Providing QOS guarantees for disk I/O *

Ravi Wijayaratne[1]
A. L. Narasimha Reddy

[1]Dept. of Comp. Sci.
Dept. of Elec. Engg.
Texas A & M University
214 Zachry
College Station, TX 77843-3128
{ravi,reddy}@ee.tamu.edu

Abstract

In this paper, we address the problem of providing different levels of performance guarantees or quality-of-service (QOS) for disk I/O. We classify disk requests into three categories based on the provided level of service. We propose an integrated scheme that provides different levels of service in a single system. We propose and evaluate a mechanism for providing deterministic service for Variable Bit Rate (VBR) streams at the disk. We will show that through proper admission control and bandwidth allocation, requests in different categories can be ensured of QOS guarantees without getting impacted by requests in other categories. We evaluate the impact of scheduling policy decisions on the provided service. We also quantify the improvements in stream throughput possible by using statistical guarantees instead of deterministic guarantees.

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

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.

Different levels of service can be provided for continuous media. *Deterministic service* provides guarantees that the required data will be retrieved in time. *Statistical service* provides statistical guarantees about data retrieval, e.g., 99% 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.

Much work has been done on providing deterministic service for CBR streams [1, 2, 3, 4, 5, 6] and many projects have started building servers for such service [7, 8, 9]. Recently, statistical service for VBR streams received attention [10, 11]. Much of this work concentrates on providing service guarantees required by continuous media in the context of Video-on-demand (VOD) servers. An evaluation of tradeoffs in a media-on-demand server can be found in [12]. This paper addresses the problem of providing different levels of service for different classes of requests in a single system.

A storage system will have to support requests with different performance requirements based on the application needs. Continuous media applications may require deterministic performance guarantees i.e., guarantee that a requested block will be available within a specified amount of time continuously during the application's execution. A request from an interactive game or a request to change the sequence of frames in a continuous media application may require that the request have low response time i.e., may require a latency guarantee. A regular file request may only require best-effort service but may require that a certain number of requests be served in a given time i.e., may require a throughput guarantee. 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. There is a clear need for supporting multiple levels of performance guarantees within the storage system. Several interesting questions need to be addressed when multiple levels of QOS need to be supported in the same system: (a) how to allocate and balance resources for the different QOS levels, (b) how to control and limit the usage of resources to allocated levels, (c) how to schedule different requests to meet the desired

performance goals, (d) how do system level parameters and design decisions affect the different types of requests and (e) how to tradeoff performance goals for higher throughput (for example, how much throughput gain can be had with statistical guarantees rather than deterministic guarantees)?

Providing deterministic service at the disk is complicated by the random service time costs involved in disk transfers (because of the random seek and latency overheads). This problem has been addressed effectively by suitable disk scheduling policies [13, 14]. 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 strategy works well with CBR streams. 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.

How to provide deterministic service for VBR streams? This problem has not received much attention and is addressed in this paper. Providing deterministic guarantees for data delivery has received considerable attention in the networking community [15, 16, 17].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. 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. How to schedule service at the disk to provide guarantees while allowing seek optimization?

Data layout policies can impact the performance of the system. Continuous media streams request disk service at regular intervals. If the data requested in an interval is entirely located on a disk, only one disk needs to be involved in the service. This may however result in variable size block allocation on disk since VBR streams request different amounts of data in different time periods. If data is distributed on a block basis, one block request may require service from several disks when the data requirement is large. Constant block allocation is however easier to implement.

In this paper, we will present an integrated scheme for providing different levels of performance guarantees. We propose a method for providing deterministic guarantees for VBR streams that exploits statistical multiplexing of resources. We will look at other issues such as data layout, and disk scheduling in providing performance guarantees.

Section 2 discusses our approach for providing different levels of QOS in a single system. Section

2 also proposes a method for providing deterministic service for VBR streams that allows exploitation of statistical multiplexing across many request streams. Section 3 discusses our scheduling algorithm for providing integrated service and also looks at other issues such as data layout. Section 4 presents a performance evaluation of these schemes based on trace-driven simulations. Section 5 summarizes our results and points out future directions.

2 Multiple levels of QOS

In this paper, we consider three different categories of requests. *Periodic* requests require service at regular intervals of time. Periodic requests model the behavior of video playback where data is retrieved at regular intervals of time. Periodic requests can be either CBR or VBR. *Interactive* requests require quick response from the I/O system. Interactive requests can be used to model the behavior of change-of-sequence requests in an interactive video playback application or the requests in an interactive video game. These requests arrive at irregular intervals of time. *Aperiodic* requests are regular file requests. In this paper, we consider (a) deterministic or statistical guarantees for periodic requests, (b) best-effort low response times for interactive requests and (c) guaranteed minimum level of service or bandwidth guarantees for aperiodic requests.

