Reducing Initial Latency in Media Servers

Edward Chang and Hector Garcia-Molina
Stanford University

A multimedia server needs to provide high bandwidth and continuous real-time delivery. We present techniques for reducing the initial latency of presentations. This is important in interactive applications such as video games and multimedia document browsing. On-disk partial data replication can significantly reduce initial latency without adversely affecting throughput.

The delivery of multimedia presentations poses many challenges, from data retrieval[1,2] to real-time network transmission[3-5] to user interface design. An effective multimedia storage system must retrieve data at very high rates and in a continuous fashion. Each multimedia presentation (such as a movie, video, or game) is essentially a sequence of data segments, each of which 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 ensuring high bandwidth and continuous delivery, we seek also to minimize the *initial latency*—the time between the arrival of a new request and the time its first data segment becomes available in the server's memory. Traditionally, initial latency has not received much attention, probably because multimedia servers primarily deliver movies on demand, where a delay of a few minutes before the start of a multi-hour movie is acceptable. Indeed, some proposed servers have either ignored the latency issue entirely[2,6] or acknowledged that they involve initial latencies on the order of minutes.[7]

For some applications, however, high latencies are not acceptable. For example, consider a video game in which a player's actions determine what short video to play next. The player should not have to wait a significant amount of time before each video scene starts. Hypermedia documents on the World Wide Web or in a digital library also require low latency. A user examining a Web page linked to multimedia presentations does not want a significant delay between clicking a link and the presentation's start. We could try to prefetch all videos that might be selected, but this would significantly increase the server load and memory requirements.

Reducing initial latency must not sacrifice throughput (that is, reduce the number of concurrent requests that can be serviced) or violate the continuous delivery constraint. Our investigation into data allocation begins with earlier techniques[7-9] that already achieve high bandwidth and continuous delivery for supporting multiple concurrent data streams. Changing their data placement and disk scheduling policies lets us reduce initial latency drastically without adversely affecting throughput. We also propose a novel on-disk partial data replication scheme that requires only a 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 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 trade-offs between latency, throughput, disk speeds, and available main memory. It is important to understand the interrelationships between these—for example, adding memory to a server can increase throughput or reduce latencies. This article investigates such trade-offs analytically and presents some guidelines for choosing appropriate parameters.

Our study assumes that M disks 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 supports 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 input and output (I/O) for one disk. The total system memory requirement is thus M times the values we report. Similarly, the throughput values equal M times the reported values. Other articles discuss various data placement methods with multiple disks in detail.[6,10-13]

Constrained and partitioned allocation

Table 1 gives the main parameters for the existing storage allocation policies we use as our start-

1070-986X/97/$10.00 © 1997 IEEE

ing point. The first tier of Table 1 lists performance-related parameters such as throughput and initial latency. The second tier describes the physical and derived characteristics of the hardware, including memory and disks. The third tier summarizes the notation we use to refer to presentations 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[1,8,14] took advantage of the sequential access nature of a presentation and proposed limiting the distance between their segments on disk. This placement policy is called *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

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

Constrained allocation reduces the seek distance from the disk's radius (worst case) 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 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 for $X_{i+1,1}$ (the 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.

Table 1. Storage allocation parameters.

Parameter	Description
N	Throughput
T_{Latency}	Initial latency to access 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
$T_{\text{regionSeek}}$	Total seek time in a disk region, ms
$T_{\text{region}TR}$	Total transfer time in a disk region, seconds
T	Period, the time to service a round of requests
X_i	ith presentation
$X_{i,j}$	Segment j of presentation i
K	Number of presentations

