. Assuming a constant service time, we quantify the

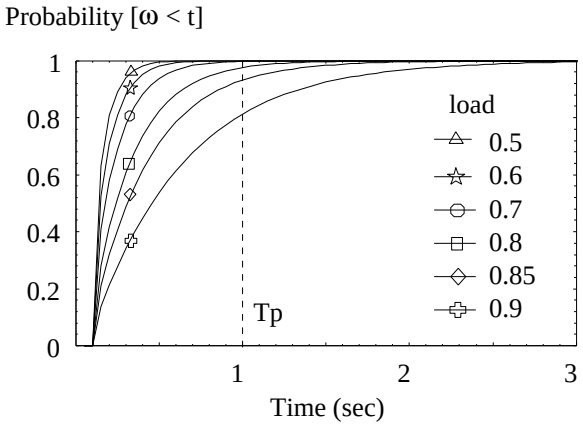


Figure 10: Retrieval time distribution

Load (ρ)	$\bar{\omega}$ (msec)	Max. ω (msec)	Probability		
			$\omega > 1T_p$	$\omega > 2T_p$	$\omega > 3T_p$
0.50	151.2	1138	0.000030	0.0	0.0
0.55	162.5	1229	0.000070	0.0	0.0
0.60	176.7	1304	0.000160	0.0	0.0
0.65	195.5	1367	0.000460	0.0	0.0
0.70	220.6	1441	0.001570	0.0	0.0
0.75	256.1	2036	0.006430	0.000020	0.0
0.80	308.3	2621	0.018990	0.000450	0.0
0.85	397.9	3415	0.055720	0.002630	0.000200
0.90	597.6	4286	0.177530	0.023490	0.003140

Table 1: Examples of retrieval time distributions

impact of waiting time variance attributed to random placement across disks on the distribution of retrieval time[5]. As shown in Table 1, some block requests experience retrieval time (ω) longer than the average. For example, when the system load is 0.8, 1.9% of requests experience hiccups, i.e., longer retrieval time than a time period (1.0 sec), with 0.045 % of these requests experiencing delays longer than two time period (2.0 sec). With double buffering, a display does not incur a hiccup when retrieval time of a block is either equal to or less than a time period long.

In Section 3, we outline several strategies to resolve bottlenecks in order to minimize the hiccup probability with random. These techniques are N buffering with bulk request scheduling and replication. Both minimize hiccup probability and startup latency. N buffering accomplishes this by requiring a client to allocate a portion of its memory for continuous display, while data replication requires extra disk space at the server.

[5]We exclude the impact of service time variance attributed to the data placement techniques within a multi-zone disk in Section 2.2 from this discussion.

