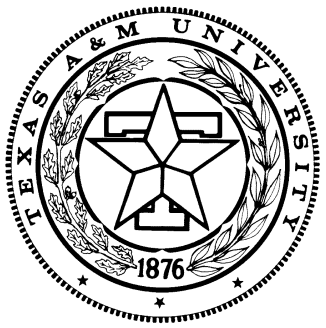
Texas A & M University



Providing deterministic I/O service for VBR streams

A. L. Narasimha Reddy and R. Wijayaratne

TAMU-ECE-9701
Aug. 1997

Computer Engineering Technical Reports
Department of Electrical Engineering
College Station, TX 77843-3128

Providing deterministic I/O service for VBR streams *

A. L. Narasimha Reddy
Ravi Wijayaratne

Dept. of Elec. Engg.
Texas A & M University
214 Zachry
College Station, TX 77843-3128
reddy@ee.tamu.edu

Abstract

In this paper, we investigate mechanisms for providing deterministic service for Variable Bit Rate (VBR) streams at the disk. We propose a scheme for providing deterministic service within the I/O system. We evaluate data layout strategies and present an analysis of the results. We show that smoothing techniques can improve performance. We propose and evaluate several stream scheduling strategies. We show that the stream scheduling policies can have a significant impact on performance. We also quantify the improvements in stream throughput possible by using statistical guarantees instead of deterministic guarantees. We show that stream startup latency can be traded off for improving stream throughput (while retaining deterministic guarantees).

1 Introduction

System level support of continuous media has been receiving wide attention. Continuous media impose timing requirements on the retrieval and delivery of data unlike traditional data such as text and images. Timely retrieval and delivery of data requires that the system and network pay attention to notions of time and deadlines. Data retrieval is handled by the I/O system (File system, disk drivers, disks etc.) and the delivery is handled by the network system (network software and the network). In this paper, we will look at the data retrieval problem.

For continuous media, service can be provided in different ways. *Deterministic service* provides guarantees that the required data will be retrieved in time. *Statistical service* provides statistical guarantees about data retrieval, e.g., 99.9% of the requested blocks will be retrieved in time. Data streams can be classified as Constant Bit Rate (CBR) or Variable Bit Rate (VBR) depending on whether the stream requests the same amount of data in an interval. Disk service tends to incur random service time costs because of the random seek and latency overheads involved in disk transfers. Much work has been done on providing deterministic service for CBR streams [1, 2, 3, 4] and many projects have started building servers for such service [5, 6, 7]. Recently, statistical service for VBR streams received attention [8, 9]. In this paper, we address the problem of providing deterministic service for VBR streams.

Providing deterministic service at disk is complicated by the random seek and latency overheads involved in a disk transfer. This problem has been addressed effectively by suitable disk scheduling policies [10, 11]. These scheduling policies group a number of requests into rounds or batches and service the requests in a round using a disk seek optimizing policy such as SCAN. Then the service time for the entire round can be bounded to provide guarantees. This works well when the request streams are CBR. However, with VBR streams, the workload changes from round to round and hence such an approach will have to consider the variations in load for providing guarantees. Long term behavior of MPEG video streams exhibit normal distribution [8, 12, 13]. This observation can be used to compute the statistical probabilities of workload variations when a number of streams require service at an I/O system. Based on these probabilities, statistical service can be provided for VBR streams.

How to provide deterministic service for VBR streams? This problem has not received much attention and is the focus of this paper. Providing deterministic guarantees for data delivery has received considerable attention in the networking community [14, 15, 16]. A VBR stream is characterized by its worst-case demand on the network over any period of time. This characterization of the stream's behavior is used to calculate the worst-case delays possible through a switch when multiple streams pass through a switch. The service time for a packet through a switch is independent of the order in which the packets are served unlike at a disk where the order of service determines the seek time cost of the disk transfer.

*This work is supported in part by an NSF Career Award and by a grant from State of Texas Higher Education Board.

Similar worst case analysis at the disk is possible but likely to give less optimal results because of the higher seek time costs likely to be paid in any FCFS (First Come First Serve) approach.

Stream smoothing [17] has been proposed to reduce the variability of VBR streams. It has been shown that smoothing reduces the variance of demand and thus enables more number of streams to be supported at a network switch. Does smoothing help in data retrieval as well? Previously proposed disk scheduling policies demonstrated the effectiveness of utilizing large blocks for data retrieval. Typically, these proposals have suggested retrieving data required for 0.25-1 seconds of display [10, 11]. Such a retrieval unit essentially smoothes data retrieval over the duration of the 'round'. Recently it has been shown that smoothing techniques can be applied across the rounds to achieve further reduction in the variability of stream's load on the disks [18].

With CBR streams, it has been shown that if the first block of the stream can be scheduled, then the entire stream can be provided guaranteed service [19]. To provide deterministic service to VBR streams, how does one guarantee that every block can actually be serviced? If such service can be provided, what is the best way to schedule the streams? If the stream is scheduled as soon as it can be scheduled, we can minimize the startup latency for that stream. However, this may hinder the scheduling of streams that may arrive in the future. This tradeoff between startup latency and the stream throughput of the system is investigated. We propose and evaluate a number of *stream scheduling* strategies for the same. This is different from a *disk scheduling* policy used for determining the order in which requests may be served at a disk.

Typically, a system will have to support continuous media along with regular I/O requests. One possible approach for dealing with the variable load due to VBR streams is to utilize the entire disk bandwidth during high loads while limiting the long term utilization to some acceptable limit. This approach essentially lets the regular I/O requests soak up the available I/O bandwidth during any given period while reserving certain fraction of the I/O bandwidth to such requests. However, if the load is very bursty, and if the entire bandwidth is utilized by VBR streams over long periods of time, the regular I/O requests may be starved of service over such a period. We investigate the impact of such an approach on the utilization of resources.

It may be desirable to provide both deterministic service and statistical service to VBR streams in the same system. Deterministic service may be too expensive on the system's resources. A user may request for statistical service when a request for deterministic service may be denied due to lack of resources. How many more streams can be supported when statistical service is requested compared to that of deterministic service? We evaluate this issue as a function of the number of blocks that can be dropped.

Section 2 proposes an approach for providing deterministic service guarantees at disk for VBR streams and the various other related issues. Section 3 presents a number of stream scheduling policies. Section 4 discusses different data layout schemes. Section 5 presents a performance evaluation of these schemes based on trace-driven simulations. Section 6 summarizes our results and points out future directions.