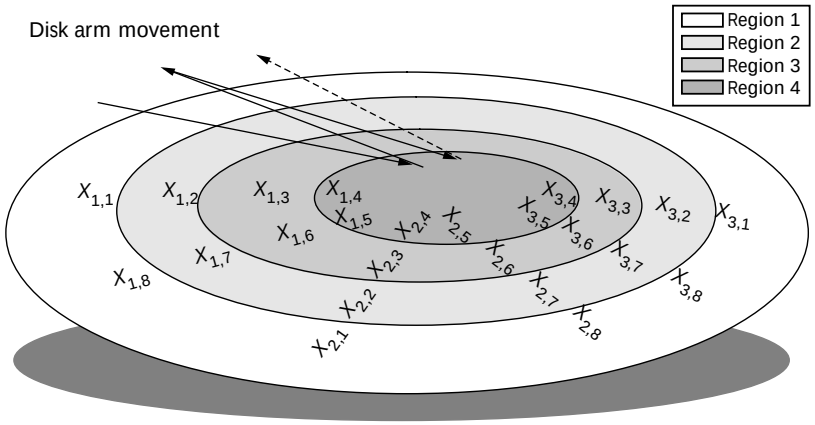

Figure 1. Bidirectional data placement.

Scheme Bidirectional

To remedy the shortcomings of constrained allocation, one study[7] introduced 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 movement of the head sweeping 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 that contains four regions and three videos. The first segment of each video is placed in region 1. Subsequent segments are placed in regions 2 through 4, and then in 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 as 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 again. 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, when on region 4 the first time, the disk head retrieves segment $X_{i,4}$; the next time 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 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 might wait for the disk head to travel up to eight regions. In general, the maximum initial latency equals the time for the disk head to service $2 \times R$ regions. (In the degenerate case $R = 1$, the delay need not be multiplied by 2.) One simulation[7] showed that this delay can be large, especially when the system is busy. For example, with 33 concurrent presentations and 24 regions, the initial latency can be up to 63.2 seconds. Such delays may be unacceptable in many applications with unpredictable requests for short presentations.

We study a variety of schemes to reduce this latency without adversely affecting the number of concurrent presentations. However, we will first briefly derive expressions for the throughput and latency. We will then contrast these expressions to those derived for the later schemes.

Performance expressions

Scheme Bidirectional 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 a private buffer for each stream and do not consider buffer sharing in this article. Issues related to buffer sharing and its performance evaluation have been considered elsewhere.[11]

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 ready to be transmitted as soon as the last one is consumed. Let's denote the time between two consecutive I/Os for a request as T. The continuity constraint requires that each I/O must retrieve a large enough segment S to sustain T time of display at a rate of DR, or

$$T \leq \frac{S}{DR} \quad \textbf{(1)}$$

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

$$T = T_{\text{Region}TR} + T_{\text{RegionSeek}} \quad \textbf{(2)}$$

where $T_{\text{Region}TR}$ is the time to transfer all requested segments in a region and $T_{\text{RegionSeek}}$ is the total seek overhead for segments fetched in a region. With N simultaneous displays, $T_{\text{Region}TR}$ is N times the time to transfer a segment, or $T_{\text{Region}TR} = N \times (S/TR)$.

The worst-case seek distance in a region, with nonconstrained allocation in the region, is the width of the region or $d = Cyl/R$ tracks. The seek for servicing the first request in a region could travel a distance up to two regions, or $d = (2 \times Cyl)/R$ tracks. To make the expressions simpler, we do not include this special case in the derivations. However, our case study takes this special case into consideration.

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

$$T_{\text{RegionSeek}} = N \times \gamma\left(\frac{CYL}{R}\right)$$

Substituting $T_{\text{Region}TR}$ and $T_{\text{RegionSeek}}$ into Equation 2 yields

$$T = N \times \left(\frac{S}{TR} + \gamma\left(\frac{CYL}{R}\right)\right) \quad \textbf{(3)}$$

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

$$N \times \left(\frac{S}{TR} + \gamma\left(\frac{CYL}{R}\right)\right) \leq \frac{S}{DR} \quad \textbf{(4)}$$

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 I/Os for a particular request were separated by exactly T time. However, the time that separates two I/Os 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 I/Os in period *A* have no seek overhead, the I/Os in period *B* suffer from the worst seek overhead $\gamma(Cyl/R)$. In period *A*, since the seek overhead is zero, the I/Os 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*, the I/Os 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 I/Os 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 I/Os of the ith-serviced request is $T + i \times \gamma(Cyl/R)$. This exceeds T, and hence S amount of buffered data per request will not satisfy the continuity constraint.