Our approach to providing QOS guarantees at the disk is shown in Fig. 1. Disk bandwidth is allocated appropriately among the different types of requests. Periodic requests and interactive requests employ admission controllers to limit the disk utilization of these requests to the allocated level for these requests. To provide throughput guarantees for aperiodic requests, we limit the allocated bandwidth for periodic and interactive requests ($< 100\%$) through admission control. Aperiodic requests utilize the remaining disk bandwidth. This guarantees that the aperiodic requests do not get starved of service and receive some minimum level of service.

The admission controllers employed for periodic requests and interactive requests depend on the service provided for these requests. In the next section, we discuss how to provide deterministic service for periodic requests. The proposed approach can be modified to implement statistical guarantees for periodic requests as well. Interactive requests are treated as high-priority aperiodic requests in our system. The scheduler and the admission controller are designed to provide low response times for these requests. We use a leaky-bucket controller for interactive requests. A leaky bucket controller controls the burstiness of interactive requests (by allowing only a specified number of requests in a given window of time) and thus limits the impact it may have on other requests.

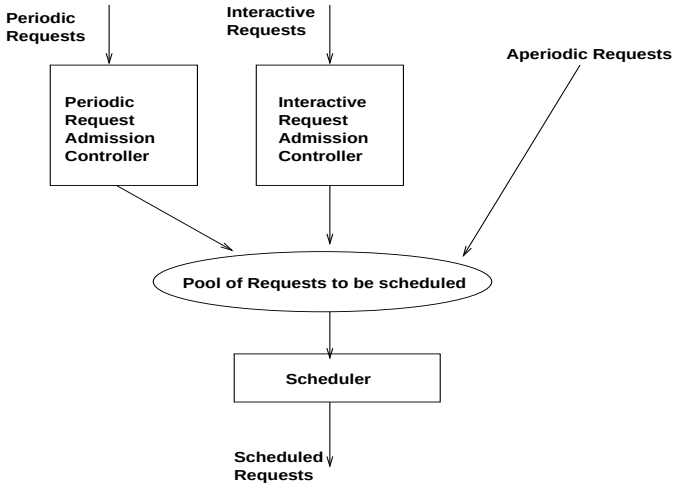

Fig. 1. Supporting multiple QOS levels.

2.1 Deterministic guarantees for VBR streams

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.

Resource allocation based on the peak demands of the stream will underutilize the disk bandwidth since the peak demand is observed only for short durations compared to the length of the duration of the stream. However, when many streams are served in the system, the peaks do not necessarily overlap with each other and it may be possible to serve more steams than what is allowed by the peak-rate allocation. Can we exploit this statistical multiplexing to increase the deterministic service provided by the system? We propose an approach that allows the system to exploit statistical multiplexing while providing deterministic service.

Disk service is broken into fixed size time units called rounds or batches. Each round may span 0.25-1 seconds of time ([13]). 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 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. 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, ... 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. 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 storage volume manager.

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. We will assume that a stream will wait for a maximum amount of time, given by *latency target*, for admittance. 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, where i = disk storing data for round j, and k is the startup latency $\leq latency\ target$. If multiple disks may store the data required by a stream in a round, the above check needs to be appropriately modified to verify that these disks can support the retrieval of needed 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. The demand trace of the application may include the extra block accesses needed for metadata.

Given a latency target L and the length of the demand trace d, the admission controller requires at most Ld additions to determine if a stream can be admitted. In the worst case, for each starting round, the admission controller finds that the very last block of the stream cannot be scheduled. On an average, the admission controller requires less computation per stream. If necessary, latency targets can be reduced to limit the time taken by the admission controller.

The impact of *stream scheduling* policies has been studied in a related paper [18]. It has been shown that stream scheduling policy can have significant impact on the number of streams supported by the storage system. We will present results based on a greedy scheduling policy which searches sequentially starting from a random location within the latency target window. This is an online scheduling policy that does not disturb already scheduled streams. Related offline algorithms are studied in storage system capacity planning [19].