2 Deterministic guarantees

Providing deterministic service for VBR streams is complicated by the following factors: (i) the load of a stream on the system varies from one round to the next, (ii) scheduling the first block doesn't guarantee that the following blocks of the stream can be scheduled. To ensure that all the blocks required by a stream can be retrieved, we can compute the peak rate of the stream and reserve enough disk bandwidth to satisfy the peak requirements of the stream. However, this approach results in considerable overestimate of required resources and results in supporting fewer streams.

It is assumed that the disk scheduling policy employs a round-based SCAN-like seek optimization [10, 11]. In such a policy, disk service is broken into fixed size time units called rounds or batches. Each round may span 0.25-1 seconds of time (based on earlier results in [10]). Requests are grouped into different rounds based on their deadlines. Requests within a round are served using SCAN or such similar seek optimizing policies. In this paper, we assume that the disk employs CSCAN, i.e., disk arm moves forward from the outermost request to innermost request during the round and jumps to the outermost request to start the next round.

In our approach, an application requiring service for a VBR stream supplies the I/O system with a trace of its I/O demand. This could be stored on the disk in the form of a special file or some attribute file. This data could be based on frame rate i.e., given on a frame to frame basis or could be more closely tied to the I/O system. Specifying the load on a frame basis is more flexible and the application doesn't have to be aware of how the I/O system is organized (block size or round size). If the I/O system's block size is known and the duration of each round is known, then the trace can be compacted by specifying the I/O load on a round by round basis in terms of the blocks. This requires less amount of data than the frame by frame basis description. For example, a 40,000 frame trace of the movie "Silence of the Lambs" (24 frames/second) requires 203,285 bytes on a frame by frame basis compared to a 3,333 byte description of the same movie when compacted with the knowledge of the round size of 0.5 seconds and a block size of 8KB. For example, a frame by frame trace may look like 83,888, 9,960, 10,008, 27,044, ...which indicates the number of

bits of data needed to display each frame. If the round size is say 2 frames i.e., 1/12th of a second, and the I/O system uses a block size of 4KB, then the compacted trace would have $\lceil(83888 + 9960)/(4 * 1024 * 8)\rceil = 3$ in the first entry. The second entry would have $\lceil(10008 + 27044) - (3 * 1024 * 8 - 83888 - 9960)/(4 * 1024 * 8)\rceil = 1$ block. Hence, the equivalent compacted trace for the stream would be 3, 1, ... It is assumed that this information is available to the I/O system in either description and we will call this the *demand trace*.

The I/O system itself keeps track of the worst-case time committed for service in each round at each of its disks in the form of a *load trace*. Before a stream is admitted, its demand trace is combined with the load trace of the appropriate disks to see if the load on any one of the disks exceeds the capacity (committed time greater than the length of the round). The load trace of a system consists of load traces of all the disks over sufficient period of time. This requires the knowledge of the placement of blocks of the requesting stream. This information can be obtained from the file system. For example, assume that the demand trace contains 1, 2, 0, and 2 blocks for 4 rounds and that the data is distributed round-robin across the four disks in the system on a block by block basis starting with disk A. Then, the load trace of disk A in round 0 has to have enough room to accommodate retrieval of one more block, similarly disks B and C have to accommodate 1 more block in round 1 and disks D and A have to accommodate 1 more block in round 3. If the stream is distributed on equal time slice basis, i.e., data required during a round is located on a single disk, then, the example stream would require 1 block from disk A in round 1, 2 blocks from disk B in round 2, 0 blocks from disk C in round 3, and 2 blocks from disk D in round 3. Hence, the demand trace of a stream can be combined with the load trace of the system only with the knowledge of the placement of the blocks of that stream.

A stream is admitted if its demand can be accommodated by the system. It is possible that the stream cannot be supported in the round the request arrives. The stream scheduling policy will look for a round in which this stream can be scheduled. Let *load[i][j]* denote the load on disk i in round j. Let the demand of a stream be given by $demand[j]$ indicating the number of blocks to be retrieved by that stream in round j. Then, a stream can be admitted if there exists a k such that $load[i][j + k] + serv\_time(demand[j]) \leq round\ time$, for all j, i = disk storing data for round j, and k is the startup latency $\leq$ *latency target*. In a CDL layout, where multiple disks may store the data required by a stream in a round, the above check needs to be appropriately modified to verify that each of the disks can support retrieval of a single block of data. The function $serv\_time()$ estimates the worst-case service time required for retrieving a given number of blocks from a disk given the current load of the disk. This function utilizes the current load of the disk (number of requests and blocks) and the load of the arriving request to estimate the worst-case time required to serve the new request along with the already scheduled requests. A similar check can be applied against buffer resources when needed.

3 Stream scheduling

The number of streams that can be supported by the server is dependent on stream scheduling policies along with the data storage and retrieval policies. An arriving request may not be scheduled immediately upon its arrival because of the already existing load on the system. By delaying the starting of the stream, we may be able to schedule a stream that may not otherwise be scheduled i.e., the startup latency can be traded off for stream throughput. How should streams be scheduled in a server? By greedily scheduling a stream as early as possible, we may impact the possibility of scheduling other streams in the future. If scheduling a stream immediately after its arrival saturates the capacity of a disk, that disk would be unavailable in that round for any other service and hence can affect schedulability of other streams. This issue is explored by evaluating a number of scheduling strategies.

All the scheduling algorithms discussed below use a *latency target* as a parameter. A stream is said to be unschedulable if it cannot be scheduled within a fixed time interval (specified by the latency target) after the arrival of the request. If a stream arrives at time t, all the slots within the time $(t,\ t + L)$ are considered for scheduling a stream, where L is the latency target. However, the order in which these slots are considered and the criterion for selection among the choices (if any) is determined by the stream scheduling policy. In the policies described below, if the starting point s for scheduling a stream is not t, after reaching $t + L$, the policy wraps around to t and explores the options between t and s.

In *greedy scheduling*, a stream is scheduled as soon as it can be from the time the request arrives at the system. In *random start* policy, a stream is scheduled greedily from a random starting point within the latency target window. In *last scheduled* policy, a stream is scheduled greedily from the scheduled point of the last stream. In *fixed distance* policy, a stream is scheduled greedily from a fixed time away from the last scheduled stream's scheduled point. In *minimal load* policy, stream is scheduled at a point that minimizes the maximum load on any disk in the system. In *prime hopping* policy, instead of serially looking at the time slots from a starting point, slots a prime distance away are considered. For example, if the request arrives at time 0, a random starting point s is chosen. Then rounds, s, $s + p$, $s + 2p$, $s + 3p$ ... are considered until the stream can be scheduled. Since p is prime, all the rounds within the latency target window will be considered.