To remedy this, we need a cushion buffer of size $DR \times i \times \gamma(Cyl/R)$ for the ith request. If I/Os are separated by the maximal amount, this cushion provides the extra data to sustain continuous display. On the other hand, if I/Os are separated by less than T, this cushion buffer serves as the extra space to store the data read in early. For N requests, the total cushion buffer required for Scheme Bidirectional is the sum of the individual ones:

$$DR \times \sum_{i=1}^{N} i \times \gamma\left(\frac{Cyl}{R}\right)$$

Adding these cushion buffers to $N \times S$, we have the memory constraint

$$N \times S + \left(DR \times \sum_{i=1}^{N} i \times \gamma\left(\frac{Cyl}{R}\right)\right) \leq Mem \qquad \textbf{(5)}$$

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. We focus rather 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.

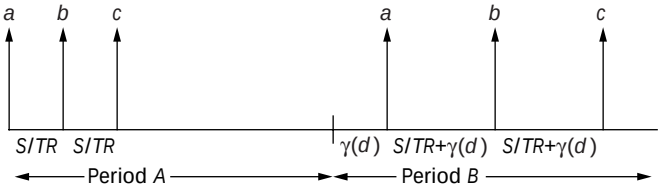

Figure 2. Seek overhead variability.

As mentioned earlier, 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 the seek overhead varies, 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 I/O is at most T away. Since for the ith 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 right-hand side of Equation 3, we have the equation for the worst initial latency:

$$T_{\text{Latency}} = 2 \times R \times N \times \left(\frac{S}{TR} + \gamma\left(\frac{Cyl}{R}\right)\right) + N \times \gamma\left(\frac{Cyl}{R}\right) \qquad \textbf{(6)}$$

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.

Allocation schemes

We examine here three data placement and disk scheduling policies that attain low initial latency while maintaining high throughput:

1. Unidirectional data layout (Scheme Unidirectional),

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

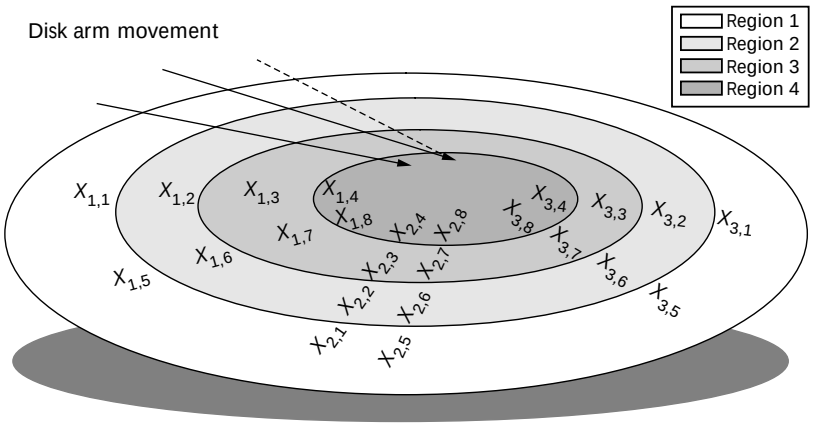

Figure 3. Unidirectional data placement.

3. Group Sweeping Scheme (GSS).

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

Scheme Unidirectional

Instead of placing the segments in an elevator-like zigzag manner as one researcher proposed,[7] others[15,16] have suggested a unidirectional layout. Figure 3 illustrates this layout on our three-video, four-region example. 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.

This strategy reduces the maximum initial latency 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, resetting the disk head typically takes less than 20 milliseconds—negligible compared with the magnitude of the initial latency that we aim to improve.

