Reducing Initial Latency in Multimedia Storage Systems

Edward Chang and Hector Garcia-Molina
Department of Computer Science
Stanford University
{echang, hector}@cs.stanford.edu

Abstract

A multimedia server delivers presentations (such as videos, movies, games), providing high bandwidth and continuous real-time delivery. In this paper we present techniques for reducing the initial latency of presentations, that is, for reducing the time between the arrival of a request and the start of the presentation. Traditionally, initial latency has not received much attention. This is because one major application of multimedia servers is "movies on demand" where a delay of a few minutes before a new multi-hour movie starts is acceptable. However, latency reduction becomes important in interactive applications such as video games and browsing multimedia documents. Various latency reduction schemes are examined and proposed, and their performance compared. Analysis shows that on-disk partial data replication technique can significantly reduce (almost eliminate in some cases) initial latency without adversely affecting throughput. The on-disk partial data replication scheme that we propose proves far more cost effective than any previous attempts to reduce initial latency.
Keywords: multimedia, data placement, data replication.

1 Introduction

The delivery of multimedia presentations poses many challenges, from data retrieval [7, 10] to real-time network transmission [11, 14, 15] to user interface design. In this paper we focus on the data retrieval aspect.

An effective multimedia storage system must retrieve data at very high rates and in a *continuous* fashion. That is, each multimedia presentation (e.g., movie, video, game) can be viewed as a sequence of data segments, where each segment must arrive at the display device at or before the time of display. To achieve this, the storage system must make each segment available to the network at the appropriate times, even if the disk arms are busy servicing other requests.

In addition to high bandwidth and continuous delivery, our goal is to also minimize the *initial latency*, the time between the arrival of a new request and the time when its first data segment becomes available in the server's memory. Traditionally, initial latency has not received much attention. We believe that this is because one major application of multimedia servers is "movies on demand" where a delay of a few minutes before a new multi-hour movie starts is acceptable. Indeed, some of the proposed servers have either ignored the latency issue entirely [6, 10] or acknowledged that they involve initial latencies on the order of minutes [8].

However, we believe that there are applications for which high latencies are not acceptable. For example, consider a video game in which at each step the player's actions determine what short video to play next. Clearly, we do not want the player to have to wait a significant amount of time before each video scene starts. Another application that requires low latency is hypermedia documents, as found in the World Wide Web or a Digital Library. Here a user may examine a web page that contains links to a variety of other pages, some of which may be multimedia presentations. In this case as well, users do not want a significant delay from when they click a link to the time the presentation starts. We could try to pre-fetch all the possible videos that might be selected, but doing so would significantly increase the server load and memory requirements.

In reducing the initial latencies, it is important neither to sacrifice throughput (i.e., reduce the number of concurrent requests that can be serviced) nor to violate the continuous delivery constraint. Our investigation into data allocation begins with the techniques of [8, 18, 19], which already achieve high bandwidth and continuous delivery for supporting multiple concurrent data streams. Our work shows that by changing the data placement and disk scheduling policies, we can reduce initial latency drastically without adversely affecting throughput. In addition, this article proposes a novel on-disk partial data replication scheme that requires only small percentage of disk space overhead to eliminate initial latency. Initial segments are replicated on a separate disk without increasing memory requirements or decreasing throughput. This on-disk replication approach proves far more cost effective than the in-memory replication scheme sometimes used to reduce initial latency.

Trying to reduce initial latencies introduces interesting tradeoffs between latency, throughput, disk speeds, and and available main memory. It is important to understand the inter-relationships between all these factors: for example, adding memory to a server can increase throughput or reduce latencies. This paper investigates such tradeoffs through an analytic model and presents some guidelines for the choice of appropriate parameters.

In this study we assume that M disks are used to store the movies or videos. Each movie is placed within a single disk, but a disk may hold multiple movies. When a request arrives, only one disk is involved in servicing it. Thus, playback of a movie is independent of the other $M-1$ disks, and we can focus our analysis on a single disk. The available system memory is used to support access to all M disks. Since we will be analyzing the one-disk case, all memory sizes in this paper refer to the memory needed to support IO's to one disk. The total system memory requirement is thus M times the values we report. Similarly, the throughput values are M times the reported values. For a discussion of other data placement methods with M disks, interested readers can consult references [2, 4, 5, 6, 13].

The rest of this paper is organized as follows. Section 2 describes the allocation strategies and performance model that we use as a starting point. Section 3 examines some allocation and disk scheduling techniques. Section 4 proposes our on-disk partial data replication scheme for reducing initial latency. Finally, Section 5 compares and evaluates the techniques presented in Section 3 and 4, by analyzing their use in a case study.

2 Constrained and Partitioned Allocation

This section summarizes the existing storage allocation policies we use as our starting point. To assist the reader, Table 1 gives the main parameters used to describe these policies, as well as ours. The first portion of Table 1 lists performance-related parameters, including throughput and initial latency. The second portion describes the physical and derived characteristics of the hardware, including memory and disks. The last part summarizes the notation we use to refer to presentations (e.g., movies, videos, games) and their segments.

Seek time is the most critical factor over which we have some control (through segment allocation). To reduce this factor, early studies [3, 7, 18] take advantage of the sequential access nature of a presentation and propose to limit the distance between their segments on disks. This placement policy is referred to as *constrained allocation*. For example, say a presentation X_i is divided into n segments, ($X_{i,1}$, $X_{i,2}$, $X_{i,3}$..., $X_{i,n}$), for storage on disk. Constrained allocation places each pair of consecutive segments within d tracks, or

$$|Track(X_{i,j+1}) - Track(X_{i,j})| \leq d$$

Parameter	Description
N	Throughput
$T_{Latency}$	Initial latency to access a media
Mem	Total size of memory, bytes
DR	Data display rate, bytes/s
TR	Disk transfer rate, bytes/s
S	Segment size, bytes
R	Number of disk partitions, or regions
CYL	Number of cylinders, or tracks per surface
M	Number of disks
α	Seek time parameter, constant part
β	Seek time parameter, distance dependent part
d	Seek distance, number of cylinders
$\gamma(d)$	A concave function that computes seek overhead given d
$T_{RegionSeek}$	Total seek time in a disk region, milliseconds
$T_{RegionTR}$	Total transfer time in a disk region, seconds
T	Period, the time to service a round of requests
X_i	i^{th} presentation
$X_{i,j}$	Segment j of presentation i
K	Number of presentations

Table 1: Parameters and Their Description

Constrained allocation reduces the seek distance from what in the worst case can be the radius of the disk, to a maximum of d tracks.