Latency target impacts stream throughput in two ways. A larger target allows us to search more slots to find a suitable starting point for an arriving stream. A larger target also allows the peaks in the demand trace to be spread out more from each other and thus allowing a future stream to find enough I/O bandwidth to be scheduled.

Stream scheduling problem can be considered in two ways. In the first, given an existing load on the system, can an arriving stream be scheduled without disturbing the guarantees of already scheduled streams? This is the problem we consider in this paper. The scheduling decisions are made one at a time. Another interesting problem arises in capacity planning: can the system support the load of a given set of streams? This problem utilizes the information about all the streams at once to answer the question whether the system can support such a load (with required guarantees)? It can be shown that the stream scheduling problem is closely related to the bin packing problem which is known to be NP-hard [20].

4 Data layout

Data layout plays a significant role on the performance of disk access. It has been suggested by many researchers that video data should be striped across the disks for load balancing and to improve throughput available for a single data stream [5, 19, 6, 7].

Data for a VBR stream can be stored in (i) Constant Data Length (CDL) units (ii) Constant Time Length (CTL) units. In CDL, data is distributed in some fixed size units, say 64KB blocks. In CTL, data is distributed in some constant time units, say 0.5 seconds of display time.

With CDL layout, when data is striped across the disks in the system, data distribution is straightforward since each disk stores a block in turn. With CDL, data will be retrieved at varying rates based on the current rate of the stream. When data rate is high, data is requested more often. The variable rate of data retrieval makes it hard to combine such a policy with round-based seek optimizing policies such as SCAN-EDF [10] or GSS [11].

To make it possible to combine CDL layout with such seek optimizing policies, we consider data retrieval separately from the layout policy. Instead of retrieving one block at a time, display content for a constant unit of time is requested from the I/O system at once. If the data rate is high, this may result in requesting blocks from multiple disks in one time unit. It is noted that the data required for display in a unit of time need not be a multiple of the I/O block size. Since the data retrieval is constrained by the I/O block size of the system, the needed data is rounded up to the next block.

In CTL layout, each disk stores the display content of the stream in a unit of time. Again, we note that the amount of data needed by the stream for display in a unit of time need not be a multiple of the I/O block size. In this paper, the data layout and retrieval is assumed to be constrained by the I/O block size. This is termed Block-constrained CTL data layout or BCTL in this paper. In BCTL, data distribution is harder since the amount of data stored on each disk depends on the data rate in that round and hence varies from disk to disk. However, data retrieval in a round-based scheme is easier since the data required by the stream in a round is located on a single disk. These issues have been addressed in [21, 9] and the BCTL data layout is termed the hybrid scheme in [9]. The discussion of CTL, CDL is analogous to segment-based or page-based virtual memory discussion.

BCTL data layout requires interaction of the application and the file system. The application has to pass the information about the data rate of the stream to the file system. This information will then be used by the file system to determine how many blocks need to be stored on each disk while striping the data across the disks. With CDL data layout, the retrieval application doesn't have to interact with the file system while storing the data on the disk. BCTL and the CDL data layouts are considered with a round-based data retrieval policy.

4.1 Performance Analysis of CDL and BCTL schemes

The total access cost of data retrieval in both the data layout schemes is compared to draw conclusions about their relative performance. If a stream retrieves n blocks in m rounds, in the BCTL scheme, a maximum of m disk accesses are made and in the CDL scheme, n disk accesses are made. The reading time or transfer time is the same in both the schemes since same number of blocks are read in both the cases. Hence, we need only look at the seektime and the latency time of a disk access.

A piecewise linear approximation of seektime function of the disk is used. This function will overestimate the seektime cost of an access but it is easier to compute than the actual non-linear seektime function. Typically, the disk manufacturers publish only three values on the seektime function, zero track seek time (or head switching time), average seek time (seek time for 1/3 of the tracks) and maximum seek time (seek time for a disk sweep). Based on this information, we can construct a piecewise linear seektime function that has two fragments (from 0 to 1/3 tracks and 1/3 tracks to total number of tracks). To accommodate for the nonlinearity and to accommodate for the worst-case estimates needed for deterministic service, we will have to adjust the seektime value for 1/3 tracks. Let's assume such a piecewise linear function is constructed with s_0, s_{av} and s_{max} as the three points on the curve.

It can be shown that worst-case seektime is obtained when the requests are equally spaced on the disk surface. In the worst-case scenario, the seektime cost per k requests is bounded by k * seektime cost of

(T/k) tracks. When the disk serves 3 or more requests, we can use the linear approximation of the seektime function from 0 to T/3 tracks to bound the total seektime by $\leq$ k * $(s_{av} - s_0)$*(T/k)/(T/3) + s_0.

Hence seektime $\leq 3 * s_{av} + k * s_0$. This shows that the total seektime cost in both schemes differs only in the second term as long as the requests are served in a SCAN order in each round.

In CDL, since each block goes to a different disk, the rotational latency is bounded by $n * rot.\ latency$. In BCTL, the rotational latency is bounded by $m * rot.\ latency$. Since $n \geq m$, the total cost of service in CDL is at least as much as in other schemes. This is based on the assumption that the blocks retrieved by a stream in a round are contiguously located on the disk.

When we consider the total access cost over the m rounds, the CDL scheme requires $(n - m) * (rot.latency + s_0)$ over the BCTL scheme. This analysis shows that if SCAN like seek optimizing policies are employed, the primary determinant of performance is the rotational latency (since $rot.\ latency > s_0$). when 3 or more streams are served at the disk in a round.

5 Performance Evaluation

5.1 Simulations

We evaluated a number of the above issues through trace-driven simulations. A system with 8 disks is simulated. Each disk is assumed to have the characteristics of a Seagate Barracuda drive [22]. The disk drive characteristics are shown in Table 1. Each disk in the system maintains a load table that depicts the load of that disk into the future. An arriving stream presents its demand trace to the simulator. Based on the data layout strategy employed, the stream's demand trace is combined with the load trace of the system. The starting point of the stream and the order in which candidate slots are tried for admitting a stream is determined by the stream scheduling policy. CDL and BCTL data layout strategies are considered. A number of stream scheduling policies are explored. Data block size on the disk is varied form 32KB to 256KB. In BCTL, it is assumed that each stream pays a latency penalty at a disk. In CDL, a stream pays at most one latency penalty at each disk per round. For example, if the stream requires 1 block each from disks 1, 2 and 3, then that stream pays a latency penalty at disks 1, 2 and 3 in that round. If the stream requires 10 blocks in a round from the 8 disks in the system, it pays a latency penalty at each disk. This is based on the assumption that the blocks retrieved in a round for a stream are stored contiguously on the disk.

