Just-in-time Scheduling for Video-on-Demand Storage Servers

Steven Berson
Richard R. Muntz
UCLA Computer Science Department

April 16, 1995

Abstract

Recent technological advances have made multimedia on-demand servers feasible. Two challenging tasks in such systems are: a) satisfying the real-time requirement for continuous delivery of objects at specified bandwidths and b) efficiently servicing multiple clients simultaneously. Real-time delivery can be achieved via clever scheduling but still often comes at a price, e.g., buffer space. Most published video server designs use a cycle based scheduling method in which each disk serves a set of I/O requests during a cycle and that data is played out to the display stations during the following cycle. This technique enables efficient disk scheduling by permitting scheduling of the requests in an order that minimizes seek times. However, this method requires significant amounts of memory to buffer data between the time when it is read until the time it is played out to a display device. In contrast, the design proposed in this paper lays out the videos and schedules request streams such that the playout of data begins almost immediately after it is read from disk. The result is a design which requires an order of magnitude less main memory buffer space.

1 Introduction

Advances in digital signal processing, data compression techniques, and high speed computer networking have made on-demand multimedia servers commercially feasible. The real-time requirement of continuous delivery of objects at a constant bandwidth per object is a challenging technological issue. The continuous delivery at a constant bandwidth is required since once a video begins, it must be transmitted continuously for the duration of the film. Another challenging design issue in multimedia systems is to service multiple clients simultaneously. For example, both of these tasks are accomplished in [3] through a novel layout of the data on the disk drives, using a technique termed *staggered striping*. There have been a number of other proposals of storage layouts and associated scheduling algorithms [5, 1, 2]. In this paper we concentrate on systems in which (1) all objects (videos) have the same required bandwidth and (2) the display bandwidths are smaller than the bandwidth of a single disk, e.g., MPEG 2 video is approximately 6 Mbs while a typical disk would have a bandwidth of 30 Mbs.

An example multimedia system is illustrated in Figure 1; it includes a multimedia server, a communication network, and a set of display stations. The multimedia server consists of a tertiary storage library, a collection of disks, and a set of processors. The entire database permanently resides on tertiary storage, from which objects are retrieved and placed on disk drives for delivery

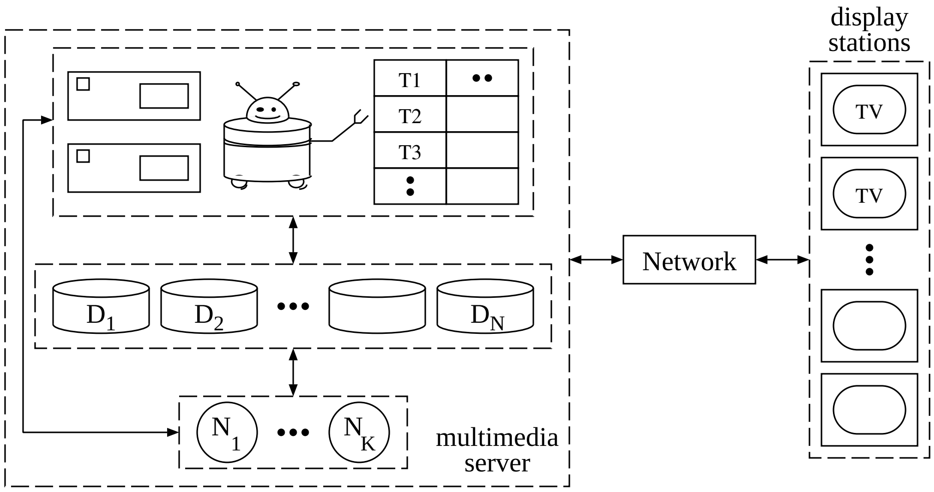

Figure 1: Multimedia System

on demand. The objects are delivered from disk drives since the size of the objects precludes them from being stored in main memory. When secondary storage capacity is exhausted and an object, which is not on the disks, is requested then one or more disk-resident objects must be deleted from the disk storage to make space for the requested object. The long latency times and high bandwidth cost of tertiary devices[1] precludes objects from being transmitted directly from the tertiary store.

To exhibit reasonable economies of scale, the multimedia server should contain a large number of disks; something on the order of 100 drives would not be uncommon. For example, 100 (2 gigabyte) disks provide enough storage for approximately 44 (100 minute) MPEG-2 movies (about 4.5 megabits per second, i.e., good TV quality) or 176 MPEG-1 movies (about 1.5 mbps, i.e., low TV quality) or some combination of the two. Similarly, assuming a bandwidth of 4 megabytes per second, 100 of these disk drives provide enough bandwidth to support approximately 500 concurrent MPEG-2 users or 2,000 MPEG-1 users. These numbers assume that the system can use the combined total of all the disk bandwidths in delivering video streams. The challenge is, of course, to at least come close to this ideal and to do it without excessive use of other resources such as main memory. The scheme we propose in this paper is particularly concerned with efficient use of main memory as it is one of the most expensive resources of the storage server.