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

$$T_{\text{Latency}} = R \times N \times \left(\frac{S}{TR} + \gamma\left(\frac{Cyl}{R} \right) \right) + N \times \gamma\left(\frac{Cyl}{R} \right) \quad \textbf{(7)}$$

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

Scheme Sequential

So far we have assumed a nonconstrained allocation of segments within a region, with which each seek distance can be as large as the region's width. 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 $Cyl/(R \times N)$, 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, 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 intersegment 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 $k + 1$, 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 are ready to repeat the process. Playback thus takes place as in the earlier scenarios, with the exception of the cushion buffers needed to handle the variable time between accesses to consecutive segments. This means that our memory constraint is

$$2 \times N \times S \leq Mem \quad \textbf{(8)}$$

The reduced seek times for the sequential policy lead 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, and for 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 intersegment seek distance d is $Cyl/(R \times N)$. 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. (Ruemmler and Wilkes[17] further explain this.) Therefore, we can express the worst seek overhead as

$$T_{\text{Latency}} = (R+1) \times N \times \left(\frac{S}{TR} + \gamma \left(\frac{Cyl}{R \times N} \right) \right) \quad \textbf{(9)}$$

To compute N for a given R, Scheme Sequential follows the same procedure described in the previous section. For a given R, we can solve N with Inequalities 1 and 8. Notice that the seek distance used to compute seek overhead is replaced with $d = Cyl/(N \times R)$. The largest N that satisfies both inequalities is Scheme Sequential's throughput. Once N is known, we can compute T_{Latency} using Equation 9.

Group Sweeping Scheme (GSS)

So far 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)[9] achieves an optimal balance between these two extremes. GSS divides N requests into G groups, each servicing N/G requests. Within each group, GSS uses Scheme Sequential to amortize seek overhead among the N/G requests. Between groups, GSS services them 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 requests in a fixed order from period to period and hence converges to Scheme Unidirectional. The optimal G value tends to be small when the disk load is high (large N).[9] Our study[11] shows that the optimal G value is also sensitive to the memory management policies employed.

GSS does not help reduce the worst-case initial latency 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 the number of 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.

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 of the data placement and disk scheduling schemes we have discussed so far; however, here we only use Scheme Sequential to illustrate.

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 nonreplicated segment on disk. At that point, playback continues from disk.

There are two memory overheads associated with this in-memory replication scheme: *replicas* overhead and *playback* overhead. First, the amount of memory needed to hold the replicas equals the number of presentations K times the data needed to cover the worst latency of T'_{Latency} seconds. Variable K represents the total number of presentations stored, not just those currently being played. 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

$$K \times DR \times T'_{\text{Latency}} \quad \textbf{(10)}$$

Let's use the example in Figure 4 (next page) to illustrate this memory overhead. A disk is partitioned into two regions and the segments of presentation X_1 placed in a round-robin fashion from the outermost region (the first region) to the innermost one (the second region). Employing Scheme

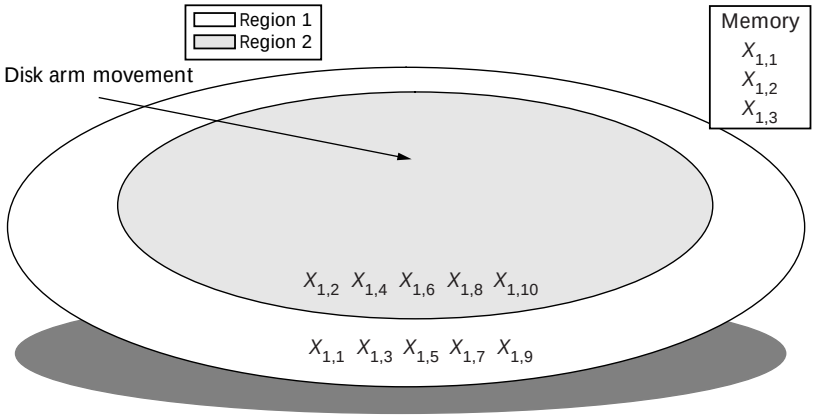

Figure 4. In-memory replication.