Stream scheduling policies do not alter already scheduled streams to admit a new stream. Also, when requests arrive at the same time, admission decisions are made one stream at a time. All the stream requests are assumed to arrive in time slot 0. If the stream requests are assumed to arrive randomly over time, more streams can be admitted. However, to study the worst-case scenario, we assumed that all the requests arrive at once. The simulator tries to schedule as many streams as possible until a stream cannot be scheduled. The number of streams scheduled is the stream throughput. This process is repeated for all the scheduling policies and block sizes. Four different video streams are considered in our study as explained below.

Table 1. Disk characteristics.

Parameter	Value
Zero Seek time	0.85 ms
Avg. Seek time	9.0 ms
Max. Seek time	18.0 ms
Min. Transfer rate	11.5 MB/s
Max. Transfer rate	17.5 MB/s
Full Rev. time	8.33 ms
Num. cylinders	6311

Table 2. Characteristics of traces.

Stream Name	Mean KB/sec	Sta. Dev.
Lambs	171.32	58.33
Term	255.54	50.92
News	484.31	108.86
Asterix	523.79	124.50

5.1.1 Traces

MPEG traces from University of Wuerzburg [23] were used in this study. From the frame by frame trace of the movie, we constructed several versions of the demand trace for each movie. For this study, we used four separate MPEG traces. These traces are named *Lambs* (for a segment of the movie "Silence of the lambs"), *Term* (for a movie segment of the movie "Terminator"), *News* (for a news segment trace), *Asterix* (for a segment of Asterix cartoon). Each trace contained 40,000 samples at a frame rate of 24 frames per second. A mixed workload based on these traces is also constructed. During simulations, with equal probability, one out of the four traces is selected for scheduling i.e., the workload consisted of a random mix of these four traces with each tracing being selected with an equal probability. Each trace has a different bit rate and different mean and variance characteristics and these are shown in Table 2. The traces obtained were of fairly low bit rates. To simulate better quality streams, we assumed that the bit trace is actually a byte trace without changing the relative weights of the frames.

A block size of 32 KB, 64KB, 128KB or 256 KB and 0.5 seconds of round time are used to convert the frame trace into a compact demand trace for each

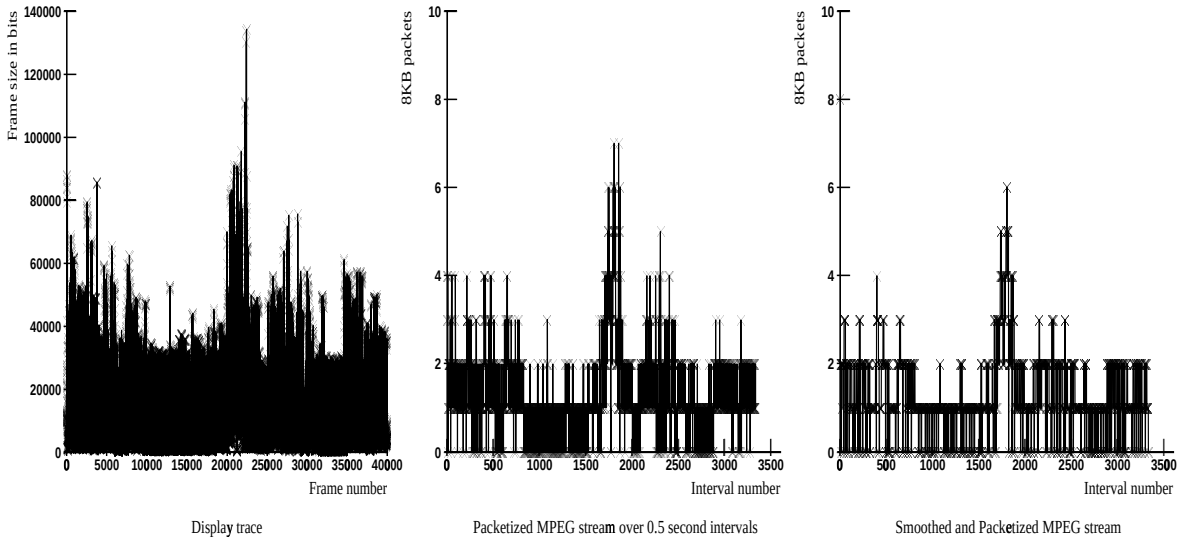


Fig. 1. Effects of smoothing on a video stream.

movie segment. With the choice of four block sizes, we get four different compact demand traces for each stream. These four different traces are used to study the impact of the block size on the results. Next we smoothed the trace of each stream by assuming that a memory buffer of 8 blocks is available and then converted the smoothed trace into a compact demand trace in terms of blocks. For each block size, we have an unsmoothed demand trace and a smoothed demand trace for a total of 8 separate traces for the 4 block sizes considered in this study. Fig. 1 shows an example of the resulting two traces of Lambs at a block size of 64KB and a round of 0.5 seconds. Smoothing reduces the peak demand of the stream and also reduces the variance of the stream's demand trace.

In simulations, it is assumed that the first block of each data stream is stored on a different disk. Data is striped across the eight disks in a round-robin order based on either CDL or BCTL data layout policy.

5.2 Results

Fig. 2 shows the impact of block size, data layout strategy and smoothing on the stream throughput of the four different data streams separately. The results shown in Fig. 2. are obtained with random-start stream scheduling policy. For the block sizes considered, the BCTL data layout strategy performs better than the CDL strategy. As explained earlier, when more than 3 streams are served at the disk using a seek optimizing strategy, rotational latency primarily determines the performance. Since CDL results in paying a rotational latency penalty for each block compared to each time slice in BCTL, CDL performs worse than BCTL.

As the block size is increased a stream fetches less number of blocks in a round and hence CDL tends to be more efficient at larger block sizes. The stream throughput for CDL improves significantly for all the data streams as the block size is increased from 32 KB to 256 KB. At smaller block sizes, the difference in performance between BCTL and CDL is larger and as the block size is increased, the difference decreases. At 256KB block size, both schemes achieve nearly the same stream throughput for Lambs and Term workloads. For these two data streams, at this block size, data retrieval requires more than a single block in very few rounds and hence the performance is nearly the same whether CDL or BCTL strategy is adopted. However, in Asterix and News, data rates are high and even at a block size of 256KB, considerable number of rounds require multiple block retrievals. As a result, we find significant difference in performance even at a block size of 256KB. The stream throughput drops slowly for BCTL as the block size is increased. This is due to effects of larger quantization of service allocation for a request.

Smoothing improves performance with both data layout strategies. This is as expected since smoothing reduces the variability of demand which in turn helps in scheduling more streams at the I/O system. Smoothing can improve stream throughput by up to 17% for these four workloads.