The rest of the paper is organized as follows. Section 2 describes the details of the proposed scheduling method. Section 3 discusses simulation results in which we compare the performance of the proposed method with other published schemes with respect to perfomance metrics such as the fraction of requests that are forced to wait (and the mean waiting time given a request has to wait) as a function of the load (in requests per second). Extensions to the basic scheme are discussed briefly in Section 4 and Section 5 presents concluding remarks.

[1]The bandwidth for a $1000 tape drive is approximately 4 megabits per second (mbps) whereas a similarly. priced disk has a bandwith of approximately 30 mbps.

2 Just in time delivery scheduling

Figure 2 illustrates what is desired for "just-in-time" reading from the disk and then playout, via the network, to the display devices. Note that the start of the playout of the $i+1$-st track starts at the completion of the playout of the i-th track. We assume that prior to the start of the playout of the i-th track, the entire i-th track is read from disk. While track i has to be read before the scheduled playout time, there is no restriction on how early it is read from disk. However, a track read early must be buffered. Our goal is to push the design in the direction of minimizing buffer requirements by reading tracks from disk so they are buffered for the least amount of time.

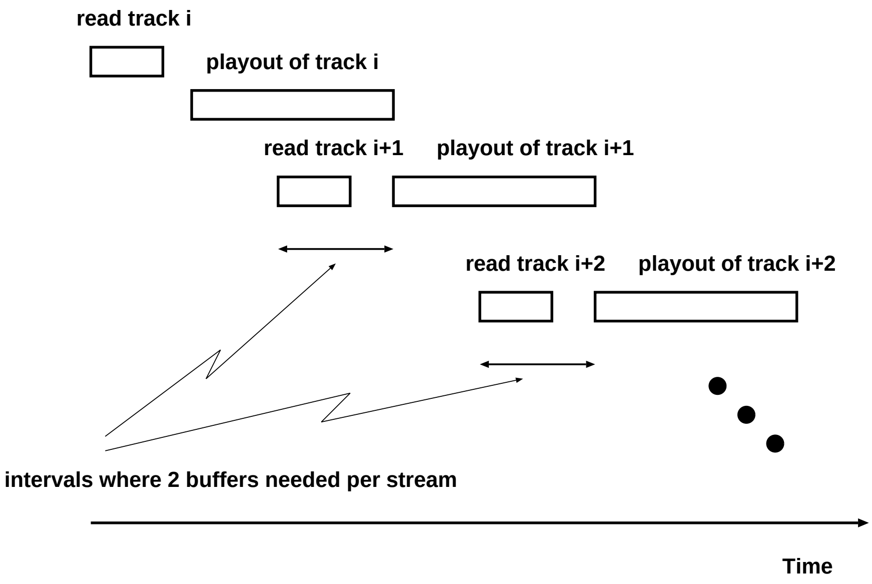

Figure 2: Just-in-time Disk Scheduling

2.1 Striping over all disks.

We assume that all videos are striped over all of the disks. As described, for example in [3], striping videos over all the disks is the most flexible scheme with respect to accomodating any partitioning of the aggregate disk subsystem bandwidth to video streams. If N is the total number of streams that can be supported, any number of streams could be active for any video as long as the total is no larger than N. For example, all N active streams could be for one video, or the number of active streams could be the same for each video.

2.2 Overview of the scheduling strategy.

First consider the scheduling of a single disk. In the simplest form of our scheme, the scheduling goes in cycles in which all active streams are served twice: once in a sweep from the outer tracks to the inner tracks (termed an O$\rightarrow$ I sweep) and once in a sweep from the inner tracks to the outer tracks (termed an I$\rightarrow$O sweep). The order of service to each stream is the same in each sweep.

The scheduling of all disks is coordinated in the following manner.

1. the order of service of streams is the same not only for the I→O and O→I sweeps but is the same for all sweeps on all disks.

2. disks are numbered 1 through n. The schedule for disk $i + 1$ is identical to that of disk i except that the initiation of the sweeps is offset by an amount Δ which is equal to the playout time of a track.

This scheme is proposed to satisfy two desires. One is to use the disks most efficiently and this is accomplished by serving all active streams during a sweep in which the heads do not reverse direction. This implies small seek times between reading tracks and therefore high disk efficiency. The second goal is the "just-in-time" delivery to main memory. Having the same ordering on each sweep supports this goal as is described next.