Constrained allocation for the segments of a presentation does not limit seek time with concurrent presentations. To illustrate, assume there are two presentations on disk, X_i and X_{i+1}. The distance between any two given segments — one from X_i and the other from X_{i+1} — could be as large as the number of tracks on the disk. An interleaved request, say for $X_{i+1,1}$ (first segment of presentation X_{i+1}), scheduled between the retrieval of segment $X_{i,j}$ and $X_{i,j+1}$ could therefore generate a seek of more than d tracks, making it hard to meet the continuous display constraint.

2.1 Scheme Bidirectional

To remedy the shortcomings of constrained allocation, the study of [8] introduces a combined allocation scheme that employs both constrained and unconstrained allocation policies. The scheme partitions a disk into R equal regions. Segments of all presentations are assigned in a zigzag manner that follows the disk head movement as the head sweeps the disk. Since the regions divide the disk evenly, the distance between two contiguous segments is constrained by a maximum of two regions (assuming that at worst one segment is at the head of one region and another is at the tail of the next region), or $d = 2/R$ of the disk radius. Within a region, segments of different presentations are placed with no constraints.

Figure 1 illustrates this scheme with a disk with 4 regions and three videos. The first segment of each video is placed in region 1. Subsequent segments are placed in regions 2-4, and then in the reverse order from 4 to 1. Additional segments (not shown) would repeat the pattern. Since the data layout is bidirectional on disk, we refer to this scheme *Bidirectional*.

To display objects, the disk head moves like an elevator from the outermost region (region 1) to the innermost (region 4). After reaching the innermost region, it swings outward to the outermost region. This procedure repeats itself until there are no more segments to retrieve. At each region, the disk head picks up all requested segments in the region. For instance, if presentation X_1 is requested, the disk head picks up segment $X_{1,1}$ as it services region 1. If X_1 and X_2 are both requested at the same time, then two segments, $X_{1,1}$ and $X_{2,1}$ are retrieved from region 1; and $X_{1,i}$ and $X_{2,i}$ are retrieved from the subsequent regions. At both ends of the disk, the regions are serviced twice before the disk head moves inward or outward. For instance, the disk head when on region 4 the first time retrieves segment $X_{i,4}$, the next time period it stays in the same region to retrieve segment $X_{i,5}$.

A new request for a currently displayed or new presentation can arrive any time. For example, say a request for presentation X_3 arrives while the disk head is servicing the second region and moving toward the third region. The new request can be serviced provided the disk bandwidth is adequate. However, the new request must wait for the disk head to reach region 4, swing back to region 1, and reverse its direction to pick up segment $X_{3,1}$.

This scheme guarantees each concurrent presentation a fraction of the disk bandwidth while at the same time limiting the length of seeks d to $2/R$ of the disk radius. However, new presentations must wait until the disk head sweeps to the starting position. In our example above, a new request could wait for the disk head to travel up to 8 regions. In general, the maximum initial latency is the time for the disk head to service $2 \times R$ regions [1]. The simulation reported in [8] shows that this delay can be large, especially when the system is busy. For

[1] In the degenerate case $R = 1$, the delay need not be multiplied by 2. The evaluation in Section 5 takes this into account.

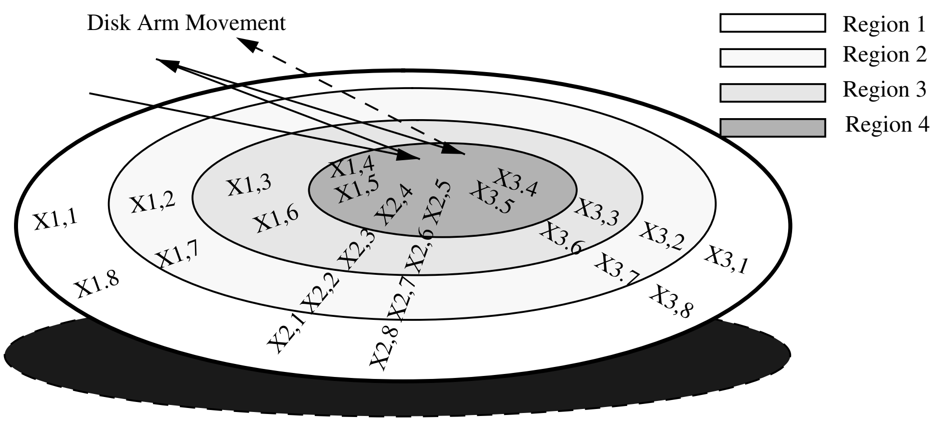


Figure 1: Bidirectional Data Placement

example, with 33 concurrent presentations and 24 regions, the initial latency can be up to 63.2 seconds. As we argued in Section 1, such delays may be unacceptable in many applications with unpredictable requests for short presentations.

In Section 3 and 4 we study a variety of schemes to reduce this latency without adversely affecting the number of concurrent presentations. However, before that, we briefly derive expressions for the throughput and latency under the initial strategy of this section. These expressions will then be contrasted to those derived for the later schemes.

2.2 Performance Expressions

This Bidirectional scheme employs a FIFO disk arm scheduling policy that services the requests in a fixed order from region (or service period) to region. For the memory management policy, we allocate for each stream a *private* buffer, and do not consider buffer sharing in this paper. For issues related to buffer sharing and its performance evaluation, please consult reference [4].

To derive the maximum throughput, N, we need to select the segment size S and the number of regions R that maximize N without violating two constraints: display continuity and memory availability. To satisfy the continuity constraint, we need to use buffering and read-ahead (or prefetching) to ensure that for each presentation there is always a segment that is ready to be transmitted as soon as the last one is consumed. Let's denote the time between two consecutive IOs for a request as T. The continuity constraint requires that each IO must retrieve a large enough segment S to sustain T time of display at a rate of DR, or

$$T \leq S/DR. \tag{1}$$

On the other hand, to service N requests simultaneously the disk head must be able to read the next segment as well as other segments (for other presentations) in T when it visits a region. To read in N segments, the total time is

$$T = T_{RegionTR} + T_{RegionSeek}, \tag{2}$$

where $T_{RegionTR}$ is the time to transfer all requested segments in a region, and $T_{RegionSeek}$ is the total seek overhead for segments fetched in a region. With N simultaneous displays, $T_{RegionTR}$ is N times the time to transfer a segment, or

$$T_{RegionTR} = N \times (S/TR).$$

The worst case seek distance in a region, with non-constrained allocation in the region, is the width of the region or $d = CYL/R$ tracks [2]. Using the seek function $\gamma(d)$ that computes seek overhead with a given seek distance (in this case $d = CYL/R$), we can express the total worst seek overhead for N segments as

$$T_{RegionSeek} = N \times \gamma(\frac{CYL}{R}).$$

Substituting $T_{RegionTR}$ and $T_{RegionSeek}$ into Equation 2 yields

$$T = N \times (\frac{S}{TR} + \gamma(\frac{CYL}{R})). \tag{3}$$

Finally, substituting T into Inequality 1, we obtain the continuity constraint as

$$N \times (\frac{S}{TR} + \gamma(\frac{CYL}{R})) \leq S/DR. \tag{4}$$

The second constraint specifies that the memory required to display N concurrent presentations be less than or equal to the available memory, Mem. At first glance, S bytes seems to be sufficient for each of the N presentations. This would be true if consecutive IOs for a

[2] The seek for servicing the first request in a region could travel a distance that is up to two regions, or $d = \frac{2 \times CYL}{R}$ tracks. To make the expressions simpler, we do not include this special case in the derivations. However, the case study in Section 5 takes this special case into consideration.