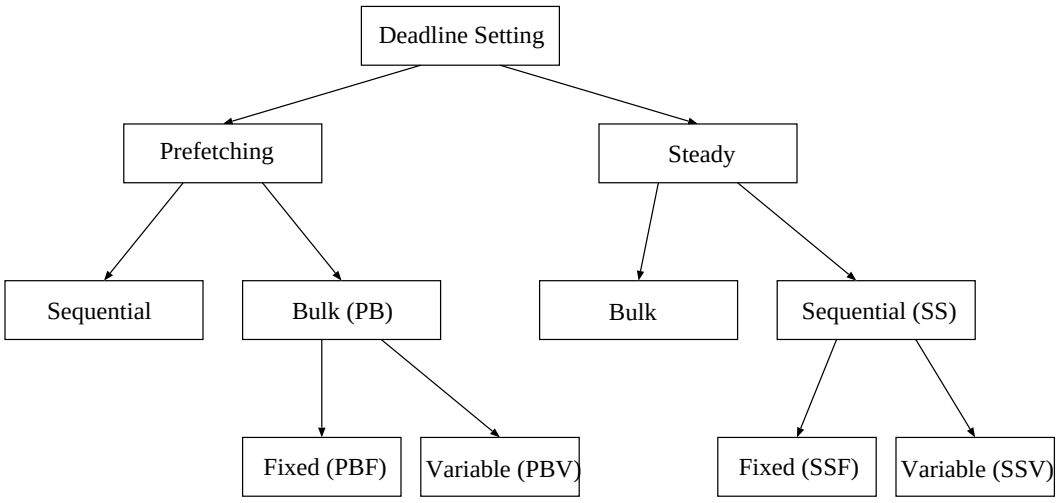


Figure 11: A taxonomy of deadline setting techniques

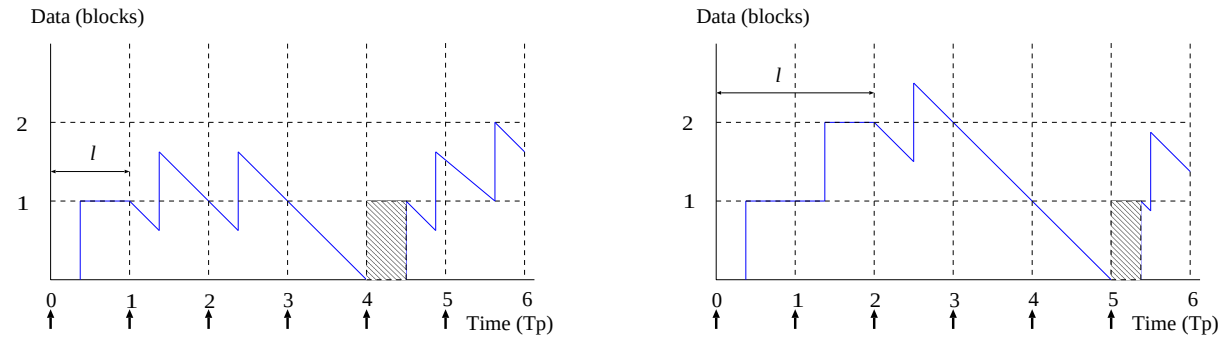
3 N Buffering and Data Replication

Without loss of generality and in order to simplify the discussion, this section assumes that a logical block is equivalent to a physical block. This is because Section 2 has already considered the impact of a logical block translating to multiple physical blocks. We start with a description of N buffering. Next, Section 3.2 introduces data replication.

3.1 N Buffering

One can generalize double buffering to *N buffering* by using N buffers and prefetching N-1 blocks before initiating a display. The system can continue to request a block every time a memory buffer becomes empty. This minimizes the probability of hiccup to $p[\omega > (N-1)T_p]$ because the retrieval time of a block must now exceed (N-1) time periods in order for a display to incur a hiccup. As demonstrated by Table 1, when the system load is 0.85, the hiccup probability decreases by a factor of 20 when N is increased from 2 to 3. With N buffering, there exist alternative ways of prefetching data and scheduling block retrievals. Figure 11 outlines a taxonomy of different techniques using a deadline driven servicing policy where each block is tagged with a deadline and the disk services requests using an earliest-deadline-first (EDF) policy.

This taxonomy differentiates between two stages of block retrieval on behalf of a display: (1) *prefetching* stage that requests the first N-1 blocks and (2) *steady* stage that requests the remaining blocks. The system may employ a different policy to tag blocks that constitute each stage. Fur-



(a) N=2 (b) N=3

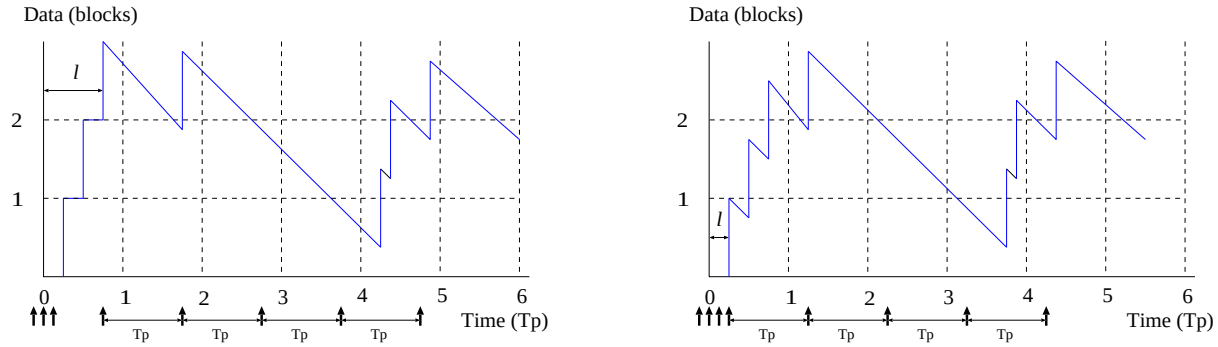
Figure 12: N buffering technique: prefetching with Sequential

thermore, blocks can be issued either in a *Bulk* or *Sequential* manner. With Bulk, all requests are issued at the same time while, with Sequential, requests are issued one at a time whenever a buffer in client's memory becomes free. Note that Bulk is irrelevant during a steady stage because it is very expensive to prefetch the entire clip at the client. This explains why the Bulk branch is as a leaf node of steady. Similarly, Sequential is irrelevant during prefetching because N buffers are available and our objective is to minimize startup latency[6]. The remaining leaves of the taxonomy are categorized based on how they assign deadline to each block: either *Fixed* or *Variable*. With Fixed, all block requests have identical deadlines while, with Variable, requests might have different deadlines.