A schedule of the type described above is illustrated in Figure 3. In this figure we have portrayed an idealized schedule but it illustrates the idea.

Now let us consider the basic parameters for this schedule and how they are chosen based on the disk characteristics and the video bandwidth requirement. Again, we will for simplicity, assume that a track is fetched per stream per sweep, but this assumption is easily eliminated and is only for the purpose of simple expostition.

Notation

n	=	number of disks
Δ	=	playout time of a track (more generally it would be the playout time for whatever number of bytes is read for a stream in a sweep).
τ_{trk}	=	time required to read a track. This includes any time required for head to slow down and stop or accelerate to full speed on the following seek.
τ_{seek}	=	time to seek from the outermost to the inner most track (or the other way) without stopping.

We already know that the schedule for each successive disk is offset by an amount Δ seconds. A typical value of Δ would be in the range of 75 msec. This corresponds, for example, to a display bandwidth of 6 Mbs and a track size of approximately 50 KB. If there are n disks, the cycle lenght is $\tau_{cyc} = n \times \Delta$ sec. The number of streams served during a sweep is determined by the number of tracks that can be read in τ_{cyc}. In this paper we use a simple disk model in which the cycle time consists of τ_{seek} defined above and a service time for each track, τ_{trk}. Then the number of streams that can be serviced in a cycle (which is the same as the total number of active streams in the system), N_{max} satisfies:

$$N_{max} \leq \frac{\Delta \times n - \tau_{seek}}{\tau_{trk}}$$

Example

As noted above, with each stream requiring a bit rate of 6 Mbs and a track size of 50 KB, Δ, the playout time for a track is approximately 67 msec. With 100 disks, $\tau_{seek} = 20$ msec. and $\tau_{trk} = 15$ msec., we get $\tau_{sweep} = 6700$ msec. and $N_{max} \leq 445$.

In the next section we consider how to layout the video data such that this type of schedulng can be supported. Note that some constraints are required since, on each successive sweep, utilizing the disks in a round robin fashion, the next logical track of a video must be stored, physically, in the same relative position on the disk in order that the sweep be able to serve each stream in the required order and not have to reverse direction of the head motion during a sweep.