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. Continuing 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 $3 \times S$. We can easily 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 later in this article 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. We can also reduce in-memory replica size 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.

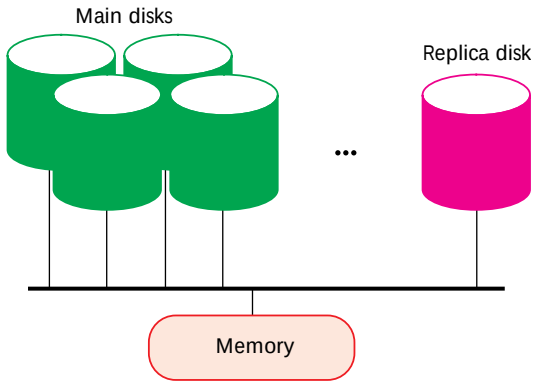

Figure 5. A replica disk configuration.

On-disk replication

To reduce the memory cost, we propose an on-disk segment replication scheme we call *Replicated*. This scheme 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 memory through a high-speed bus. Briefly, Scheme Replicated 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.

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

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 to finish the current disk sweep and another 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 $2 \times S$.

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 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 (we make a strong assumption here to simplify the discussion; elsewhere[18] we discuss more flexible data placement policies), 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. Assume that the I/O from the replica disk is scheduled to complete at time t_1 and that at this time playback starts. Assume also 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. As the period ends on the main disk, we will have exactly S data for this request, which is normal. During the next scan, we play back 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.

Cost. For each presentation, the replica disk stores only one copy of its initial segments. For K presentations, the total disk space requirement is $K \times DR \times T'_{\text{Latency}}$.

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

Reduced initial latency. Since we do not consider request bursts in this study, the worst initial latency is the worst seek overhead before the first I/O can commence for a newly arrived request on the replica disk. 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 $T_{\text{Latency}} = \gamma(Cyl)$. For modern disk drives, this worst initial latency is below 30 ms, negligible to human perception.

However, if the replica disk is not full, then we can place the replicas in the disk's outer zones. 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 rates.[17] Second, using a limited number of cylinders decreases seek time. Therefore, outer zone data placement may further cut latency from the equation given above.

Table 2. Disk parameters for the Seagate Barracuda 4LP family.

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 ms
Min. seek time	0.9 ms
Max. seek time	17.0 ms
$\alpha 1$	0.6 ms
$\beta 1$	0.3 ms
$\alpha 2$	5.75 ms
$\beta 2$	0.0021 ms

Case analysis of the proposed schemes

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.