During the steady stage (SS), a client issues a block request when a buffer in its memory becomes free. Typically, a memory buffer becomes free every time period because the display time of a block is one time period long. With SSF, a fixed deadline[7], $(N-1)T_p$, is assigned to all steady requests to maximize the tolerable variance of block retrieval time to prevent hiccups. However, with SSV, deadlines are determined by the number of blocks in the buffer. If the number of un-displayed blocks in the buffer is k when a block request is issued, then its deadline is set to $k \times T_p$. SSV strives to maintain the maximum data in the buffer by making the buffer full as soon as possible, while SSF strives to prevent the data starvation in the buffer. The results demonstrate that both techniques provide an almost identical performance.

[6]If block requests are issued sequentially during the prefetching stage, the startup latency would increase as a linear function of N, see Figure 12.

[7]We are using this notation for simplicity in this paper but the real deadline is $t_{issue} + (N-1)T_p$, where t_{issue} is the time that this request is issued.



(a) Conservative display (b) Aggressive display

Figure 13: N buffering with prefetching bulk requests (N=4)

3.1.1 PB: Bulk Dispatching of Blocks During Prefetching Stage

With Bulk, when a clip A is referenced, N-1 requests are concurrently dispatched to the server for its first N-1 blocks, see Figure 13.a. This section describes alternative strategies for (a) when to initiate the display of A relative to the arrival of the N-1 blocks? and (b) how to set the deadline for these bulk requests? Consider each in turn. A client may initiate display in two alternative ways: either 1) once all N-1 blocks have arrived, termed Conservative Display (CD), or 2) upon the arrival of block A_0, termed Aggressive Display (AD). In Section 3.1.3, we compare these two alternatives. The results demonstrate that AD is superior to CD.

The deadline assigned to the first N-1 blocks can be either fixed (termed *Fixed*, PBF) or variable (termed *Variable*, PBV). With PBV, block B_i is tagged with $(i+1)T_p$ as its deadline. Assuming that a client initiates display when all N-1 blocks have arrived (i.e., CD), the startup latency is determined by the longest retrieval time of the first N-1 requests, $max(\omega_0, ..., \omega_{N-2})$ where ω_i is the retrieval time of block B_i. PBF is more aggressive because it can set the deadline for all N-1 requests to T_p in order to minimize startup latency. These requests might compete with block requests issued by other clients that are in their steady stage, increasing the probability of hiccups. However, this increase is negligible because the number of clients that are in their prefetching stage is typically small.

Section 3.1.3 compares these four alternatives, namely, PBF-CD, PBF-AD, PBV-CD, PBV-AD. The results demonstrate that PBV-AD provides a performance almost identical to PBF-AD. These two techniques are superior to the other alternatives.

3.1.2 Two Approaches to Handle Hiccups

While our proposed techniques strive to minimize hiccups, they cannot eliminate them all together. Moreover, the policy used at the client to respond to a hiccup impacts the server. To elaborate, a client may respond in two alternative ways to a hiccup: either wait for the missing data indefinitely (termed *Wait*) or skip the missing data and continue the display with remaining blocks (termed *Skip*). In the first case, the display is resumed upon the arrival of the missing data. This means that the server must service all block requests, even those whose deadline has been violated. With Skip, the server may discard these block requests because a client no longer needs them, minimizing the server load.

With Skip, in addition to skipping content with hiccups, every time that a display incurs a hiccup, the probability of it incurring another hiccup increases exponentially. This is because the waiting tolerance of N buffering decreases to that of N-1 buffering since the number of buffers in the client's memory is reduced to N-2. One may extend Skip to delay the display of remaining blocks by one T_p in order to prevent this undesirable situation. (There is no advantage to making Skip delay for multiple time periods.)

As detailed in Section 3.1.3, Skip results in a lower hiccup probability when compared with Wait because it reduces the server load. In passing, it is important to note that a client should not issue block requests while waiting because its buffers may overflow.

3.1.3 Evaluation

We evaluated various techniques described in Section 3.1 using a trace driven simulation study. This trace was generated synthetically using a Poisson arrival pattern. In all simulations presented in this section, we assumed a single media type with 4 Mb/s bandwidth requirement and a block size of 0.5 Mbytes (T_p = 1 second). Blocks were distributed across twenty Quantum Atlas XP32150 disks using random. The disk model for Quantum Atlas XP32150 disk is identical to the multi-zone model shown in Figure 6.b. Our repository consisted of 50 different edited clips each consisting of 120 logical blocks. We assumed a uniform distribution of access to the clips.