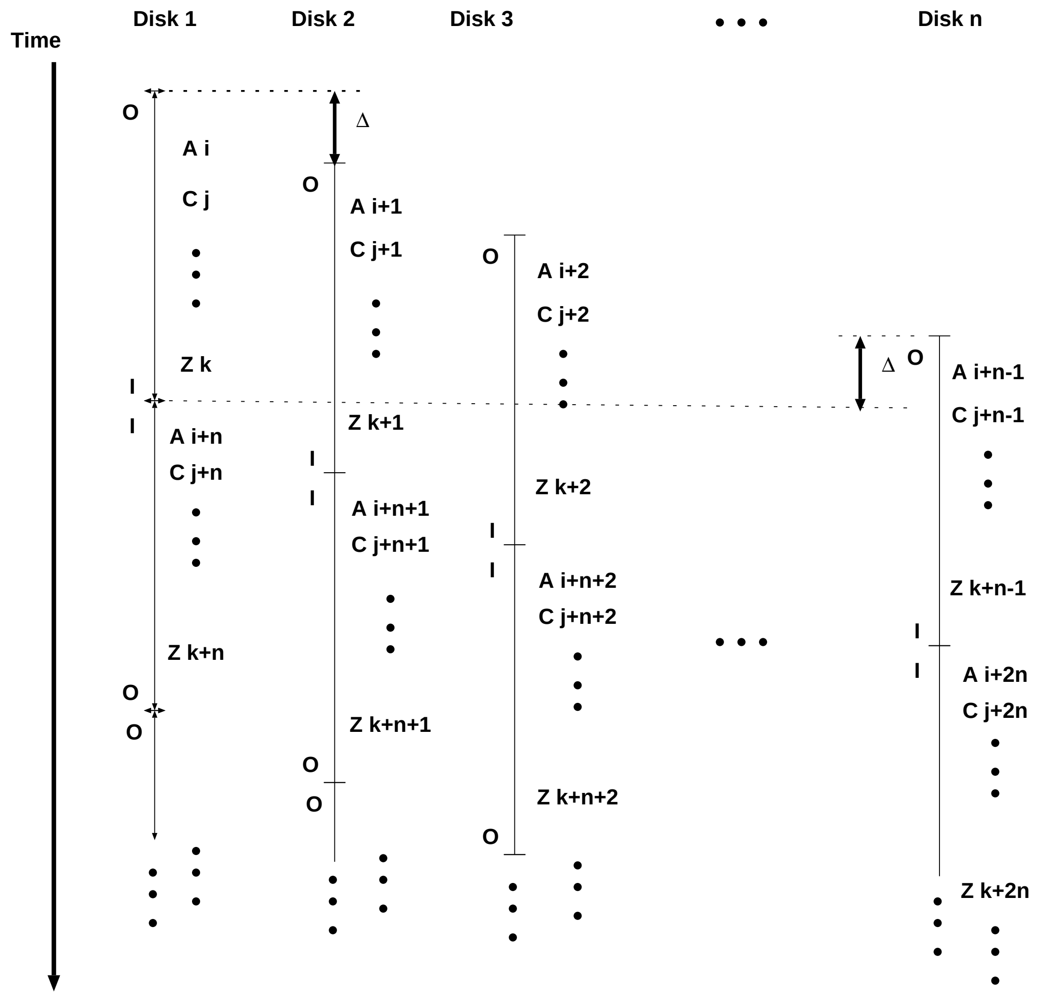

Figure 3: Full Sweep Scheduling of Requests

2.3 Layout of videos on disk

The layout of the videos on disk has to done such that the scheduling policy as outlined in the previous section can be accomodated. This means that the space allocated to a video will be in one "band" (same set of consecutive tracks on each disk) for the $O \rightarrow I$ sweep and another band for the $I \rightarrow O$ sweep. One way to see how this can be done is to first consider a single large video that occupies all of the disk storage. If we had such a video then we would lay out the successive tracks of this video as illustrated in Figure 4. This layout defines a linear ordering of all the tracks on all the disks. The i-th track in this linear ordering is the physical track that the i-th logical track of our large video would be stored, i.e., logical track i in this linear ordering is found by expressing $i = d \times n + j$ where $0 \leq j < n$. Then logical track i is on disk j and the physical track number is given by $d/2$ if d is even, or $T - \frac{d+1}{2}$ if d is odd.

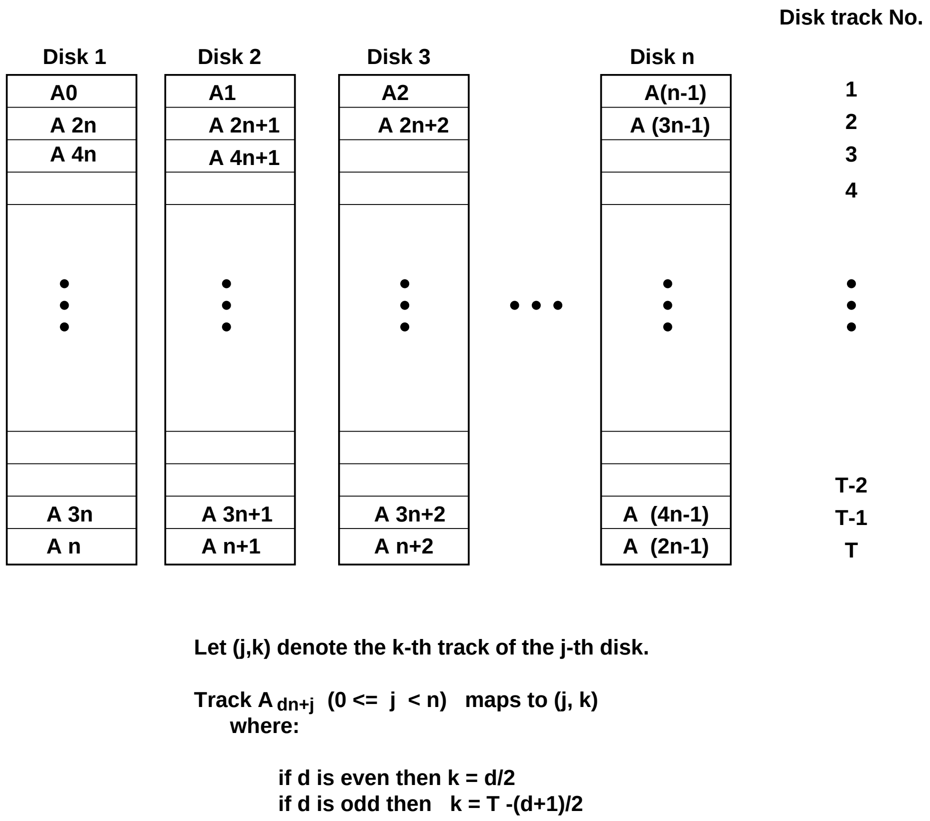

Figure 4: Logical Ordering of Disk Blocks

Figure 5 illustrates how disk storage allocation is done. Storage for videos is allocated from contiguous sections of the linearized space of cylinders. The region denoted by R in the top portion of the figure is shown as the shaded bands across all the disks in the bottom portion of the figure. The logical tracks of the video are stored in the corresponding physical tracks assigned to the video.