2.2 Latency and bandwidth guarantees

Aperiodic requests are provided bandwidth guarantees by restricting the periodic and interactive requests to certain fraction of the available bandwidth. The admission controller for periodic and interactive requests enforce the bandwidth allocations. Aperiodic requests utilize the remaining I/O bandwidth. If periodic and interactive requests cannot utilize the allocated bandwidths, aperiodic requests are allowed to utilize the available bandwidth to improve the response times for aperiodic requests. In our system, bandwidths are allocated statically to different types of requests.

Latency guarantees are provided by the disk scheduler. We will describe in the next section how the scheduler provides low response times for interactive requests. We modeled the arrival behavior of interactive requests by a random exponential arrival process. When requests arrive randomly, a burst of requests can possibly disturb the guarantees provided to the periodic requests. To avoid this possibility, interactive requests are controlled by a leaky-bucket controller which controls the burstiness by allowing only a certain maximum number of interactive requests served in a given time window. For example, when interactive request service is limited to, say, 5 per second, the leaky-bucket controller will ensure that no more than 5 requests are released to the scheduler irrespective of the request arrival behavior. Hence, an interactive request can experience delay in the controller as well as at the scheduler for service. If sufficient bandwidth is allocated for these requests, the waiting time at the controller will be limited to periods when requests arrive in a burst. We will

describe below how the scheduler provides short latencies for these requests.

3 Scheduling and other issues

3.1 Scheduling for multiple QOS levels

Disks typically employ seek optimization techniques to minimize the seek costs in serving requests. Deadline scheduling and such similar real-time requests provide real-time guarantees but may not be efficient because of associated higher seek costs. It has been shown that disk scheduling strategies that allow seek optimizations while providing delivery guarantees can provide better throughput than strict real-time scheduling policies [14, 13, 5, 3]. Most of this work considered seek optimizing policies that can provide delivery guarantees required by continuous media in Video-on-demand (VOD) and such similar servers. Some of these studies [13, 3] have considered scheduling policies for multiple levels of service with CBR streams. A file system that provides guarantees for periodic CBR streams is implemented in RT-Mach [6]. Our work considers VBR streams and also considers interactive requests in addition to periodic and aperiodic requests. Thread scheduling policies [20, 21] have been proposed that consider multiple classes of service at the CPU.

Since the requests do not have strict priorities over each other, priority scheduling is not feasible. Periodic requests have to be given priority over others if they are close to missing deadlines. But, if there is sufficient slack time, interactive requests have higher priority such that they can receive lower latencies. Periodic requests are available at the beginning of the round and interactive and aperiodic requests arrive asynchronously at the disk. If periodic requests are given higher priority and served first, aperiodic and interactive requests will experience long response times at the beginning of a round until periodic requests are served. Moreover, it may be possible to better optimize seeks if all the available requests are considered at once.

In our system, it is assumed that the requests are identified by their service type at the scheduler or provided different queues for different request types. The scheduler is designed such that it is independent of the bandwidth allocations. This is done such that the bandwidth allocation parameters or the admission controllers can be changed without modifying the scheduler. To enable this, the aperiodic requests are queued into two separate queues. The first queue hold requests based on the minimum throughput guarantee provided to these requests. The second queue holds any other requests waiting to be served. The scheduler considers the requests from the second queue after periodic requests and interactive requests are served such that these requests

can utilize the unused disk bandwidth.

The scheduler merges the periodic requests and aperiodic requests (from queue 1) into a SCAN order at the beginning of a round. These requests are then grouped into a number of subgroups based on their location on the disk surface. The scheduler serves a subgroup of requests at a time and considers serving interactive requests only at the beginning of a subgroup i.e., the disk SCAN order is not disturbed within a subgroup. If an interactive request is to be served, the scheduler groups this request into the closest subgroup and serves that group next. When there is no subgroup into which the interactive request can be merged, it is served out of SCAN order and the scheduler then moves to serving the next subgroup. An interactive request hence gets a quick response. Interactive requests are served by a first-come first-serve policy to limit the maximum response time of a single request. Since an interactive request (if waiting for service) is served every subgroup, the response time for an interactive request $\leq (n + 2)^*$ (time for serving a subgroup), where n is the number of interactive requests waiting in front of an arriving request. Hence, the response times for interactive requests are determined by the burstiness of the interactive requests and the size of the subgroup. The size of the subgroup can be decreased if tighter latency guarantees are required.