Fig. 3. shows the stream throughput achieved when a mixed workload is considered. Similar performance trends were observed even when a mixed workload is considered. Performance of the CDL scheme improves as the block size is increased. Smoothing improved performance in both data layouts. CDL performs worse than BCTL, but the difference in performance reduces as the block size is increased.

From figures 2 and 3, BCTL is seen to offer better

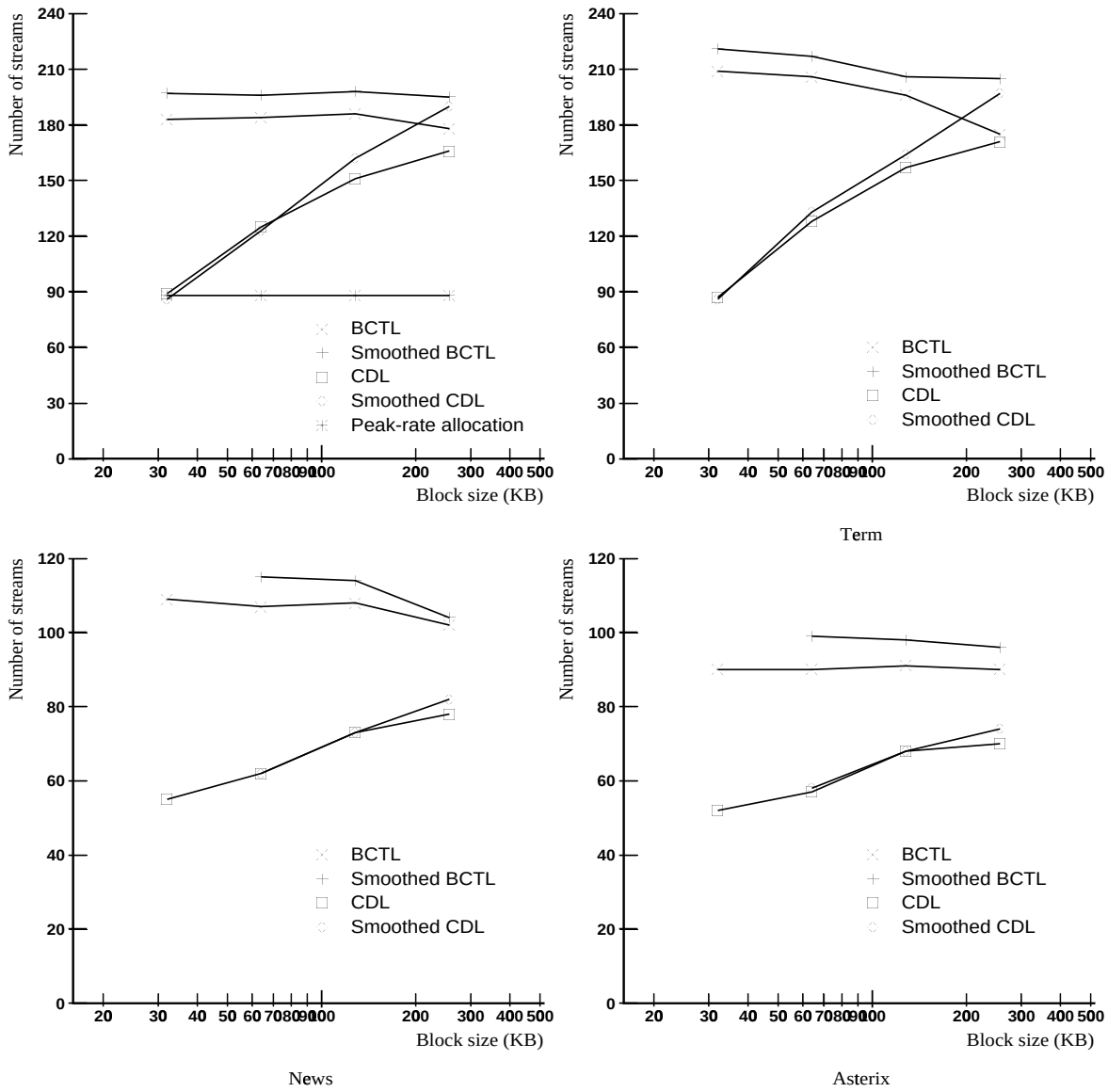


Fig. 2. Impact of smoothing and data layout.

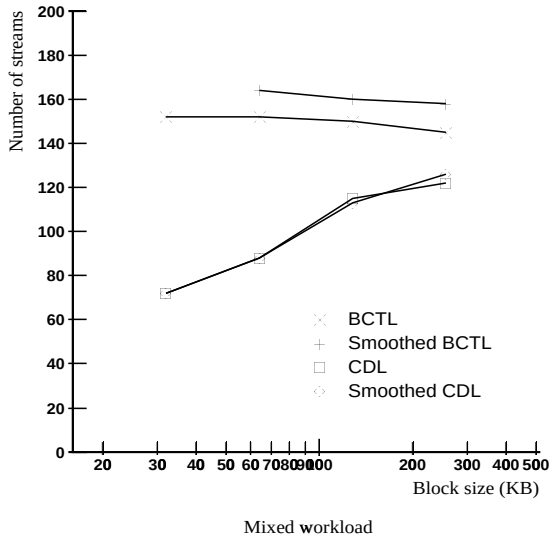


Fig. 3. Results of mixed workload.

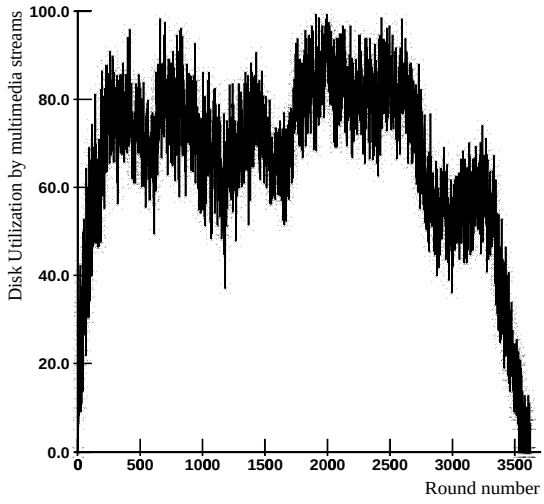


Fig. 4. Disk utilization by VBR streams.

performance over CDL approach. However, at larger blocks, the performance differences are less significant. CDL approach is more flexible since it doesn't tie data distribution closely with the application characteristics (data consumption rate in each frame). To obtain good performance out of CDL, the block size should be large enough such that multiple blocks of data are not requested by a stream in most rounds. If the system supports streams at different data rates, then the block size and round size should be chosen such that the stream with highest data rate would retrieve one block in most rounds. This suggests employing larger block sizes than previously suggested by the disk scheduling policies [10].