We note that fragmentation is a potential disadvantage in this scheme. The magnitude of this problem will depend on distribution of sizes of videos, rate of deletion/loading of videos and other characteristics of the workload. However we do not believe this will be a significant cost for several reasons. The first is that, with a moderate amount of bandwidth, one can move videos from one location to another and thereby coalesce free space to reduce fragmentation. The second reason is that disk storage capacity is increasing faster than disk bandwidth. The 10% or so that is typically lost to fragmentation will not be significant to the extent that disk bandwith becomes a bottleneck before storage capacity and this is the trend with current technology.

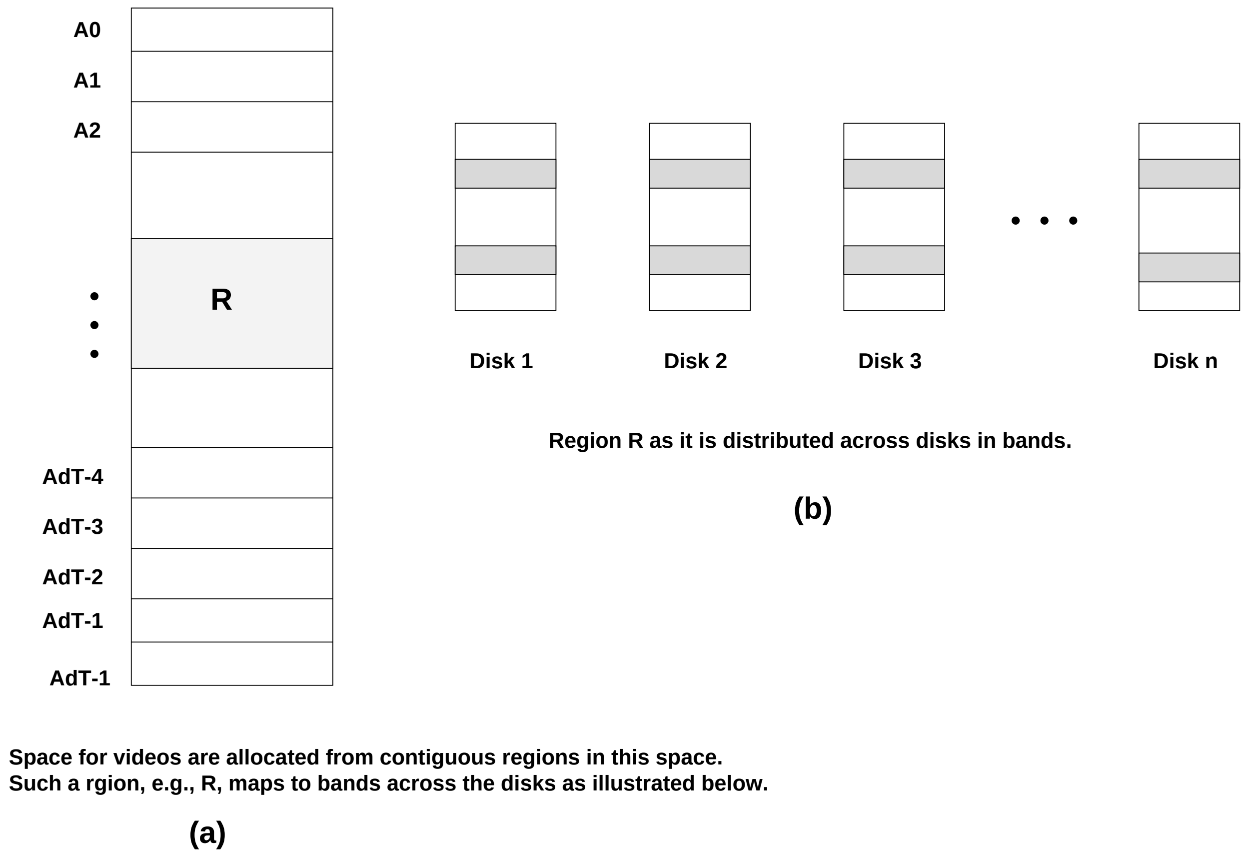

Figure 5: Linear Mapping of Disk Blocks

2.4 Scheduling streams

Assume a new request arrives for a video and there are some number of streams in progress. The new request has to be assigned a slot in the sweep schedule. The slot assigned also has to be consistent with the assignment of slots to streams for other videos. To formally express the restrictions we need a few definitions.