If aperiodic requests are considered in a round only after serving periodic requests, aperiodic requests will experience long response times [13]. If aperiodic requests are considered only at the beginning of the round, it is likely that requests arriving after the beginning of the round will suffer long response times. To avoid this problem, aperiodic requests (of queue 1) are considered and merged into one of the subgroups on their arrival. Admission controllers are designed such that the total time for serving the requests seen by the scheduler in the periodic, interactive and the first aperiodic queue takes less time than a round (based on the analysis presented below). Requests in the second aperiodic queue are served when no periodic and interactive requests are waiting for service. The requests in this queue use the unutilized bandwidth of aperiodic and interactive requests. In our simulation experiments, we considered a round of 0.5 seconds. We considered eight subgroups in each round i.e., a subgroup of requests takes about 62.5 milliseconds.

3.2 Data layout

Data layout plays a significant role on the performance of disk access. It has been suggested by many researchers [7, 8, 9] that video data should be striped [22] across the disks for load balancing and to improve throughput available for a single data stream. 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. 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. Data allocation is rounded up to the next full block so as not to waste space. 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. Some of these issues have been addressed in [23, 11]. The discussion of CTL, CDL is analogous to segment-based or page-based virtual memory discussion.

3.3 Analysis of data retrieval times

In this section, we analyze the data retrieval times to ensure that the total time to serve requests within a round do not exceed the length of the round for providing deterministic guarantees. If m requests retrieve n blocks in a round, in the BCTL scheme, a maximum of m disk accesses are made and in the CDL scheme, n disk accesses are made (assuming that data is distributed across disks on a block basis). 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 seek time and the latency time of a disk access.

It can be shown that worst-case seek time is obtained when the requests are equally spaced on the disk surface. This is based on the fact that the seektime function is a concave function.

In the worst-case scenario, the seek time cost per k requests is bounded by k * seek time cost of (T/k) tracks, where T is the maximum number of tracks on the disk. In the scheduler presented in this paper, the periodic and aperiodic requests are served in the SCAN order. If the total number of these requests is k, then the total seek time for these requests is $\leq k * seek - time(T/k)$. The interactive requests may disturb the SCAN order and hence we use a worst-case estimate for these requests. If there are i such requests, then the total seek time can be bounded by $k * seek - time(T/k) + 2 * i * full - seek - time$. The second term is obtained, in the worst case scenario where all the interactive requests are at one end of the disk surface and all the other requests are at the other end of the surface.

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 $m * rot.\ latency$. This is based on the assumption that the blocks retrieved by a stream in a round are contiguously located on the disk. Based on the above analysis of costs, we can bound the total cost of retrieving a number of blocks in either data layout. If this total cost is below a round at each disk, then data retrieval can be guaranteed.

This analysis also shows that even though the cost of an individual request may depend on the relative location of the blocks on the disk, the overall seek time for a round can be bounded. Hence, when the round is sufficiently large to allow the service of many requests, we can use the number of requests (and their sizes) to estimate the service time of that round accurately (even though individual request service times may be random). This analysis is used by the admission controllers for periodic requests and for the minimum throughput guarantees for aperiodic requests. It is to be noted that overestimates used in admission control do not leave the disk idle as aperiodic requests will soak up any unutilized bandwidth. We will show later that the periodic requests can utilize most of their allocated bandwidth, but the full-seek overestimate for interactive requests limits the disk utilization of interactive requests.

4 Performance Evaluation

4.1 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 [24]. The disk drive characteristics are shown in Table 1. Each disk in the system maintains a load

Table 1. Disk characteristics.

Parameter	Value
Zero Seek time	0.60 ms
Avg. Seek time	8.0 ms
Max. Seek time	17.0 ms
Min. Transfer rate	11.5 MB/s
Max. Transfer rate	17.5 MB/s
Ave. latency	4.17 ms
Spindle speed	7200 RPM
Num. cylinders	3711

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. 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.

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. Four different video streams are considered in our study as explained below.

4.1.1 Traces

MPEG traces from University of Wuerzburg [25] 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

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

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 (about 27 minutes in duration). 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 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.

In simulations, it is assumed that the first block of each movie is stored on a random disk. Data is striped across the eight disks in a round-robin order based on either CDL or BCTL data layout policy. Interactive requests and aperiodic requests are modeled by poisson arrival. Periodic requests are based on real traces of VBR movies. Interactive requests always ask for 64KB of data and aperiodic requests are uniformly distributed over (4kB, 128KB). The burstiness of interactive requests is controlled at each disk by a leaky bucket controller that allowed a maximum of 12 interactive requests per second.

4.2 Results

First, we will show the results of serving VBR streams alone in the system. Then, we will present results of the integrated service.