Fig. 4. shows the disk utilization by the video streams as a function of time. The figure shows the load at one of the eight disks in the system with a mixed workload and *minimal load* stream scheduling policy. Even though the average utilization is 66%, for several seconds, the disk is nearly 100% busy between rounds 1800 and 2600. If we allowed the video streams to occasionally utilize the full 100% I/O bandwidth of the system while maintaining the average utilization below say 65%, the aperiodic requests could get starved for service for long periods of time (in this case for 400 seconds). Hence, this is unacceptable in a system that has to support both types of requests. This result shows that the system has to reserve certain minimum amount of I/O bandwidth for aperiodic requests to provide reasonable response to those requests.

Fig. 5. shows the impact of stream scheduling policies on the stream throughput at various block sizes. The results in Fig. 5. are for a mixed workload and a latency target of 300 rounds. Greedy policy achieves the least stream throughput in both data layout schemes. It is observed that the minimal load policy achieves high stream throughput consistently in BCTL data layout. However, minimal load policy doesn't perform as well with CDL data layout. Prime hopping, fixed distance and random start achieve nearly the same stream throughput. Minimal load policy achieves on an average 15% better stream throughput than these three policies with BCTL layout. It is observed that stream scheduling policy has a significant impact on performance. For example, minimal load policy improves performance by about 80% compared to greedy policy at a block size of 32KB in BCTL data layout. This shows the importance of studying the stream scheduling policies.

Fig. 6. shows the startup latencies achieved by different stream scheduling policies. Greedy policy, by its nature, achieves the smallest startup latency. However, as observed earlier, it also results in lower stream throughput. The policies based on randomness, random start, prime hopping and fixed distance achieve average startup latencies close to 150, which is half of the latency target of 300 rounds considered for these results. Minimal load and Last scheduled achieve better latencies than these policies for both the data layouts. Based on achieved stream throughput and startup latencies, minimal load and last scheduled are better candidates than the other policies. Last scheduled has a more consistent performance across the two data layout strategies. (*Note to reviewers:* Minimal load seems to be getting stuck close to the starting point with CDL data layout and we are investigating ways to improve this policy for CDL data layout.)

Fig. 7. shows the impact of latency target on the stream throughput of mixed workload. Minimal load, prime hopping, fixed distance, and random start continue to achieve higher throughput at different latency targets compared to the greedy policy. As

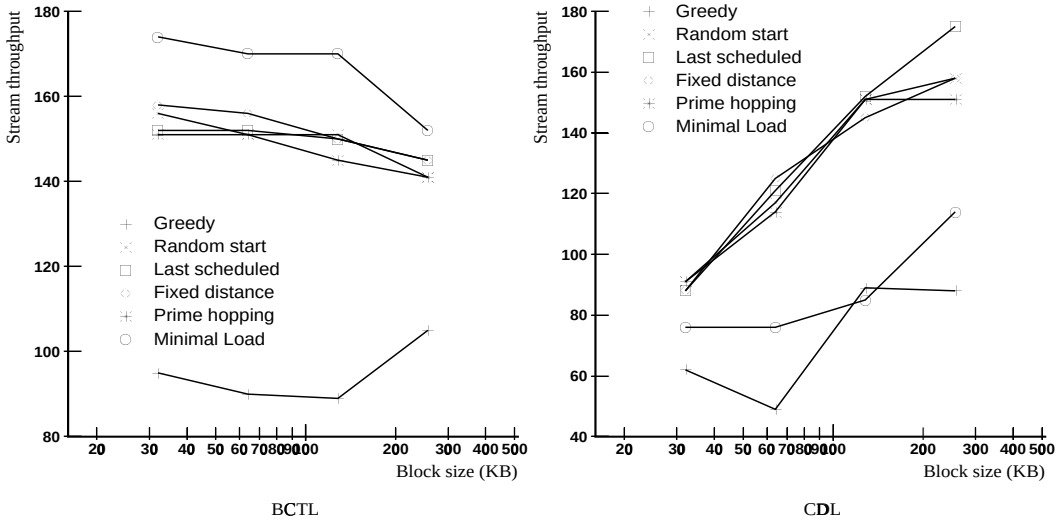


Fig. 5. Impact of stream scheduling.

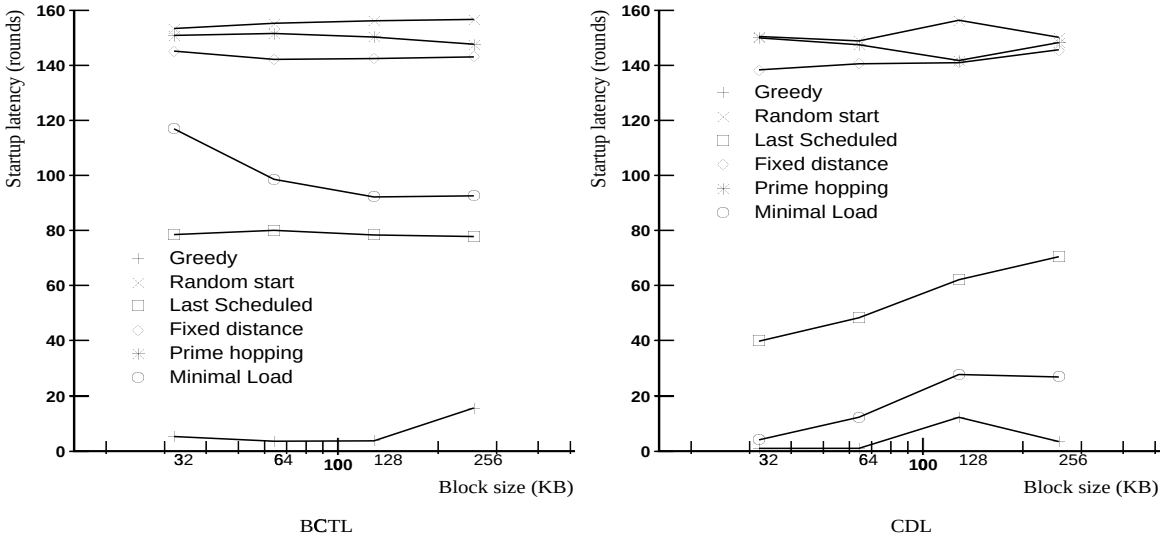


Fig. 6. Average startup latency.