All results demonstrate that the hiccup probability decreases as a function of N (number of buffers). For example, we can reduce the hiccup probability to less than one in a million by increasing the number of buffers to seven. This would be satisfactory for almost all applications.

	techniques	number of buffers (N)					
		2	3	4	5	6	7
avg. retrieval time (msec)	Skip	296.4	338.7	348.8	351.9	352.6	352.6
	Wait	336.6	347.9	351.9	352.4	352.6	352.6
hiccup probability	Skip	0.011388	0.001195	0.000238	0.000043	0.000002	$< 10^{-6}$
	Wait	0.042648	0.005485	0.001064	0.000212	0.000005	$< 10^{-6}$

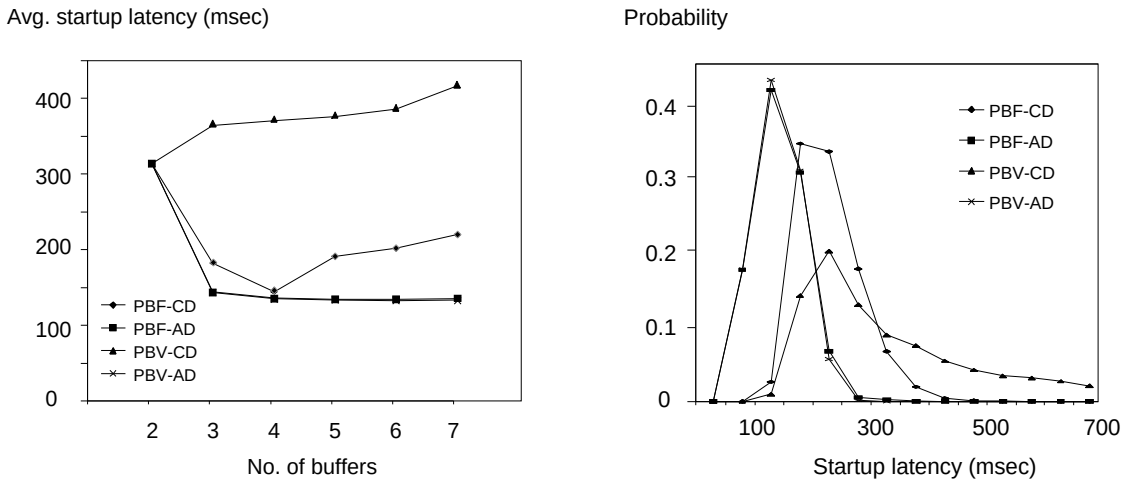
Table 2: Skip vs. Wait (utilization = 0.807)

	techniques	number of buffers (N)					
		2	3	4	5	6	7
avg. retrieval time (msec)	PBF-CD	335.7	348.1	352.5	354.3	355.4	357.1
	PBF-AD	335.7	348.1	352.7	354.3	355.6	357.0
	PBV-CD	335.7	347.8	350.5	352.7	353.8	355.9
	PBV-AD	335.7	347.7	352.5	354.0	355.6	357.3
hiccup probability	PBF-CD	0.041364	0.005574	0.001200	0.000293	0.000027	$< 10^{-6}$
	PBF-AD	0.041364	0.005598	0.001213	0.000281	0.000023	$< 10^{-6}$
	PBV-CD	0.041364	0.005276	0.001115	0.000196	0.000021	$< 10^{-6}$
	PBV-AD	0.041364	0.005681	0.001212	0.000261	0.000021	$< 10^{-6}$
avg. startup latency (msec)	PBF-CD	318.2	186.6	148.7	195.9	206.2	224.2
	PBF-AD	318.2	148.5	140.5	139.4	138.5	139.8
	PBV-CD	318.2	369.0	375.3	380.1	389.8	420.6
	PBV-AD	318.3	147.7	139.5	138.1	136.9	137.7
standard deviation of startup latency	PBF-CD	281.7	144.2	93.5	59.5	48.3	52.9
	PBF-AD	281.7	106.0	65.4	48.0	40.7	40.5
	PBV-CD	281.7	325.3	334.6	326.2	327.5	351.2
	PBV-AD	281.7	103.8	64.4	45.1	38.1	37.4
worst startup latency (msec)	PBF-CD	2861	2708	2430	1883	881	512
	PBF-AD	2861	2534	2524	1999	980	406
	PBV-CD	2861	3515	4409	4373	4740	5087
	PBV-AD	2861	2552	2437	1828	891	336

Table 3: PB techniques (utilization = 0.807)

Table 2 shows that Skip provides a lower hiccup probability than Wait because Skip minimizes server's load by not servicing requests that have already violated their deadlines. This difference becomes more profound when the number of buffers decreases because this increases the percentage of requests that miss their deadline. It is important to note that the use of Skip is application dependent. Those applications that can tolerate skipping content should use Skip in order to minimize the probability of hiccup. For the rest of this evaluation, we assume the Wait scheme.