Def. For a video V, let $B(V)$ be the set of videos such that any active streams on a video in $B(V)$ must be scheduled prior to any stream on V in a sweep. $B(V)$ contains all videos whose allocation is prior to that of V in the linear ordering of tracks. Similarly let $A(V)$ be the set of videos such that any active streams on a video in $A(V)$ have to be scheduled after all the streams on V in a sweep.

Note that the storage location of a video creates the restriction on how streams can be ordered within the sweep schedule such that reversal of direction of the read/write heads is never required.

Within a sweep up to N_{max} tracks can be read, one for each active stream. One can think of the sweep as containing N_{max} *slots* to be allocated to streams but with the restrictions just discussed above. These restrictions can possibly lead to a kind of fragmentation problem. For example, suppose all slots $\geq i$ are assigned and slot i is allocated to a stream for a video V_0. Now if a new request for a video V_1 arrives where $V_1 \in A(V_0)$ we can not assign a slot to the new request since it must appear after V_0 in the schedule and all of these slots are busy. If there are idle slots prior to slot i in the sweep schedule we have a form of fragmentation.

Our solution to this problem is to allow schedules to be "slipped" by one slot. For example, if we need to put a new stream in slot i and slots i through $i+k$ are occupied, but $i+k+1$ is idle, then we can reassign the streams occupying slots i through $i+k$ to slots $i+1$ through $i+k+1$. The new request can then be assigned to slot i which has been freed up. We note that this will cause a discontinuity in the streams that have been slipped by one slot. Note that no data is lost, it is just that there will be no new data delivered for the steams that have been slipped for an interval of time corresponding to τ_{trk} which is about $\frac{1}{80}$-th of a second. This is small enough that it will not be noticeable by the viewer.

Based on this observation, the scheduling strategy is to allocate to a new stream, the lowest numbered slot consistent with the storage location assigned to the video relative to others. That is, when a new request for a video V_0 is to be scheduled, we determine first the minimum number slot that can be assigned which is immediate after the highest slot number currently assigned to any stream for an object in $B(V_0)$. If that slot is available then nothing is required. If that slot is busy, and there are available slots later in the sweep schedule, then we slip slot assignments to allow the new stream to be assigned a slot.

Simulation results are presented in the following section which demonstrate the effectiveness of this scheme in keeping fragmentation effects low.

3 Evaluation

In this section, we compare our ordering scheme with Streaming RAID[2] and show that it provides superior performance using less resources.

We will be using the following parameters for comparison: disk bandwidth = 30 megabits per second (mbps), object delivery bandwidth = 6 mbps, disk track size = 50 kilobytes (kB), average seek time 13 milliseconds (ms). For Streaming RAID, we will assume that 40 tracks of data are read per cycle. Reading 40 tracks per cycle will give an aggregate disk bandwidth efficiency of 95 percent. There is a memory cost for reading 40 tracks per cycle. The buffer requirements for Streaming RAID are 40 buffers per disk, each buffer of 50kB in size, giving a total buffer requirement of approximately 2 megabytes (MB) per disk. We will be considering a moderate size disk array of 100 disks meaning 200MB of memory required for the entire array. An alternative scheme called staggered grouping[4] can reduce this memory requirement to 100MB.

By contrast our just-in-time scheme requires 3 tracks of buffering per disk for staging the data, or 150kB of buffering per disk or 15MB for the entire disk array. In addition to the memory needed for staging, it is also desirable to have some buffers for delivering the data out of order. In this section, we assume that 3 buffers are used for delivering data out of order. As described earlier if a disk can support the delivery of 5 objects simultaneously, a single buffer can support 5 streams being delivered out of order. Thus 3 buffers allow 15 streams to be delivered out of order at any one time.

With these numbers, we can examine our simulation data. The simulations were run simulating one week of server execution. The length of each video was 6000 seconds (or 100 minutes) at a bandwidth of 6 mbps for a total size of of 4 gigabytes (GB) per video. Using 100 disks of 4GB each allows us to store 100 videos per server. With the ordering scheme, 5 (= 30 mbps/6mbps) streams can be supported by each disk allowing up to 500 simultaneous streams on the server. By contrast Streaming RAID using 40 buffers can achieve 95 percent disk bandwidth utilization limiting the number of simultaneous streams that can be supported to 475.

We have simulated both these schemes, Streaming RAID with 40 buffers and ordering with 6 buffers, for a range of utilizations. The results here are shown for a movie arrival request of 0.075 arrivals per second or approximately one arrival every 13.3 seconds. This arrival rate gives an average utilization of 0.9 for the ordering scheme and 0.95 for the Streaming RAID scheme. It has been estimated that the arriving requests for movies obeys a Zipf distribution[6], and this is what we have simulated. We have also investigated other movie arrival distributions such as a truncated geometric distribution, and a bimodal (popular versus unpopular) distribution and the results were very similar.