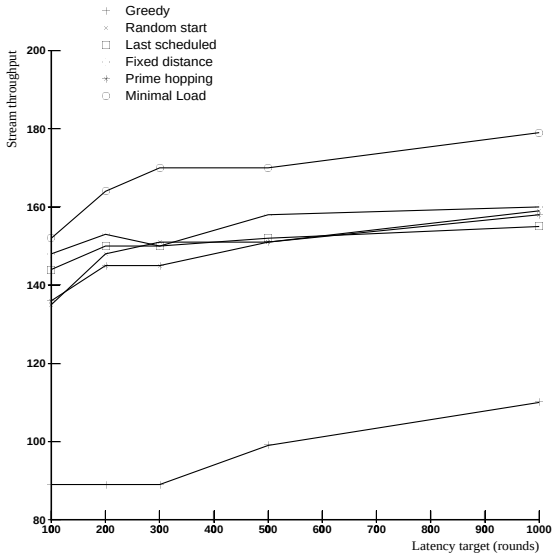


Fig. 7. Effect of latency target.

the latency target is increased, the stream throughput increases. This suggests that startup latency can be traded off for improving the stream throughput. This provides an alternative to dropping blocks when higher stream throughput is needed.

Fig. 8. shows the impact of statistical guarantees on stream throughput. Instead of requiring that every block of data be retrieved in time, we allowed a fraction of the blocks for each stream to miss deadlines or not be provided service. This fraction is varied among 0.1%, 0.2%, 0.5%, 1.0%, 2.0% and 5.0% at various latency targets. It is observed as more blocks are allowed to be dropped, it is possible to achieve more throughput compared to deterministic guarantees. Stream throughput can be improved by up to 20% by allowing 5% of the blocks to be dropped. Dropping blocks is more effective at lower latency targets than at higher latency targets. For example, dropping up to 2% of the blocks improves the stream throughput by 14.5% at a latency target of 100 rounds compared to an improvement of 6% at a latency target of 1000 rounds.

Fig. 8. also shows the tradeoffs possible between latency targets and the number of blocks allowed to be dropped. At a latency target of 100 rounds, 152 streams can be provided deterministic service. To achieve higher stream throughput, we can either increase the latency target or allow blocks to be dropped. For example, when we increase the latency target to 1000 rounds, 179 streams could be scheduled without dropping any blocks. However, to achieve the same throughput at a latency target of 100 rounds, more than 2% of the blocks have to be dropped. Hence, desired throughput can be achieved either by allowing larger latency targets or by allowing a fraction of the blocks to be denied service.

Fig. 8. also shows the impact on the average disk utilizations as a function of the statistical allowances and latency targets. As higher statistical allowances are made, the disk utilizations are improved as more number of streams are supported. As latency targets are increased from 100 rounds, average disk utilizations first increase and then decrease to lower levels. As latency targets are increased, an arriving stream finds more choices to find a suitable starting spot to utilize the available bandwidth. However, as the latency targets are increased further, the streams are scheduled farther and farther into the future and hence result in decreasing disk utilizations.

Usually considered statistical guarantees of 99% (i.e., dropping 1% of blocks) did not provide significant improvements in stream throughput compared to deterministic guarantees. In our measurements, the improvements were less than 6% for all the latency targets except 100 which achieved an improvement of 11%. We observed that most of the dropped blocks tended to be dropped in a small range of time. Even though a 30 minute movie drops only a few blocks during its playback, the dropped blocks tended to be clustered over a 2 minute window instead of over the entire movie. This observation that the dropped blocks tended to be clustered together over shorter time windows leads us to the following idea. It may be possible to drastically improve the stream throughput of a popular movie if the blocks during its peak demand slots are stored in memory. By storing only a few blocks instead of the entire movie, we require less memory. If the movie is popular, several streams will get benefit out of storing these blocks in memory. These ideas will be explored in the future work.

Fig. 9. shows the impact of arrival rates over number of streams scheduled and the average disk utilization. As expected, as the inter arrival time increases, more streams can be scheduled. However, the average disk utilization decreases with increased inter-arrival times.

6 Conclusions and Future work

In this paper, we provided a comprehensive evaluation of several issues in providing deterministic I/O service to VBR streams. BCTL provides better performance than CDL and the main factor contributing to the differences in performance is the cost of rotational latency penalty. Larger block sizes improved performance significantly with the CDL data layout while larger blocks reduced performance slightly with the BCTL data layout. We argued that CDL data layout provides more flexibility at the system level. Several stream scheduling strategies were presented and evaluated. Several of these proposed policies are shown to improve performance significantly compared to the greedy policy. We also showed that startup latency is an effective tradeoff parameter for improving stream throughput. For the workloads considered,

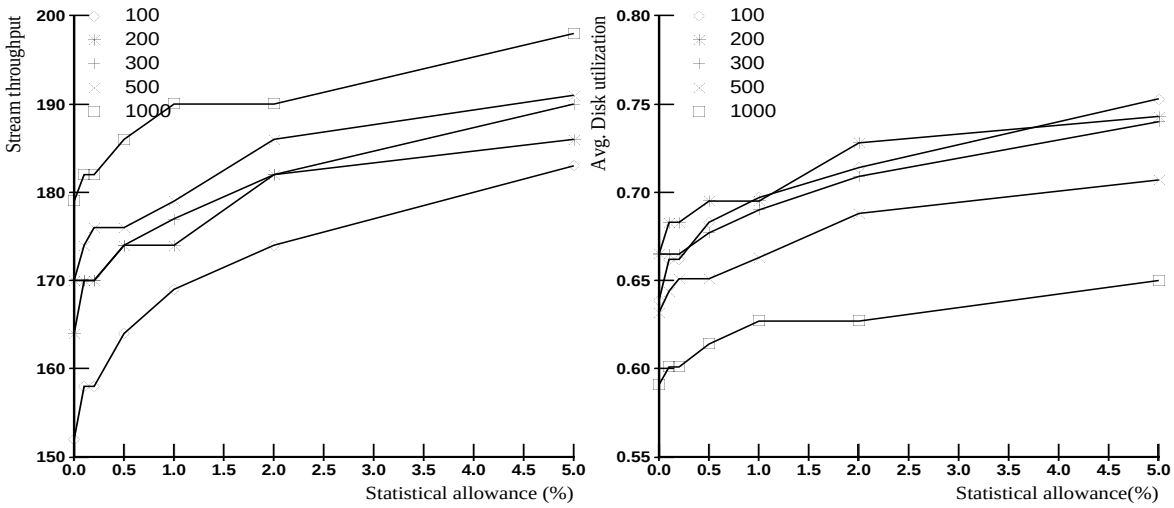


Fig. 8. Effect of statistical allowances.