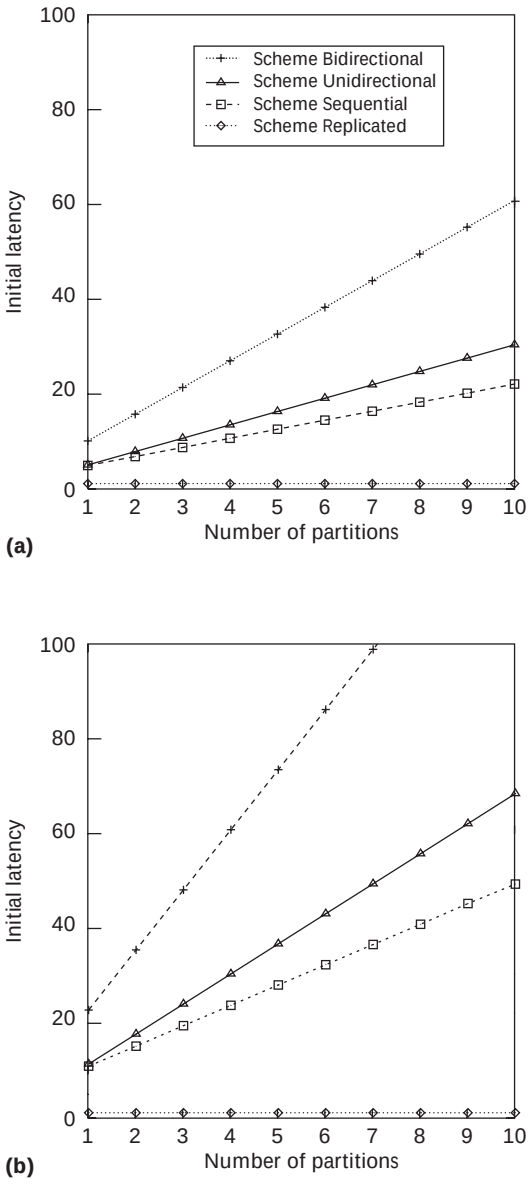

Figure 6. Initial latency versus throughput for (a) **N** *= 40 and (b)* **N** *= 45.*

For the seek overhead we use the following concave function:[19]

$$\gamma(d) = \alpha 1 + (\beta 1 \times \sqrt{d}) + 8.33 \text{ if } d < 400$$
$$\gamma(d) = \alpha 2 + (\beta 2 \times d) + 8.33 \text{ if } d \geq 400$$

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 two-thirds 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 $\beta 1$ 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 manufacturer's time (17 ms) and so that function γ is continuous at $d = 400$. The values obtained are given in Table 2. (Ruemmler et al.[17] suggest using between 200 and 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.) We also add a worst-case full rotational delay to the seek overhead.

Figure 6 and Figure 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 Replicated. As we discussed, 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.)

Our evaluations use two memory configurations: 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 the bidirectional, unidirectional, and sequential schemes. Also as expected, Scheme Unidirectional cuts the latency of Scheme 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 Scheme Unidirectional more quickly. (We discuss this further below.) 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, 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 the 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 six regions and 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. We thus must consider how Bidirectional, Unidirectional, and Sequential would benefit if they were given the extra memory or the extra disk.

First, if given more memory, all schemes might be able to improve throughput. However, this would be limited, since the memory requirement to support additional requests grows without bound as disk utilization approaches its capacity. Figure 7 illustrates this trend. When memory doubles from 32 to 64 Mbytes, the throughput increases by only up to four streams, a 10-percent 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.