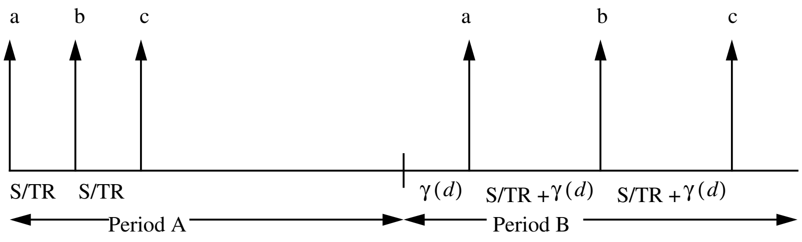


Figure 2: Seek Overhead Variability

particular request were separated by exactly T time. However, the time that separates two IOs of a request can vary from period to period because of the unpredictable seek overhead.

To illustrate, Figure 2 shows a variable seek overhead example where three requests a, b, and c are scheduled in the same order in two periods, A and B. While all IOs in period A have no seek overhead, the IOs in period B suffer from the worst seek overhead $\gamma(CYL/R)$. In period A, since the seek overhead is zero, the IOs for request a, b, and c can start at time 0, S/TR, and $2S/TR$ respectively (S/TR is the time to transfer a segment for a request). In period B, on the other hand, the IOs for request a, b, and c cannot start until time $\gamma(CYL/R)$, $S/TR+2\times\gamma(CYL/R)$, and $2S/TR+3\times\gamma(CYL/R)$ into the period. The longest time between IOs for request a is $T+\gamma(CYL/R)$, for request b it is $T+2\times\gamma(CYL/R)$, and for request c it is $T+3\times\gamma(CYL/R)$. In general, the longest time between two consecutive IOs of the i^{th} serviced request is $T+i\times\gamma(CYL/R)$. Notice that this time is larger than T, and hence S amount of buffered data per request is not sufficient to satisfy the continuity constraint.

To remedy this, we need a cushion buffer of size $DR\times i\times\gamma(CYL/R)$ for the i^{th} request. If IO's are separated by the maximal amount, this cushion provides the extra data to sustain continuous display. On the other hand, if IOs are separated by less than T, this cushion buffer serves as the extra space to store the data that is read in early. For N requests, the total cushion buffer required for scheme Bidirectional is the sum of the individual ones: $DR\times\sum_{i=1}^{N} i\times\gamma(CYL/R)$. Adding these cushion buffers to $N\times S$, we have the memory constraint as:

$$N\times S+(DR\times\sum_{i=1}^{N} i\times\gamma(\frac{CYL}{R}))\leq Mem. \tag{5}$$

Finally, we derive the *initial latency* for scheme Bidirectional. We define initial latency as the worst case time between the arrival of a *single* new request (when the system is unsaturated) and the time when its first data segment becomes available in the server's memory. In computing the initial latency we do not take into account any time spent by a request waiting because the system is saturated, as this time could be unbounded no matter what scheduling policy is in place. In other words, our focus is on evaluating the worst initial delay when both disk bandwidth and memory resources are available to service a newly arrived request. Also, we do not consider the case of a burst of new requests arriving.

As mentioned in Section 2.1, the worst case initial latency for the R region disk partitioning scheme is the time to service $2 \times R$ regions ($R \geq 2$). Since the time to service a region is T, we can write this worst delay as $2 \times R \times T$. In addition, because of the variability of seek overhead, we must delay the playback startup time as if the worst possible seek overhead has occurred in the first period. This extra delay ensures that the next IO is at most T away. Since for the i^{th} request this extra delay is $i \times \gamma(CYL/R)$, for the last (out of N) request, this extra delay can be as long as $N \times \gamma(CYL/R)$. Adding this worst possible extra delay to $2 \times R \times T$, and substituting T with the righ hand side of Equation 3, we have the equation for the worst initial latency as:

$$T_{Latency} = 2 \times R \times N \times (\frac{S}{TR} + \gamma(\frac{CYL}{R})) + N \times \gamma(\frac{CYL}{R}). \quad (6)$$

Parameters DR, TR, and Mem are given by the hardware configuration or by the requests. For a given R, we can solve N from Inequalities 1 and 5. The largest N that satisfies both inequalities is the storage system's throughput. Finally, we can compute $T_{Latency}$ once N is known.

3 Allocation Schemes

This section presents data placement and disk scheduling policies that attain low initial latency while maintaining high throughput. We examine three such policies:

1. Unidirectional data layout (Scheme *Unidirectional*),

2. Unidirectional, plus sequential access of segments in a region (Scheme *Sequential*), and

3. Group Sweeping Scheme (*GSS*).

After presenting each scheme, we will derive expressions for $T_{Latency}$.

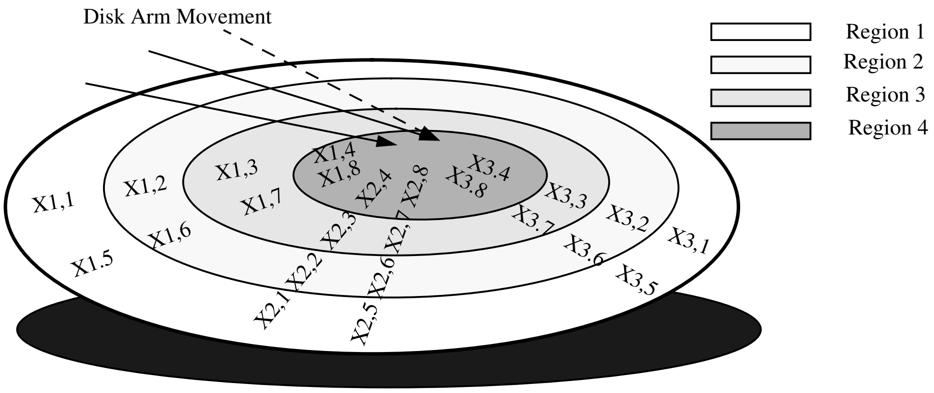


Figure 3: Unidirectional Data Placement

3.1 Scheme Unidirectional

Instead of placing the segments in an elevator-like zigzag manner as proposed in [8] (see Section 2), references [9, 12] suggest a unidirectional layout. Figure 3 illustrates this layout on our three video, 4 region example. As shown, the segments of a given presentation are placed from the outermost region to the innermost. When the innermost region is reached, the next segment is placed in the outermost region again. This scheme modifies the disk scheduling policy slightly: The disk head retrieves data only in the same direction as the data placement. After a unidirectional sweep, the disk head returns to the other end to start the next round of retrieval.