A Zipf distribution is skewed, with a steep drop at the beginning, but with a flat tail. If p_k is the probability of requesting movie k, then $p_k = C/k$ where

$$C = \sum_{i=1}^{N} 1/i$$

Given the 100 movies in our configuration, the Zipf distribution gives $p_1 = 0.19$, $p_2 = 0.096$, $p_3 = 0.064$, while $p_{98} \approx p_{99} \approx p_{100} = 0.0019$. Another measure of skewness is the number of movies whose probability totals 80 percent of the requests. In the case of Zipf distribution with 100 movies, the total probability of the 35 most popular movies exceed 80 percent.

For further comparison of whether ordering schemes are affected by movie distribution, we will use a bimodal distribution and a truncated geometric distribution. We use a bimodal distribution where 80 percent of the probability is divided evenly among the 50 most popular movies, and the remaining 20 percent probability is divided evenly among the 50 least popular movies. This is a much more uniform distribution (i.e. less skewed) than the Zipf distribution. The truncated geometric distribution with a parameter of 0.1 is much more skewed than a Zipf distribution with the most popular 20 movies comprising 80 percent of all requests and each movie 10 percent less requested than its predecessor.

We look at 2 different results from the simulations. The first result is the percentage of movie requests that are forced to wait at all while the second result is the average wait for a request that is forced to wait.

The following table shows the probability that a request needs to queue for service. Using Streaming RAID, over 13 percent of all requests (at 0.9 utilization) need to queue for service. The percentages of requests that need to queue using our ordering scheme with a variety of movie distributions all have a percentage of requests that need to be queued of less than 1 percent.

RAID	13.3%
Zipf	0.88%
Geometric	0.78%
Bimodal	0.76%

The next table shows the average wait faced by an arriving movie request that is forced to queue for service. Again, due to the better bandwidth utilization, the ordering scheme makes customers wait less given that a customer has to queue. The wait is almost twice as long using Streaming RAID as opposed to the ordering scheme.

RAID	131 seconds
Zipf	73 seconds
Geometric	77 seconds
Bimodal	78 seconds

The performance of Streaming RAID can be improved by increasing the length of the cycle and thus increasing the number of tracks read per cycle. But increasing the number of tracks read per cycle in Streaming RAID can only be accomplished by increasing the amount of buffers per disk. By taking advantage of the ordering of data on disk, highly efficient disk utilization can be achieved using only a constant amount of memory.

4 Extensions

The basic scheme was presented and evaluated in previous sections and the description was kept as simple as possible for ease of exposition. There are a number of possible extensions to the basic strategy which we are exploring currently. We briefly describe some thoughts on a few of these here.

First, there is no need to require that the entire time $\Delta \times n$ be devoted to one sweep time. The sweep gets to be quite long (5 to 10 seconds typically) and when a new stream request can not be fit into the current sweep schedule, there is nothing that can be done until some stream completes. If we halve the sweep time so that two complete sweeps are done in $\Delta \times n$, then there are two sweep schedules that the new request could fit into. Each of course has fewer slots. Initial experiments have shown that having more, shorter sweeps is beneficial up to a point. When the sweep times get too short, of course, the disk seek time becomes significant, efficient use of the disks suffers and the gains in scheduling flexibility are more than offset by loss of disk bandwidth.

Another variation being pursued is motivated by the presence of zones on todays disks in which the outer and inner cylinders (and their associated tracks) do not contain the same amount of data. The outer tracks contain more data and when being read, the disk bandwidth is higher. To obtain an average bandwidth for each stream it is natural to think of alternating reading outer tracks (big ones) with inner tracks (smaller ones) to obtain an overall average that would be the same for all streams. This motivates the following scheme in which we not only offset the sweep schedule on successive disks but set up the schedules so that if disk i does a $I \rightarrow O$ sweep starting at time t, then disk $i+1$ starts a $O \rightarrow I$ sweep at $t + \Delta$. This is illustrated in Figure 6. The only difference between this and Figure 3 is that the sweep direction alternates from one disk to the next. This simple variation maintains the advantages of the just-in-time scheduling and, in addition, can support an average delivery rate which is the same for all streams.