4.2.1 VBR streams

Fig. 2 shows the impact of block size and data layout strategy on the stream throughput of the four different data streams and the mixed workload. Similar performance trends were observed across individual streams and mixed workloads. It is observed that peak rate based allocation leads to significantly less throughput than the proposed approach. The proposed approach achieves 130% -195% more stream throughput than the peak rate allocation. This improvement is primarily achieved by exploiting the statistical multiplexing of different streams. When requests arrive at the same time, flexible starting times (through latency targets) allow the peaks in demand to be spread over time to improve the throughput. We consider the mixed workload for further experiments.

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 (due to smaller seek and rotational latency costs). The stream throughput for CDL improves significantly for all the data streams as the block size is increased from 32 KB to 256 KB. 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.

For the block sizes considered, the BCTL data layout performs better than the CDL data layout. 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. At smaller block sizes, the difference in performance between BCTL and CDL is larger and as the block size is increased, the difference decreases. In Lambs and Term workloads, at a block size of 256KB, data retrieval requires more than a single block in a 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.

Fig. 3. shows the disk utilization by the video streams as a function of time. The figure shows

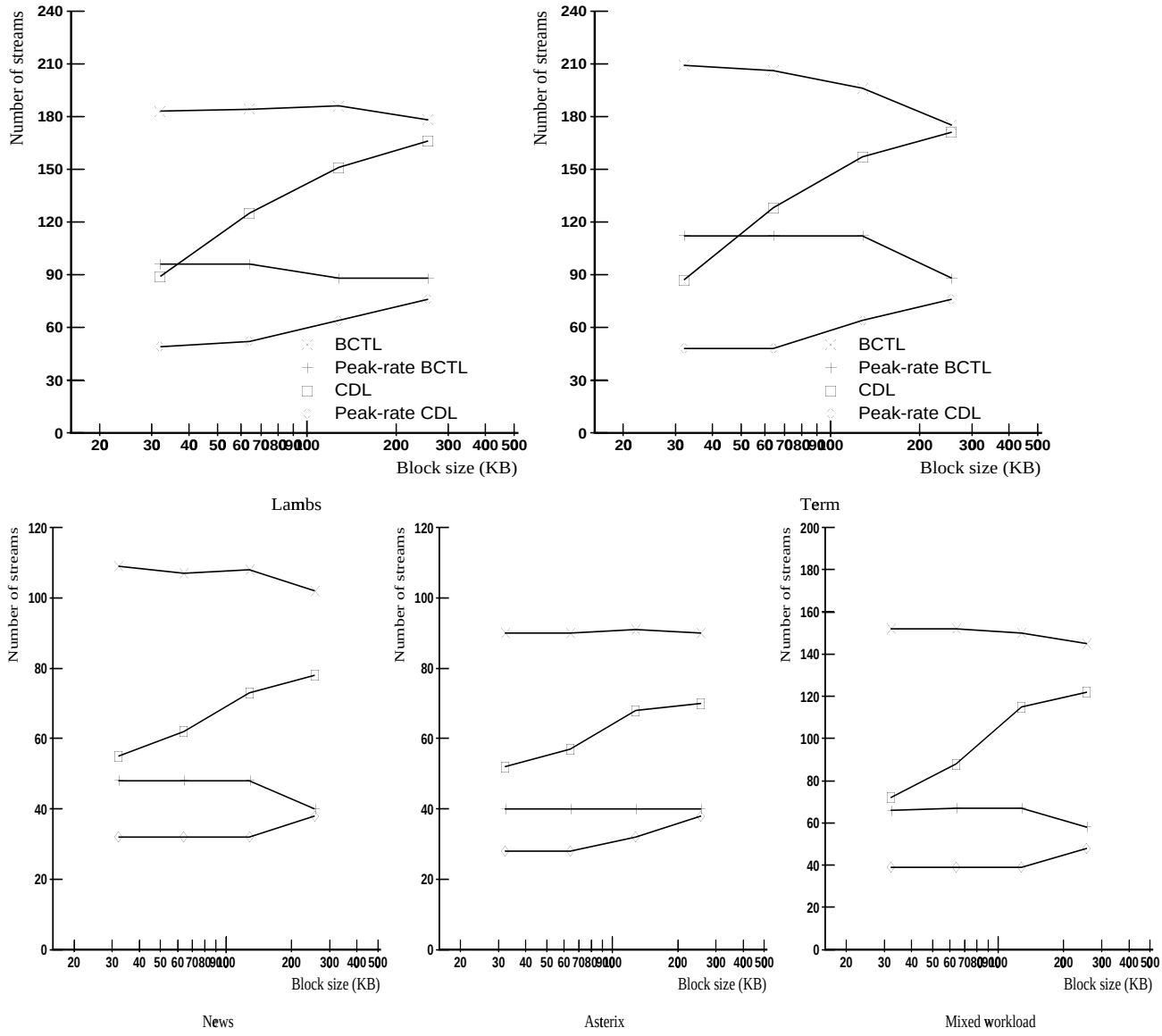

Fig. 2. Impact of data layout and block size on VBR streams.