The maximum initial latency is reduced from the time to service $2 \times R$ regions, to the time to service R regions plus the cost of resetting the disk head to the first region. For modern disks, this disk head reset from one end of the disk to another is typically less than 20 milliseconds. Compared with the magnitude of the initial latency that we aim to improve, this 20 millisecond cost is negligible.

Since the maximum initial latency is roughly half of that for our first scheme, our latency expression is simply the first part of Equation 6 divided by 2 added to by the part that deals with seek overhead variability:

$$T_{Latency} = R \times N \times (\frac{S}{TR} + \gamma(\frac{CYL}{R})) + N \times \gamma(\frac{CYL}{R}). \tag{7}$$

This represents a reduction of about 50% in initial latency, with essentially no change to the cost and the maximum throughput achievable.

3.2 Scheme Sequential

So far we have assumed a non-constrained allocation of segments within a region, with which each seek distance can be as large as the width of the region. However, forcing the disk head to pick up segments in their on-disk sequential order (as opposed to some fixed presentation order) cuts down the expected seek distance to $\frac{CYL}{R \times N}$, where N is the number of concurrent presentations. Thus, with this scheme, the disk head scans through a region once, picking up segments for the N presentations.

However, this elevator disk scheduling policy has a potential drawback. When segments within a region are read in a fixed presentation order (Section 2), we know that between the time segment $X_{i,j}$ is read, and its continuation segment $X_{i,j+1}$ is read, we will do at most $N-1$ accesses to other segments. With sequential access, on the other hand, segments may be read in different order within each region. In the worst case, segment $X_{i,j}$ could be read first in one region, while the next segment $X_{i,j+1}$ could be read last in its region. This means that we could have $2 \times (N-1)$ other accesses in between. However, if $X_{i,j}$ is physically last in its region, and $X_{i,j+1}$ is first, then there could be no other accesses between these two X_i reads.

To insulate the playback process from this variability in inter-segment times, we proceed as follows. Say a new presentation X_i needs to be started and its first segment $X_{i,1}$ is in region k. When region k is scanned, we read $X_{i,1}$ into a first memory buffer (size S) but do not immediately start playback. When region k is fully scanned, we start playback of $X_{i,1}$. If $X_{i,2}$ occurs at the beginning of region $k+1$, then we need to read it into a *cushion* buffer because the first buffer is still in use. At the other extreme, if $X_{i,2}$ appears at the end of $k+1$, then we do not need the cushion buffer at all, since the new segment arrives in memory as playback of the first one completes. In any case, by the time the scan of region $k+1$ completes, we have a full buffer of X_i ready for transmission, and we are ready to repeat the process. Thus, playback takes place as in the earlier scenarios, with the exception of the cushion buffers that are needed to handle the variable time between accesses to consecutive segments. This means that our memory constraint is

$$2 \times N \times S \leq Mem. \tag{8}$$

The reduced seek times for the sequential policy leads to a smaller initial latency. However, we also need to take into account the startup delay to fill up a buffer with $X_{i,1}$ and wait for the region scan to complete. In the worst case, the request for presentation X_i arrives just after the disk head has passed over $X_{i,1}$ and while this segment is at the very beginning of region k. In this case, we need to wait for R region scans to return to the beginning of region k, plus the full scan of region k before we start playback. Hence, the equation for latency is similar to Equation 7 except that the R factor is replaced by $R+1$. In addition, the inter-segment seek distance d is $\frac{CYL}{R \times N}$. Within a region, some seeks could cover more than $CYL/(R \times N)$ cylinders, but then some of the other seeks would have to be shorter. Given that the seek time function is concave, the worst case occurs if the total seek distance that must be covered within the region, CYL/R, is uniformly split among the N seeks. (Please also see [17] for explanation.) Therefore, the worst seek overhead is expressed as

$$T_{Latency} = (R+1) \times N \times (\frac{S}{TR} + \gamma(\frac{CYL}{R \times N})). \tag{9}$$

To compute N for a given R, scheme Sequential follows the same procedure described at the end of Section 2. For a given R, we can solve N with Inequality 1 and 8. Notice that the seek distance used to compute seek overhead is replaced with $d = \frac{CYL}{N \times R}$. The largest N that satisfies both inequalities is scheme Sequential's throughput. Once N is known, we can compute $T_{Latency}$ using Equation 9.

3.3 Group Sweeping Scheme

So far in this section, we have discussed disk scheduling policies at two extremes. Scheme Unidirectional has a longer worst seek distance, but its fixed order scheduling requires a smaller cushion buffer. On the other hand, scheme Sequential amortizes seek overhead over N requests and has a smaller seek overhead. However, its out of order scheduling requires a larger cushion buffer.

The Group Sweeping Scheme (GSS) proposed in [19] achieves an optimal balance between these two extremes. GSS divides N requests into G groups, each servicing N/G requests. Within each of these G groups, GSS uses scheme Sequential to amortize seek overhead among the N/G requests. Between groups, GSS services the groups in a fixed order to reduce the cushion buffer requirements. Notice that GSS is scheme Sequential when $G = 1$. When $G = N$ (one request per group), GSS services one request after another in a fixed order from period to period, and hence converges to scheme Unidirectional. The study of [19] shows that the optimal

G value tends to be small when the disk load is high (large N). Our study in [4] shows that the optimal G value is also sensitive to the memory management policies employeed.

Scheme GSS does not help reduce the worst case initial latency. This is because in the worst case all groups minus one are "full" servicing N/G requests, and the one group that has capacity for a newly arriving request has just been missed. Thus, playback is delayed for about the G disk scans it takes to service all groups. Within this worst case scenario, the best case is if $G = 1$, in which the initial latency is the same as with scheme Sequential. Since GSS cannot further reduce the worst initial latency, we do not include it in our subsequent evaluation and analysis.

4 Replication Schemes

Initial latency can be entirely eliminated by replicating an initial portion of a presentation in memory, but the memory cost is high. In this section we show the cost of the in-memory caching approach, and propose a more cost effective on-disk replication scheme. Replication schemes can work with any data placement and disk scheduling schemes we have discussed so far; however, in this section we only use scheme Sequential to illustrate.

4.1 In-Memory Replication

When a request comes in, one approach is to play the replicated copy in memory until the disk head reaches the first non-replicated segment on disk. At that point, playback continues from disk.