Table 3 shows a comparison of alternative PB techniques. The results demonstrate that the probability of hiccups is almost identical with all techniques. Figure 14.a shows the average startup latency as a function of N. Figure 14.b shows the distribution of startup latency when N=7. The y-



(a) Avg. startup latency (b) Startup latency distribution (N=7)

Figure 14: Startup latency distribution of PB techniques

axis of this figure is the probability of a display incurring a certain startup latency. For example, the peak point of PBV-AD illustrates that 44% of displays experiences startup latency between 100 msec and 150 msec. Aggressive display (AD) approach provides a better startup latency distribution, i.e., a smaller average and variance, than the conservative display (CD) approach. Overall, both PBV-AD and PBF-AD are superior to the other alternatives when it comes to startup latency. Hence, this can satisfy almost all the latency-sensitive applications, even in the worst case scenario (336 msec), with a hiccup probability that is less than one in a million ($< 10^{-6}$).

Similar trends were observed with our other simulation studies that utilized with different parameters such as utilization, block size, number of disks. Generally, a lower hiccup probability is observed when the system utilization is lower. The number of disks does not impact the performance unless it is too small (less than 6). With a smaller block size, the portion of disk seek time becomes greater and it results in a higher disk utilization and a lower hiccup probability.

3.2 Data Replication

An alternative approach to reduce the hiccup probability with random is to replicate blocks. Assuming that each block has two copies, termed primary and secondary, they are randomly assigned to disks such that the two copies reside on different disks. The system can employ the disk with fewer requests when servicing a block request. To illustrate, assume that all blocks of a clip X have a secondary copy. When a request for block X_i arrives, the system locates the two disks that contain

Load	Hiccup probability		
	no replication	25% replication	full replication
0.50	0.000002	0.0	0.0
0.55	0.000026	0.0	0.0
0.60	0.000100	0.000001	0.0
0.65	0.000441	0.000005	0.0
0.70	0.001762	0.000018	0.0
0.75	0.006069	0.000058	0.0
0.80	0.020525	0.000231	0.0
0.85	0.067085	0.001008	0.000001
0.90	0.197485	0.023567	0.013934

Table 4: Hiccup probability with data replication

its primary and secondary copies. After comparing the number of requests in two queues, the system assigns the request to the disk with fewer requests, resulting in a reduction of the probability of hiccups. The disadvantage of data replication is its extra space requirement, which might be significant due to the large clip sizes. We quantified the hiccup probability with random while varying the amount of extra space for replication. In our simulations, we assumed fifty movies which are one-hour long and requires a 4 Mb/s bandwidth. The total database size was 90 GBytes in size.

Table 4 shows the hiccup probability with various amount of space for replication. We assumed a traditional double buffering in these simulations such that a hiccup occurs when the block retrieval time is longer than the duration of one time period. First, we randomly selected 25% of blocks and replicated them across the disks. This provides a hiccup probability that is two orders of magnitude smaller than the case without replication (see the third column). Next, we replicated all blocks (full replication) in the database. This minimizes the hiccup probability further compared with the 25% replication, see the fourth column. However, this approach doubles the disk space requirement (180 GBytes of total disk space).

4 Conclusion and Future Directions

This paper describes the requirements and design of client-server architectures in support of nonlinear editing systems. Due to lack of space, we focused on the server side of the system. After analyzing the tradeoffs between frame and block based retrieval, we proposed a hybrid approach to maximize the bandwidth utilization. We also compared two alternative data placement strategies for a multi-disk platform and showed that a random placement incurs a lower startup latency compared with a

round-robin placement. Subsequently, we proposed and evaluated buffering techniques to minimize both the hiccup probability and startup latency with random placement. We are extending our approach for the synchronized presentation of multimedia such as a video and eight audio channels. We are also investigating several possible client-server models for nonlinear editing systems that include local disk caching for continuous media data.

References

[1] 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.

[2] S.M. Chung, editor. *Multimedia Information Storage and Management.* Kluwer Academic Publishers, 1996.

[3] P. J. Denning. The Working Set Model for Program Behavior. *Communications of the ACM*, 11(5):323–333, 1968.

[4] M. Garofalakis, B. Ozden, and A. Silberschatz. Resource Scheduling in Enhanced Pay-Per-View Continuous Media Databases. In *Proceedings of the International Conference on Very Large Databases*, 1997.

[5] A. Ghafoor. Special Issue on Multimedia Database Systems. *ACM Multimedia Systems*, 3(5-6), November 1995.