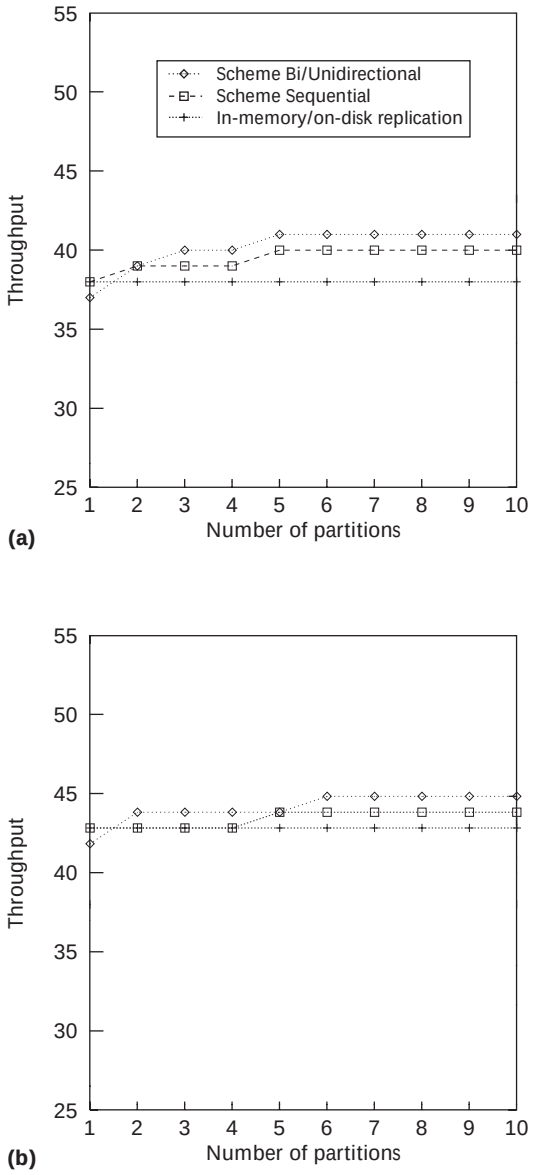

Figure 7. Throughput versus number of partitions for (a) 32 Mbytes and (b) 64 Mbytes of available memory.

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's examine Inequality 1. Leaving N on the left side of the inequality, we rewrite it as

$$N \le \frac{TR}{DR} \times \frac{S}{S + \left(\gamma(d) \times TR\right)} \qquad \textbf{(11)}$$

Notice that in the inequality TR/DR is the throughput cap. The major factor that influences N is $\gamma(d)$. The best way to improve N is thus to reduce $\gamma(d)$. Now recall that the d for Scheme Sequential is $Cyl/(R \times N)$, hence the worst seek overhead $\gamma(Cyl/(R \times N))$ is

$$\alpha = \left(\beta \times \sqrt{\frac{CYL}{R \times N}}\right)$$

Parameter Cyl is a constant, so the $\gamma(Cyl/(R \times N))$ 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 $\gamma(d)$), elimi-

Table 3. Memory requirement versus partitions.

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 4. Summary of schemes discussed in the article.

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

nating 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 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

$$S \geq \gamma\left(\frac{Cyl}{N \times R}\right) \times \frac{N \times DR \times TR}{TR - N \times DR} \quad \textbf{(12)}$$

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

Conclusion and future work

Designing a storage system for multimedia applications focuses on high throughput and low initial delay with minimum hardware, implementation, and maintenance costs. The latter depends on a simple design.

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, replicating data will further reduce it. 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.

As mentioned, analyzing latency under burst arrivals is beyond the scope of this article. However, we believe that Scheme Replicated is especially well suited to cope with bursts because it permits servicing 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 I/O, 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.

Table 4 summarizes pros and cons of the schemes discussed in this article. In addition to the data placement schemes (allocation and replication) discussed and proposed here, we have also developed a resource management scheme, BubbleUp,[18] that effectively minimizes the initial latency without adversely affecting throughput. As part of our research, we plan to scale up these results into a multiple disk environment. MM