There are two memory overheads associated with this in-memory replication scheme: the *replicas* overhead, and the *playback* overhead. First, the amount of memory needed to hold the replicas is the number of presentations K times the data needed for covering the worst latency of $T'_{Latency}$ seconds. Variable K represents the total number of presentations stored, not just those that are currently being played. Notice that we use a "prime" symbol to refer to the latency of the original storage system without replication; the new latency will be zero. Thus, the memory needed for replicas is

$$K \times DR \times T'_{Latency}. \tag{10}$$

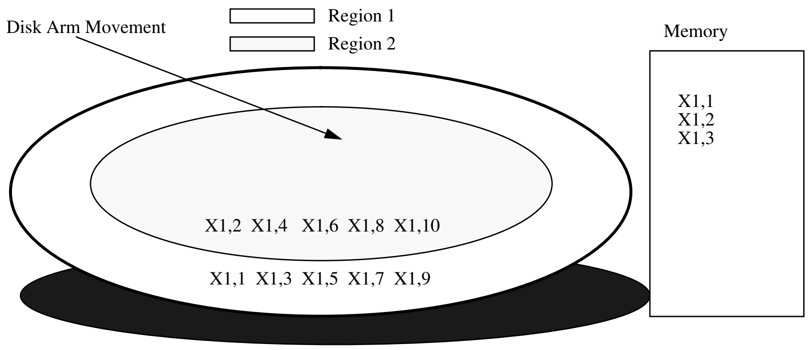


Figure 4: In-memory Replication Example

Let's use the example in Figure 4 to illustrate this memory overhead. Figure 4 shows that a disk is partitioned into two regions and the segments of presentation X_1 are placed in a round-robin fashion from the outermost region (the first region) to the innermost one (the second region). Employing scheme Sequential, the disk head services the first and second region in each inward sweep, then resets to the outermost to repeat this cycle. According to Equation 9, the worst initial latency $T'_{Latency}$ of scheme Sequential is the time to service $R+1$ regions, in this case, three regions. Therefore, the in-memory replica must be able to cover three periods of playback. Since we know that each segment can sustain one period of playback, we need to replicate the first three segments of each presentation in memory. For presentation X_1, for example, we need to replicate segments $X_{1,1}$, $X_{1,2}$, and $X_{1,3}$.

The playback overhead is the memory required during media playback. Let's continue with the example in Figure 4. Since the first three segments of X_1 are in memory, playback can start as soon as a new request arrives. The worst case happens when playback starts when the disk head just passed $X_{1,1}$ in the first region. The playback must start from the in-memory replica, and the replica can sustain the playback for three periods. During this time we have to read $X_{1,4}$, the first disk segment we need. This has to be read during the second period, while $X_{1,2}$ is played back from memory. Thus, $X_{1,4}$ must be held in memory in the worst case for two periods. Similarly, $X_{1,5}$ must be read and held for one period. Furthermore, we may have to read $X_{1,6}$ just as we start playback of $X_{1,4}$, so at that instant we have up to three full segments in memory. Therefore, the worst playback memory requirement in this example is $3 \times S$. It is easy to generalize this example to any values of R and show that the playback overhead is

$$(R+1) \times S.$$

Both memory requirements we have described are very sensitive to R. First, the size of a replica depends on $T'_{Latency}$, and $T'_{Latency}$ in turn depends on R as shown in Equation 9. Second, the memory needed during playback is $(R+1) \times S$, also proportional to R. On the other hand, we show in Section 5 that throughput is not very sensitive to R. Therefore, $R = 1$ is the most reasonable value to reduce memory use for the in-memory replication scheme. In addition to using $R = 1$, we can also reduce the size of the in-memory replicas by decreasing $T'_{Latency}$. Still, the amount of memory use can be large. For instance, if DR is 1.5 Mbps (a typical value), our system stores $K = 100$ presentations, and $T'_{Latency} = 5$ seconds for scheme Sequential, we need nearly 100 MBytes (750 Mbits), just for the replica overhead! This leads to our on-disk replication proposal.

4.2 On-Disk Replication

To reduce the memory cost, we propose an on-disk segment replication scheme called *Replicated* hereafter. Scheme Replicated stores the replica of the initial segments of each presentation on a separate disk, called the *replica disk*. For simplicity, this section presents how scheme Replicated works with scheme Sequential only, and assumes $R = 1$.

Figure 5 depicts a sample configuration where one replica disk stores replicas for M main disks. As before, we assume that the main copy of each presentation is stored on a single disk, and hence the main disks service requests independently of each other. The main disks and the replica disk share the same memory through a high speed bus. Briefly, scheme Replicate operates as follows:

1. When a new request arrives, buffer space, and disk bandwidth on the presentation's main disk are allocated.

2. The request is dispatched to the replica disk to retrieve the initial segments. The playback starts as soon as the replica is available in memory.

3. The request's playback resumes from the main disk.

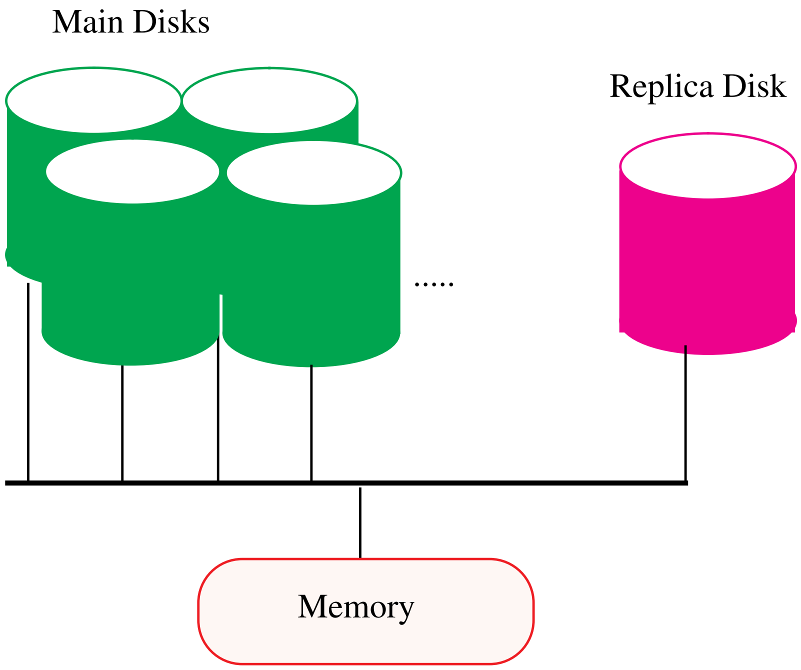


Figure 5: A Replica Disk Configuration

4.2.1 Replica Size

The size of a replica must be large enough to sustain the playback during the worst possible initial latency on the main disk. That is, the size of a replica should be

$$DR \times T'_{Latency}.$$

For scheme Sequential, the worst initial delay occurs when a request for a presentation arrives just after the disk head has passed the first segment of the presentation on the main disk. Since the playback cannot start until the end of the period in which the first segment is retrieved, this initial delay can be as long as two periods: one period to finish the current disk sweep, and another one to complete the sweep in which the first segment is retrieved. To cover this worst possible initial latency, the size of the replica must be adequate to sustain two periods of playback, or $DR \times 2T$. Since $S = DR \times T$, the size of each replica for scheme Sequential is

$$2 \times S.$$

4.2.2 Memory Requirement

Scheme Replicated may also incur playback memory overhead. If we read the full $2 \times S$ replica, in the worst case the third segment that is read in from the main disk may be read in when none of the $2 \times S$ replica has been consumed. This makes the memory requirement $3 \times S$ rather than $2 \times S$. This additional S amount of memory per presentation remains in memory for the duration of playback.