5 Conclusions

We presented a new storage layout and scheduling strategy with the motivation of reducing the buffering requirements relative to existing proposals. The method proposed can result in a reduction

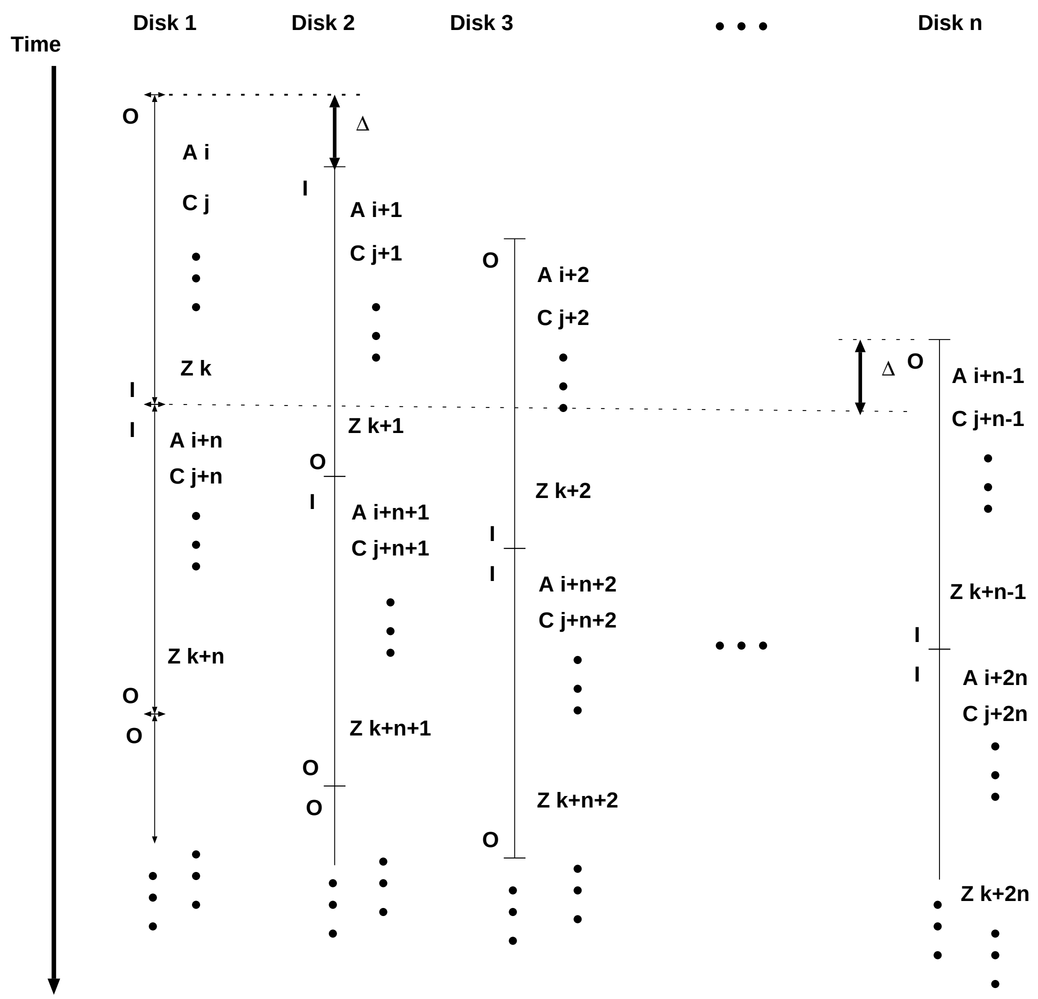

Figure 6: Sweep Scheduling for Zones

of memory requirements by an order of magnitude or more. Detailed simulation results show the large gain in both reduced need for buffer memory and the increased performance attainable with the proposed scheme. These early results indicate that the basic idea has significant potential. Variations on the basic scheme to further reduce waiting times for stream initiation and also to account for disk zones were briefly discussed although work in these ideas is very preliminary at this point.

References

[1] P.V. Rangan adn H.M. Vin. Efficient storage techniques for digital continuous multimedia. *IEEE Transactions on Knowledge and Data Engineering*, august, 1993.

[2] F. Tobagi adn J. Pang, R. Baird, and M. Gang. Streaming raid - a disk array management system for video files. *ACM Multimedia '93*, pages 393–400, 1993.

[3] Steven Berson, Shahram Ghandeharizadeh, Richard R. Muntz, and Xiangyu Ju. Staggered Striping in Multimedia Information Systems. *SIGMOD*, 1994.

[4] Steven Berson, Leana Golubchik, and Richard R. Muntz. A Fault Tolerant Design of a Multimedia Server. *to appear in SIGMOD 1995.*

[5] M. Chen, D. Kandlur, and P. Yu. Optimization of the grouped sweeping scheduling (gss) with heterogeneous multimedia streams. In *ACM Multimedia '93*, pages 145–186, 1993.

[6] Chervenak. *Ph.D. Dissertation, UCB*, 1994.