References

1. S. Ghandeharizadeh, "Continuous Retrieval of Multimedia Data Using Parallelism," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, No. 4, 1993, pp. 658-69.
2. T. Kunii, "Issues in Storage and Retrieval of Multimedia Data," *Multimedia Systems*, Vol. 3, No. 5-6, 1995, pp. 198-304.
3. J. Nussbaumer, "Networking Requirements for Interactive Video-on-Demand," *IEEE J. on Selected Areas In Comm.*, Vol. 13, No. 5, June 1995, pp. 779-787.
4. S. Ramanathan and P.V. Rangan, "Adaptive Feedback Techniques for Synchronized Multimedia Retrieval over Integrated Networks," *IEEE/ACM Trans. on Networking*, Vol. 1, No. 2, Apr. 1993, pp. 246-260.
5. S. Ramanathan, P.V. Rangan, and H.M. Vin, "Optimal Communication Architectures for Multimedia Conferencing in Distributed Systems, *Proc. 12th Conf. on Distributed Computing Systems*, IEEE Computer Society Press, 1992, pp. 46-53.
6. A.L. Chervenak and D.A. Patterson, "Choosing the Best Storage System for Video Service," *Proc. ACM Multimedia 95*, ACM Press, New York, 1995, pp. 109-118.
7. S. Ghandeharizadeh, S. Kim, and C. Shahabi, "On Configuring a Single-Disk Continuous Media Server," *Sigmetrics Performance Evaluation*, Vol. 23, No. 1, 1995, pp. 37-46.
8. H.M. Vin and P.V. Rangan, "Designing a Multi-User HDTV Storage Server," *IEEE J. on Selected Areas in Comm.* (special issue on HDTV and digital video communication), Vol. 11, No. 1, Jan. 1993, pp. 153-164.
9. P. Yu, M.-S. Chen, and D. Kandlur, "Grouped Sweeping Scheduling for DASD-Based Multimedia Storage Management," *Multimedia Systems*, Vol. 1, No. 1, Jan. 1993, pp. 99-109.
10. S. Berson and S. Ghandeharizadeh, "Staggered Striping: A Flexible Technique to Display Continuous Media," *Multimedia Tools and Applications*, Vol. 1, No. 2, June 1995, pp. 127-148.
11. E. Chang and H. Garcia-Molina, "Effective Memory Use in a Media Server," Stanford Tech. Report SIDL-WP-1996-0050, Sept. 1996, http://www-diglib.stanford.edu; to appear in *Proc. 23rd Very Large Database Conference*, Aug. 1997.
12. P.M. Chen and E.K. Lee, "RAID: High-Performance, Reliable Secondary Storage," *ACM Comp. Surveys*, Vol. 26, No. 2, June 1994, pp. 145-188.
13. B. Ozden, R. Rastogi, and A. Silberschatz, "Disk Striping in Video Server Environments," *IEEE Int'l Conf. on Multimedia Computing and Systems*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1996, pp. 147-154.
14. P. Bocheck and H. Meadows, "Disk Partitioning Technique for Reducing Multimedia Access Delay," *ISMM Distributed Systems and Multimedia Applications*, Aug. 1994, pp. 27-30.
15. S. Ghandeharizadeh, S.H. Kim, and C. Shahabi, "On Disk Scheduling and Data Placement for Video Servers," Tech. Report USC-CS-TR97-650, University of Southern California, Los Angeles, Dec. 1995.
16. B. Ozden et al., "A Low-Cost Storage Server for Movie-on-Demand Databases," *Proc. VLDB 94*, IEEE Computer Society Press, Los Alamitos, Calif., Sept. 1994, pp. 594-605.
17. C. Ruemmler and J. Wilkes, "An Introduction to Disk Drive Modeling," *Computer*, Vol. 27, No. 3, Mar. 1994, pp.17-28.
18. E. Chang and H. Garcia-Molina, "BubbleUp: Low Latency Fast-Scan for Media Servers," to appear in *Proc. 5th ACM Int'l Multimedia Conf.*, ACM Press, New York, Nov. 1997.
19. F. Tobagi et al., "Streaming RAID—A Disk Array Management System for Video Files," *First ACM Conf. on Multimedia*, ACM Press, New York, 1993, pp. 393-400.

Edward Y. Chang received a BS degree from Tunghai University, Taiwan, in 1981, an MS in industrial engineering from the University of California, Berkeley, in 1985, and an MS in computer science from Stanford University, Stanford, California, in 1994. He is presently a PhD candidate in the Department of Electrical Engineering at Stanford University. His research interests are in database and multimedia systems.

Hector Garcia-Molina is the Leonard Bosack and Sandra Lerner Professor in the Departments of Computer Science and Electrical Engineering at Stanford University. From 1979 to 1991 he was on the faculty of the Computer Science Department at Princeton University. His research interests include distributed computing systems, database systems, and digital libraries. He received a BS in electrical engineering from the Instituto Tecnologico de Monterrey, Mexico, in 1974. He received an MS in electrical engineering in 1975 and a PhD in computer science in 1979 from Stanford University.

Contact Chang and Garcia-Molina at the Computer Science Dept., Stanford University, Stanford, CA 94305, e-mail {echang, hector}@cs.stanford.edu.