However, if presentations are stored consecutively on disk, and if we read the replica carefully, we can eliminate the need for this playback overhead. The key is to read just enough data from the replica to sustain playback until the main disk can take over. Let us assume that the IO from the replica disk is scheduled to complete at time t_1, and that at this time playback starts. Let us also assume that the current disk scan on the main disk is scheduled to end at time t_2. (Note that periods are of fixed length, regardless of the number of requests currently in service.) In this case, we read the first segment of the replica plus $(t_2 - t_1) \times DR$ of the second segment. At the period ends on the main disk, we will have exactly S data for this request, which is the normal situation. During the next scan, we playback this data and read another S worth of data from the main disk. Notice that the main disk takes over from the replica disk at a point $(t_2 - t_1) \times DR$ into the presentation.

4.2.3 Cost

For each presentation, the replica disk stores only one copy of its initial segments. For K presentations, the total disk space requirement is

$$K \times DR \times T^{'}_{Latency}.$$

The disk space required for scheme Replicated is the same as the replica overhead for in-memory replication (Section 4.1). However, since disk storage is much cheaper than memory by a factor of 50 to 100, scheme Replicated is significantly more cost effective.

4.2.4 Reduced Initial Latency

Since we do not consider a burst of requests in this study, the worst initial latency is the worst seek overhead before the first IO can commence for a newly arrived request on the replica disk.

Parameter Name	Value
Disk Capacity	2.25 GBytes
Number of cylinders, CYL	5,288
Min. Transfer Rate TR (Outermost Zone)	120 Mbps
Min. Transfer Rate TR (Innermost Zone)	75 Mbps
Max. Rotational Latency Time	8.33 milliseconds
Min. Seek Time	0.9 milliseconds
Max. Seek Time	17.0 milliseconds
$\alpha 1$	0.6 milliseconds
$\beta 1$	0.3 milliseconds
$\alpha 2$	5.75 milliseconds
$\beta 2$	0.0021 milliseconds

Table 2: Disk Parameters for the Seagate Barracuda 4LP Family

If a replica can be placed anywhere on the replica disk, the worst seek distance is the radius of the disk CYL. The worst case initial latency is therefore

$$T_{Latency} = \gamma(CYL).$$

For modern disk drives, this worst initial latency is below 30 milliseconds, negligible to human.

However, if the replica disk is not full, then we can place the replicas in the outer zones of the disk. This has two advantages. First, in modern disk drives, since the recording density is uniform across the disk, the outer zones enjoy higher capacity and faster data transfer rate [16]. Second, using only a limited number of cylinders decreases seek time. Therefore, outer zone data placement may further cut latency from the equation given above.

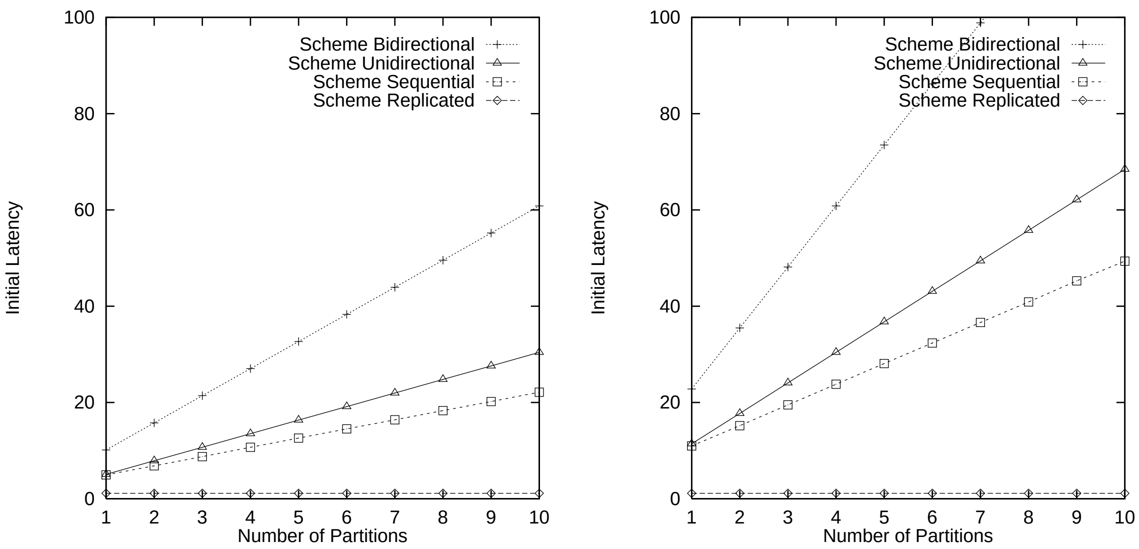
5 Case Study Analysis of the Proposed Schemes

In this section, using a case study, we evaluate and compare the performance of the proposed schemes. We assume a display rate DR of 1.5 Mbps, which is sufficient to sustain typical video playback. For our evaluation, we use the Seagate Barracuda 4LP disk parameters in Table 2. We study the worst case initial latency when both disk bandwidth and memory resources are available to service a newly arrived request.

For the seek overhead we use the following concave function [17]:

$$\gamma(d) = \alpha 1 + (\beta 1 \times \sqrt{d}) + 8.33 \text{ if } d < 400$$

$$\gamma(d) = \alpha 2 + (\beta 2 \times d) + 8.33 \text{ if } d \geq 400$$



(a) N = 40

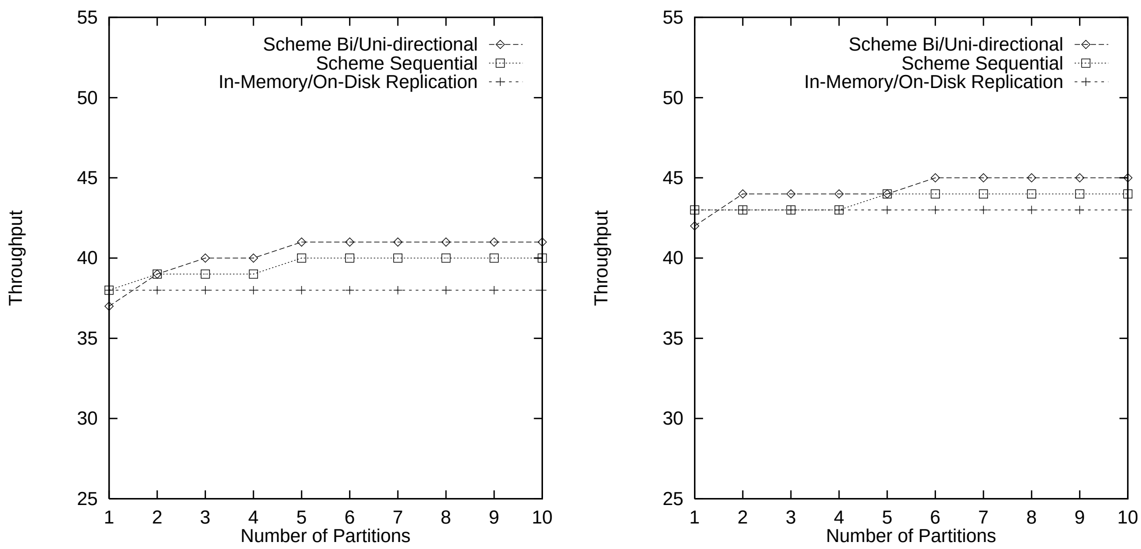
(b) N = 45

Figure 6: Initial Latency versus Throughput

Note that the seek time is proportional to the square root of the seek distance when the distance is small, and is linear to the seek distance when the distance is large. We derive the parameters for this function as follows. We first allocate 2/3 of the minimum seek time provided by the vendor (0.9 ms) as the disk arm's fixed overhead $\alpha 1$ (which includes the speedup, slowdown, and settle phases). Parameter $\beta 1$ then is the remaining portion of the minimum seek time. We then select $\alpha 2$ and $\beta 2$ so that the maximum seek time matches the manufacture's time (17 ms), and so that function γ is continuous at $d = 400$. The values obtained are given in Table 2.[3] We also add a worst case full rotational delay to the seek overhead.

Figure 6 and 7 present initial latency and maximum throughput results for five different data placement and disk head scheduling policies: Bidirectional, Unidirectional, Sequential, In-Memory Replication, and scheme Replicated. As we discussed in Section 4, both in-memory and on-disk replication schemes consider only $R = 1$. However, we plot their initial latency in Figure 6 on different R's, just to contrast the difference. (The initial latency of the in-memory replication scheme is not shown in the figure because the value is zero.)

[3] Incidentally, [17] suggests using between 200 *to* 600 cylinders to separate short and long seeks. Although we do not show it here, our results are not very sensitive to the exact value used in this range.



(a) 32 MBytes Memory

(b) 64 MBytes Memory

Figure 7: Throughput versus Number of Partitions

Our evaluations use two memory configuration: 32 and 64 MBytes. Figure 6 shows the relationship between $T_{Latency}$ and R under $N = 40$ and 45. As predicted, $T_{Latency}$ increases linearly with R under schemes Bidirectional, Unidirectional, and Sequential. Also as expected, scheme Unidirectional cuts the latency of Bidirectional by half. It is reduced further by scheme Sequential, and the replication schemes make it negligible. For the on-disk replication scheme, $T_{Latency}$ is 25.33 ms, which includes a worst case seek (17 ms) and a worst case rotational delay (8.33 ms).

Figure 7 shows the relationship between the maximum throughput N and number of partitions R. Scheme Unidirectional enjoys the same throughput as scheme Bidirectional, as it improves only initial latency. Scheme Sequential increases throughput by only one when R is small, but performs worse when R is large. This is not surprising since a large R reduces the seek overhead for scheme Bidirectional and Unidirectional more quickly. (See more discussion in Section 5.1.) Both replication schemes achieve the same throughput as scheme Sequential at $R = 1$. Since the replication schemes were only developed for $R = 1$, their throughput does not change as R varies. Overall, notice that the throughput differences are not marked.

The most important conclusion to draw from this case study is that our initial latency reduction scheme, scheme Replicated, does not have much adverse impact on the best possible achievable throughput. Taking 64 MByte memory configuration as an example, choosing $R = 1$ for scheme Replicated supports 43 streams, only three less than the best possible case 46. However, supporting 46 requests requires partitioning the disk into at least 6 regions that incurs an initial latency of 35 seconds.

One may argue that the performance comparison thus far may not be entirely fair, since both the in-memory and on-disk replication schemes use more hardware resources. An important question to answer is thus: how would Bidirectional, Unidirectional, and Sequential benefit if they were given the extra memory or the extra disk?

First, if given more memory, all schemes may be able to improve throughput. However, the throughput gained is limited since the memory requirement to support additional requests grows without bound as the disk utilization approaches its capacity (see [4] for detailed discussion). One can also see this trend from Figure 7. When memory doubles from 32 to 64 MBytes, the throughput increases by only up to 4 streams, a 10% improvement at most. Going to 128 MBytes would produce an even smaller gain.

Finally, say we added one disk to the M disks in use by schemes Bidirectional, Unidirectional, and Sequential. The presentations can now be distributed over $M + 1$ disks, slightly reducing the number of requests a disk must support, for the same total throughput. This reduces initial latency by a small amount, especially if M is large. However, since we have not added extra memory, we now have less memory available per disk, and this may degrade throughput. Therefore, we believe that the extra disk is better invested in reducing latency, as scheme Replicated does.

5.1 Additional Observations and Analysis

The case study reveals a few important facts. First, Unidirectional and Sequential policies works well with a small number of regions. Increasing the number of disk partitions accomplishes a marginal improvement in throughput, but increases initial latency dramatically. To verify this observation let us examine inequality 1. Leaving N on the left side of the inequality, we rewrite it as

$$N \leq \frac{TR}{DR} \times \frac{S}{S + (\gamma(d) \times TR)}. \tag{11}$$

Regions	*Throughput*	*Memory Requirement*	*Initial Latency*
1	*45*	92.58 MBytes	11 seconds
2	*45*	85.34 MBytes	15 seconds
4	*45*	80.32 MBytes	24 seconds
8	*45*	76.66 MBytes	41 seconds
16	*45*	74.28 MBytes	75 seconds
32	*45*	72.39 MBytes	142 seconds
64	*45*	71.06 MBytes	274 seconds

Table 3: Memory Requirement versus Partitions

Notice that in the inequality $\frac{TR}{DR}$ is the throughput cap. The major factor that influences N is $\gamma(d)$. In other words, the best way to improve N is to reduce $\gamma(d)$. Now recall that the d for scheme Sequential is $\frac{CYL}{R\times N}$, and hence the worst seek overhead $\gamma(\frac{CYL}{R\times N})$ is $\alpha + (\beta \times \sqrt{\frac{CYL}{R\times N}})$. Parameter CYL is a constant, so the $\gamma(\frac{CYL}{R\times N})$ reduction can be achieved by increasing N or R. Under scheme Bidirectional, the seek time is not reduced by the N factor, so throughput can only be improved by increasing R. With the Sequential (or Replicated) policy, seek times are also reduced by the N factor (within the radical in $\gamma(d)$), and this eliminates the need for a large R. Thus, with a large N, the magnitude left to be improved by a large R is insignificant. In short, with all policies except Bidirectional and Unidirectional, a large R does not increase throughput significantly, but it does increase latency linearly. Thus, if low initial latency is a goal, $R = 1$ is a good choice for all policies except Bidirectional and Unidirectional.