[6] S. Ghandeharizadeh, R. Hull, and D. Jacobs. Design, Implementation, and Application of Heraclitus[Alg,C]. *ACM Transactions on Database Systems*, 21(3), September 1996.

[7] S. Ghandeharizadeh, D. Ierardi, D. Kim, and R. Zimmermann. Placement of Data in Multi-Zone Disk Drives. In *Proceedings of Second International Baltic Workshop on DB and IS*, June 1996.

[8] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi. On Disk Scheduling and Data Placement for Continuous Media. *Submitted to IEEE Tansactions on Knowledge and Data Engineering.*

[9] S. Ghandeharizadeh, S. H. Kim, and C. Shahabi. On Configuring a Single Disk Continuous Media Server. In *Proceedings of the ACM SIGMETRICS*, May 1995.

[10] S. Ghandeharizadeh and S.H. Kim. Striping in Multi-disk Video Servers. In *High-Density Data Recording and Retrieval Technologies*, pages 88–102. Proc. SPIE 2604, October 1995.

[11] S. Ghandeharizadeh, S.H. Kim, W. Shi, and R. Zimmermann. On Minimizing Startup Latency in Scalable Continuous Media Servers. In *Proceedings of Multimedia Computing and Networking*, pages 144–155. Proc. SPIE 3020, Feb. 1997.

[12] Ng C. Hock. *Queueing Modeling Fundamentals*, page 140. John Wiley & Sons, 1996.

[13] L. Kleinrock. *Queueing Systems Volume I: Theory*, page 105. Wiley-Interscience, 1975.

[14] R. Muntz, J. Santos, and S. Berson. RIO: A Real-time Multimedia Object Server. *ACM Sigmetrics Performance Evaluation Review*, 25(2), Sep. 1997.

[15] T. A. Ohanian. *Digital Nonlinear Editing - New Approaches to Editing Film and Video.* Focal Press, 1993.

[16] B. Ozden, R. Rastogi, and A. Silberschatz. Disk Striping in Video Server Environments. In *IEEE International Conference on Multimedia Computing and System*, June 1995.

[17] P. Rangan and H. Vin. Efficient Storage Techniques for Digital Continuous Media. *IEEE Transactions on Knowledge and Data Engineering*, 5(4), August 1993.

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

[19] R. Tewari, R. King, D. Kandlur, and D.M. Dias. Placement of multimedia blocks on zoned disks. In *Proceedings of Multimedia Computing and Networking*, Jan. 1996.

[20] R. Tewari, R. Mukherjee, D.M. Dias, and H.M. Vin. Design and Performance Tradeoffs in Clustered Video Servers. In *Proceedings of IEEE ICMCS*, June 1995.

[21] R. Tewari, R. Mukherjee, D.M. Dias, and H.M. Vin. Design and Performance Tradeoffs in Clustered Video Servers. In *IEEE International Conference on Multimedia Computing and System*, June 1996.

[22] F.A. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming RAID-A Disk Array Management System for Video Files. In *Proceedings of the First ACM Conference on Multimedia*, August 1993.

[23] V.S.Subrahmanian and S.Jajodia, editors. *Multimedia Database Systems*. Springer, 1996.

[24] 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.

Appendix A: Expected Time to Retrieve a Clip

An edited clip is a sequence of shots; a shot is one or multiple frames generated and recorded contiguously and represents a continuous action in time and space. A shot is a sequence of frames; a frame is an image and the smallest meaningful entity in video. We can quantify the time to retrieve a shot with *BF* approach. Let F_S denote the number of frames in a shot and F_B to denote the number of frames in a block. The number of seeks to retrieve a shot, F_S contiguous frames, depends on the location of those frames on physical blocks. For example, when F_S is 15 and F_B is 10, this shot may be stored either in two physical blocks (2 seeks) or in three (3 seeks). The minimum number of seeks (S_{min}) and the maximum number of seeks (S_{max}) to retrieve a shot with F_S frames is:

$$S_{min} = \lfloor \frac{F_S - 1}{F_B} \rfloor + 1$$

$$S_{max} = S_{min} + 1 = \lceil \frac{F_S - 1}{F_B} \rceil + 1$$

Assuming every frame has the same probability to be selected as a starting frame in a shot, the probability that S_{max} is required to retrieve a shot is:

$$P[S_{max}] = \frac{(F_S - 1) \ mod \ F_B}{F_B}$$

The probability that S_{min} is required to retrieve a shot is:

$$P[S_{min}] = 1 - P[S_{max}] = 1 - \frac{(F_S - 1) \ mod \ F_B}{F_B}$$

Thus, the expected number of seeks in retrieving a shot is:

$$P[S_{max}] \times S_{max} + P[S_{min}] \times S_{min}$$

When an edited clip contains N_T transitions, i.e., $N_T + 1$ shots with F_{S_i} $(i = 0, ..., N_T)$, and the total number of frame $F_T = \sum_{i=0}^{N_T} F_{S_i}$, the total number of seeks to retrieve this clip is:

$$E[S] = \sum_{i=0}^{N_T} \lfloor \frac{(F_{S_i} - 1) \ mod \ F_B}{F_B} (\lceil \frac{F_{S_i} - 1}{F_B} \rceil + 1) + (1 - \frac{(F_{S_i} - 1) \ mod \ F_B}{F_B})(\lfloor \frac{F_{S_i} - 1}{F_B} \rfloor + 1) \rfloor$$

Hence, the total time to read this clip is:

$$T_T = E[S]T_S + F_T T_F$$

where T_S is the average seek time and T_F is the time to read a frame.

Appendix B: Evaluation of Data Placement Techniques with a Multi-zone Disk

Based on simulations, we quantified the average block retrieval time with multiple cases of access frequencies and different block placement approaches. In our simulations, we used two different disk models (their zone information is listed in Figure 6 and three media types (M_1=10 Mb/s, M_2=7.5 Mb/s, and M_3=5 Mb/s) with three block sizes (B_1 = 1.25MBytes, B_2 = 0.94MBytes, and B_3 = 0.63MBytes) assuming a fixed time period (T_p = 1 sec). We assumed that each media type

	Placement	$\bar{s}$ (msec)	σ_s	$\bar{\omega}$ (msec)					
				$\lambda = 3.5$	$\lambda = 4.0$	$\lambda = 4.5$	$\lambda = 5.0$	$\lambda = 5.5$	$\lambda = 6.0$
Case 1	RP	154.8	53.9	264.5	314.8	412.7	706.1	-	-
	MVP	153.6	23.2	246.7	281.1	334.8	432.2	657.3	-
	MTP	153.7	23.1	247.0	281.4	335.4	433.2	659.5	-
Case 2	RP	137.9	49.5	214.3	240.7	281.3	355.2	543.1	-
	MVP	145.9	21.7	224.6	251.7	290.6	353.7	472.5	771.3
	MTP	132.1	58.3	204.2	228.8	266.4	336.3	531.5	-
Case 3	RP	172.3	54.8	332.0	432.7	731.4	-	-	-
	MVP	161.7	22.9	271.6	317.1	395.3	563.3	-	-
	MTP	161.8	22.8	271.8	317.4	395.8	564.3	-	-

Table 5: Comparison (Quantum Atlas XP32150 disk)

	Placement	$\bar{s}$ (msec)	σ_s	$\bar{\omega}$ (msec)					
				$\lambda = 5.5$	$\lambda = 6.0$	$\lambda = 6.5$	$\lambda = 7.0$	$\lambda = 7.5$	$\lambda = 8.0$
Case 1	RP	117.3	45.1	272.2	366.7	794.1	-	-	-
	MVP	115.9	20.1	223.1	257.4	311.7	410.4	631.0	-
	MTP	116.0	20.0	223.2	257.6	312.0	410.9	632.0	-
Case 2	RP	102.5	40.9	187.3	213.3	255.7	338.2	979.7	-
	MVP	109.5	18.3	195.5	218.6	251.7	303.3	394.2	585.9
	MTP	101.0	41.1	181.0	203.8	238.5	301.5	500.1	-
Case 3	RP	132.9	46.1	476.1	-	-	-	-	-
	MVP	123.2	20.2	262.5	319.7	432.4	758.9	-	-
	MTP	123.2	20.2	262.6	319.9	432.7	759.3	-	-

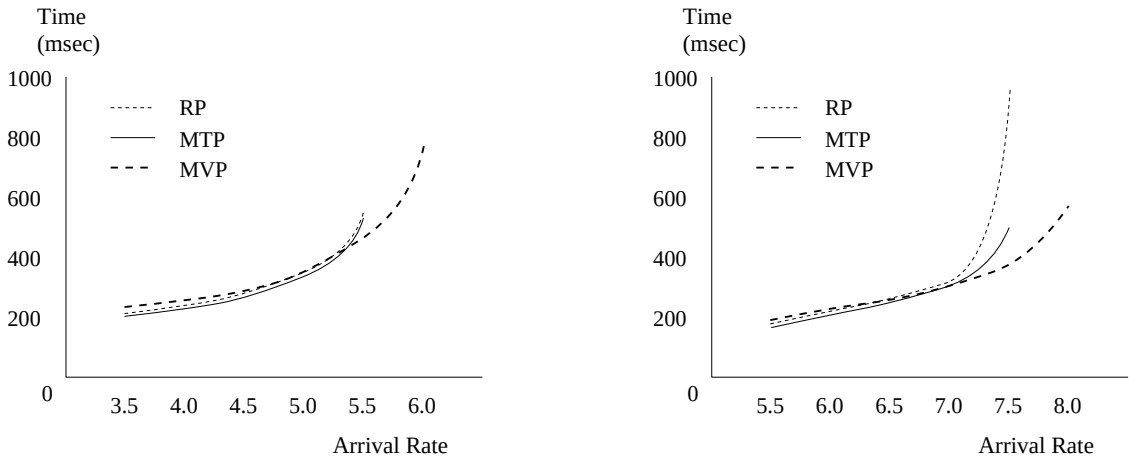
Table 6: Comparison (Seagate Barracuda 4LP, ST32171W disk)