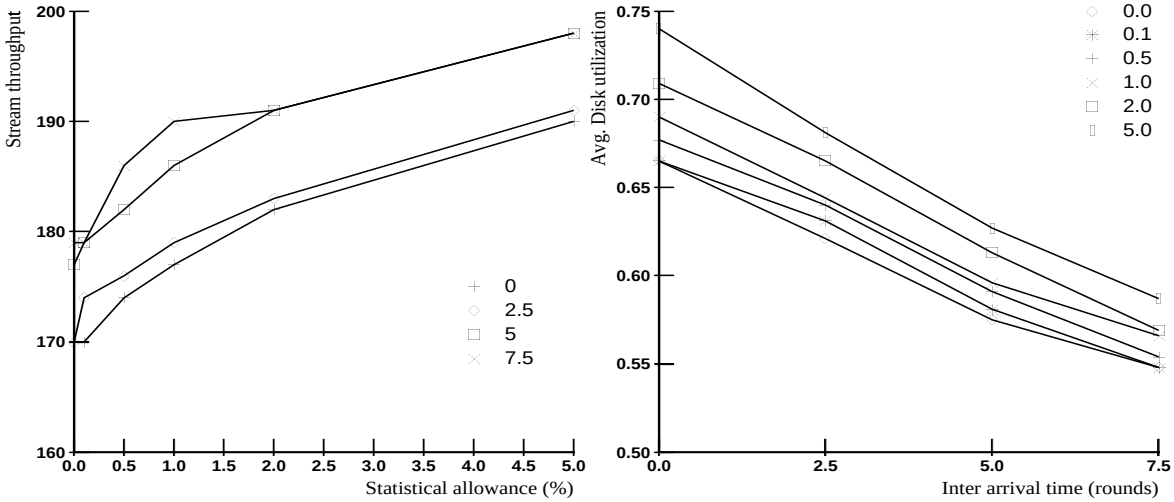


Fig. 9. Impact of arrival rates.

dropping 1% of the blocks did not provide significant improvement in stream throughput because of the load concentration over shorter periods of time. Our results also showed the need for reserving certain minimum I/O bandwidth for aperiodic requests.

The presented results indicate that stream scheduling is likely to be a significant factor in providing service to VBR streams. We are exploring more refined approaches to stream scheduling and data layout. Some of the other policies we are studying include: minimizing the load imbalance caused by the admission of a new stream, adapting data layout (instead of round-robin distribution across the disks) to a bin-packing type of approach to balance a stream's demand across the disks.

References

[1] H. M. Vin and P. V. Rangan. Designing file systems for digital video and audio. *Proc. of 13th ACM Symp. on Oper. Sys. Principles*, 1991.

[2] D. Anderson, Y. Osawa, and R. Govindan. A file system for continuous media. *ACM Trans. on Comp. Systems*, pages 311–337, Nov. 1992.

[3] C. Martin, P. Narayanan, B. Ozden, R. Rastogi, and A. Silberschatz. The Fellini multimedia storage server. *in Multimedia Information Storage and Management, Ed: S.Chung, KluwerPublishers*, 1996.

[4] D. J. Gemmell and S. Christodoulakis. Principles of delay-sensitivie multimedia storage and retrieval. *ACM Trans. on Info. Systems*, pages 51–90, 1992.

[5] R. Haskin. The shark continuous-media file server. *Proc. of IEEE COMPCON*, Feb. 1993.

[6] Microsoft. The tiger video server. *Microsoft Press Release*, Apr. 1994.

[7] A. Laursen, J. Olkin, and M. Porter. Oracle media server: providing consumer based interactive access to multimedia data. *Proc. of SIGMOD*, pages 470–477, 1994.

[8] H. Vin, P. Goyal, and A. Goyal. A statistical admission control algorithm for multimedia servers. *Proc. of ACM Multimedia*, Nov. 1994.

[9] E. Chang and A. Zakhor. Scalable video data placement on parallel disk arrays. *Proc. of SPIE Symp. on Elec. Imaging Sci. and Tech.*, Feb. 1994.

[10] A. L. Narasimha Reddy and Jim Wyllie. Disk scheduling in a multimedia I/O system. *Proc. of ACM Multimedia Conf.*, Aug. 1992.

[11] P. S. Yu, M. S. Chen, and D. D. Kandlur. Grouped sweeping scheduling for dasd-based multimedia storage management. *Multimedia Systems*, 1:99–109, 1993.

[12] R. T. Ng and R. Dilek. Statistical modeling and buffer allocation for MPEG streams. *in Multimedia Information Storage and Management, Ed: S. M. Chung, Kluwer Academic Publishers*, 1996.

[13] S. Gupta, R. Wijayaratne, and A. L. Narasimha Reddy. Impact of MPEG stream behavior on multimedia server design. *Tech. rep. in preparation*, Apr. 1997.

[14] D.E. Wrege, E. W. Knightly, H. Zhang, and J. Liebeherr. Deterministic delay bounds for VBR video in packet-switching networks:fundamental limits and practical trade-offs. *IEEE/ACM Trans. on Networking*, pages 352–362, June 1996.

[15] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated service networks: The multiple node case. *Proc. IEEE INFOCOMM'93*, pages 521–530, Mar. 1993.

[16] R. Cruz. A Calculus for Network Delay, Part I: Network elements in isolation. *IEEE Trans. on Information Theory, vol.37, no.1*, pages 114–131, Jan. 1991.

[17] J. Salehi, Z. L. Zhang, J. Kurose, and D. Towsley. Supporting stored video: reducing rate variability and end-to-end resource requirements through optimal smoothing. *Proc. of ACM SIGMETRICS*, May 1996.

[18] S. Sahu, Z. L. Zhang, J. Kurose, and D. Towsley. On the efficient retrieval of VBR video in a multimedia server. *Proc. of IEEE Int. Conf. on Multimedia Computing and Systems*, Jun. 1997.

[19] A. L. N. Reddy. Scheduling and data distribution in a multiprocessor video server. *Proc. of Int. Conf. on Multimedia Computing and Systems*, pages 256–263, 1995.

[20] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. *W.H.Freeman & Co., New York, NY*, 1979.

[21] H. Vin, S. S. Rao, and P. Goyal. Optimizing the placement of multimedia objects on disk arrays. *Proc. of IEEE Conf. on Multimedia Computing and Systems*, pages 158–165, May 1995.

[22] Seagate Corp. Disk drive product info. *http://www.seagate.com/*, 1997.

[23] O. Rose. Mpeg trace data sets. *ftp-info3.informatik.uni-wuerzburg.de*, 1995.