The next observation concerns the memory requirement. Intuitively, with a large R, the seek distance is shorter in a region, therefore the memory required to buffer data is smaller. In other words, a large R value conserves memory. This argument is accurate, but the question remains: how much memory can disk partitioning save? To illustrate, Table 3 lists the memory requirement and R values needed for scheme Sequential to achieve the same throughput $N =$ 45. The table also shows the initial latency for each scenario. We can see that the memory savings are moderate as R grows, but the latency growth is significant. To see why, we rewrite inequality 1 for scheme Sequential as

$$S \geq \gamma(\frac{CYL}{N \times R}) \times \frac{N \times DR \times TR}{TR - N \times DR}. \quad (12)$$

If N, DR and TR are fixed, then $\gamma(\frac{CYL}{N\times R})$ is the only factor that can save memory. As with throughput, a large R value does not save much on memory.

Scheme	Pros	Cons
Bidirectional	Minimum memory requirement with large number of partitions.	Highest initial latency.
Unidirectional	Minimum memory requirement with large number of partitions.	High initial latency.
Sequential	Reduced initial latency.	Slightly lower throughput.
In-Memory Replication	Zero initial latency.	High memory requirement.
On-Disk Replication	Negligible Initial Latency, No extra memory requirement.	One more disk required

Figure 8: Scheme Summary

6 Conclusion

In designing a storage system for multimedia applications, the goal is high throughput and low initial delay with minimum cost for hardware, implementation, and maintenance. For the latter, a simple design is critical.

Partitioning the disk into regions ($R \geq 2$) is beneficial mainly if memory conservation is important and the initial latency incurred is not critical. Interactive applications with unpredictable access patterns are not in this category.

With a single partition ($R = 1$), it is actually quite simple to use the Sequential policy and get its high throughput and low latency. With $R = 1$ the distance between retrieving two segments of a presentation is effectively bounded by CYL tracks. This CYL distance may look large at first glance, but because segments are retrieved in physical order, each seek distance is cut on the average by a factor of N.

If the above scheme still does not provide low enough initial latency, it can be further reduced by replicating data. The in-memory and on-disk replication schemes can lower the latency to essentially zero, but as we discussed, the on-disk approach is more cost effective.

Recall that we have not considered the arrival of burst of new requests. Analyzing latency under burst arrivals is beyond the scope of of this paper. However, we believe that scheme Replicated is especially well suited to cope with bursts because one can service the new requests in a round robin fashion. For example, say that two new requests arrive simultaneously. Instead of reading the full replica of one before reading the replica of the second, we can read enough of the first replica to start the IO, and then switch to the second request. After both requests have started, we can continue reading the replica data. This works because the load on the replica disk is expected to be low.

Finally, Figure 8 summarizes pros and cons of the schemes we discussed in this paper.

References

[1] Seagate barracuda 4lp family product specification. *URL: http://www.seagate.com*, 1996.

[2] S. Berson and S. Ghandeharizadeh. Staggered striping: A flexible technique to display continuous media. *MULTIMEDIA TOOLS AND APPL.*, 1(2):127–148, June 1995.

[3] P. Bocheck and H. Meadows. Disk partitioning technique for reducing multimedia access delay. *ISMM Distributed Systems and Multimedia Applications*, August 1994.

[4] E. Chang and H. Garcia-Molina. Minimizing memory use in video servers (ext. version). *Stanford Technical Report SIDL-WP-1996-0050 URL: http://www-diglib.stanford.edu*, Sept 1996.

[5] P. M. Chen and E. K. Lee. Raid: High-performance, reliable secondary storage. *ACM Comp. Surveys*, 26(2), 1994.

[6] A. L. Chervenak and D. A. Patterson. Choosing the best storage system for video service. *Proceedings of ACM Multimedia 95*, pages 109–118, November 1995.

[7] S. Ghandeharizadeh. Continuous retrieval of multimedia data using parallelism. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, 5(4):658–69, 1993.

[8] S. Ghandeharizadeh, S. Kim, and C. Shahabi. On configuring a single disk continuous media server. *SIGMETRICS PERFORMANCE EVALUATION*, 23(1):37–46, May 1995.

[9] S. Ghandeharizadeh, S. Kim, and C. Shahabi. On disk scheduling and data placement for viedo servers. *USC TR*, December 1995.

[10] T. Kunii. Issues in storage and retrieval of multimedia data. *MULTIMEDIA SYSTEMS*, 3(5-6):198–304, 1995.

[11] J. Nussbaumer. Networking requirements for interactive video on demand. *IEEE Journal On Selected Areas In Communications*, 13(5):779–787, June 1995.

[12] B. Ozden, A. Biliris, R. Rastogi, and A. Silberschatz. A disk-based storage architecture for movie on demand. *Information Systems*, 20(6):465–82, 1995.

[13] B. Ozden, R. Rastogi, and A. Silberschatz. Disk striping in video server environments. *EEE International Conference on Multimedia Computing and Systems*, June 1996.

[14] S. Ramanathan and P. V. Rangan. Adaptive feedback techniques for synchronized multimedia retrieval over integrated networks. *IEEE/ACM Trans. on Networking*, 1(2), 1993.

[15] S. Ramanathan, P. V. Rangan, and H. M. Vin. Optimal communication architectures for multimedia conferencing in distributed systems. *Proc. of 12th Conference on Distributed Computing Systems*, June 1992.

[16] C. Ruemmler and J. Wilkes. An intro to disk drive modeling. *Computer*, 2:17–28, March 1994.

[17] F. Tobagi, J. Pang, R. Baird, and M. Gang. Streaming raid-a disk array management system for video files. *First ACM Conference on Multimedia*, August 1993.

[18] H. M. Vin. and P. V. Rangan. Designing a multi-user hdtv storage server. *IEEE Journal on Selected Areas in Communication, Special Issue On HDTV and Digital Video Communication*, 11(1), January 1993.

[19] P. Yu, M.-S. Chen, and D. Kandlur. Grouped sweeping scheduling for DASD-based multimedia storage managemen. *Multimedia Systems*, 1(1):99–109, January 1993.