consists of the same number of objects and varied the frequency of access to the objects as follows:

- Case 1: $M_1 = 0.33$, $M_2 = 0.33$, $M_3 = 0.33$: all objects have the same access frequency
- Case 2: $M_1 = 0.2$, $M_2 = 0.3$, $M_3 = 0.5$: objects with higher bandwidth requirements have lower access frequencies
- Case 3: $M_1 = 0.5$, $M_2 = 0.3$, $M_3 = 0.2$: objects with higher bandwidth requirements have higher access frequencies

We used an open simulation model for this study. The arrival rate of block requests follows the Poisson distribution. For each case, we quantified the average service time ($\bar{s}$) and the average retrieval time ($\bar{\omega}$) for the three alternative placement techniques.

Tables 5 and 6 show the simulation results with different case studies and demonstrate the impact of data placement on the average service time, the variance of service time, and the average retrieval



(a) Quantum Atlas XP32150 (b) Seagate Barracuda ST32171W

Figure 15: Retrieval time with Case 2

Placement	$\bar{s}$	σ_s	$\bar{\omega}$	$P[\omega > T_p]$	$P[\omega > 2T_p]$	$P[\omega > 3T_p]$	$P[\omega > 4T_p]$
RP	137.9	49.5	281.3	0.0187	0.00057	0	0
MVP	145.9	21.7	290.6	0.0095	0	0	0
MTP	132.1	58.3	266.4	0.0186	0.00081	0	0

Table 7: Retrieval time distribution with Case 2 ($\lambda = 4.5$) (Quantum disk)

time while varying the arrival rate of block requests[8]. In all simulations, MTP minimizes the average service time while MVP minimizes the variance in service time. With Cases 1 and 3, MVP and MTP are almost identical in placing data blocks. Consequently, their results are almost identical and superior to RP. With Case 2, MTP and RP produces better average retrieval time than MVP when the arrival rate is low. However, while we increase the arrival rate, the average retrieval times of MTP and RP increase faster than that of MVP and finally MVP produces the best result when the arrival rate is high (Figure 15). This is because the variance of the service time becomes more important in determining the average retrieval time when the system load becomes higher. Moreover, even though MVP has a longer average retrieval time than both MTP and RP with a low system load (less than 10% longer), the distribution of the retrieval time is more desirable than both of MTP and RP (two examples in Tables 7 and 8) minimizing hiccup probability (see Section 3). MVP is superior to both MTP and RP in all cases when the arrival rate is high.

[8]In these tables, a '-' denotes the occurrence of hiccups: a retrieval time larger than the duration of a time period ($T_p = 1$ sec in our simulations).

Placement	$\bar{s}$	σ_s	$\bar{\omega}$	$P[\omega > T_p]$	$P[\omega > 2T_p]$	$P[\omega > 3T_p]$	$P[\omega > 4T_p]$
RP	137.9	49.5	355.2	0.0516	0.00858	0.00191	0.000096
MVP	145.9	21.7	353.7	0.0329	0.00129	0.000055	0
MTP	132.1	58.3	336.3	0.0499	0.0088	0.0033	0.00071

Table 8: Retrieval time distribution with Case 2 ($\lambda = 5.0$) (Quantum disk)

One drawback of MTP is that it requires accurate access frequency of blocks. It is not always possible to estimate the access frequency of a block at the time of data block placing. Moreover, the access frequency of blocks may evolve over time. In such an environment, MTP can be the worst choice of block placement when access frequency of blocks changes. For example, suppose that access frequency changes to Case 1 after blocks are assigned to the zones based on the access frequency of Case 2. Then MTP produces the average retrieval time of 330.6 msec ($\lambda = 4.0$) with Quantum disk which is longer than that of both MVP and RP (see Case 1 in Table 5). Changing access frequency from Case 2 to Case 3 produces the same trend.