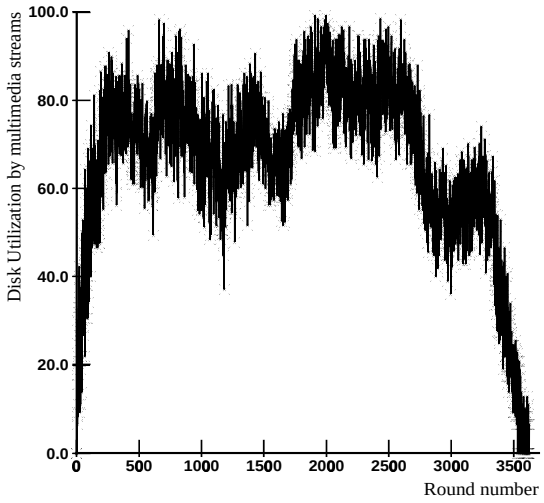

Fig. 3. Disk utilization by VBR streams.

the load at one of the eight disks in the system with a mixed workload. Even though the average utilization is 66%, the disk is nearly 100% busy for several seconds 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. 4. 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 that 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. Stream throughput can also be improved by relaxing the latency targets.

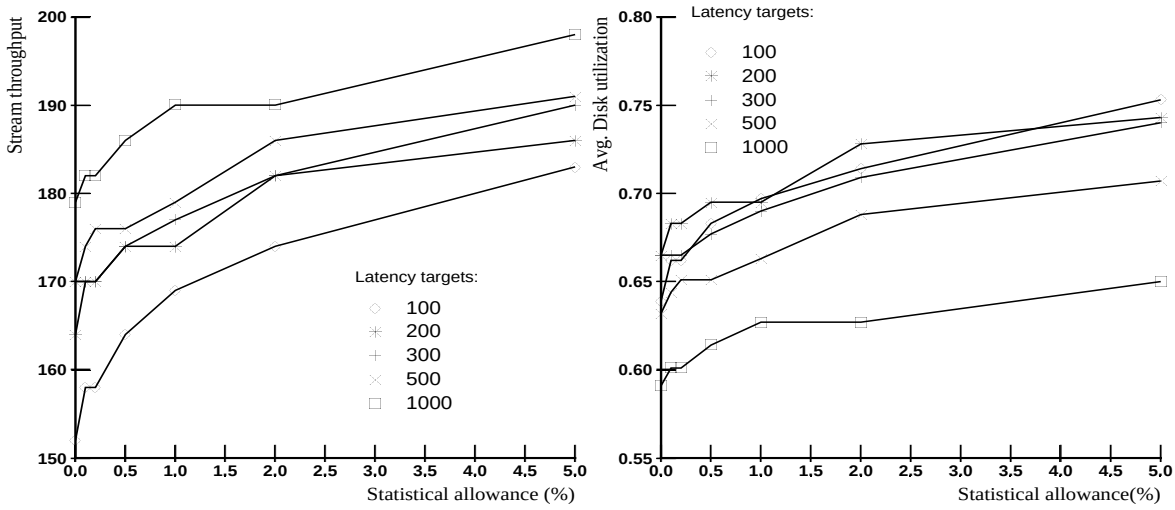

Fig. 4. Effect of statistical allowances.

Fig. 4. 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. 4. 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. Since we are considering admission of requests only at time 0, the larger latency targets increase the time window over which utilizations are being computed and as a result the average disk utilizations

decrease. If we continue admitting new requests (at times other than 0) as earlier requests leave the system, the disk utilizations will continue improving with increased latency targets.

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 suggests that storing the data during the peak demands of popular movies in memory (not the entire movie) can lead to considerable improvement in performance.

4.2.2 Integrated service

Fig. 5. shows the average disk utilization when periodic requests are allocated 50% bandwidth, aperiodic and interactive requests are each allocated 25% bandwidth. The number of periodic requests streams was maintained at the maximum that the system can support. Aperiodic request rate is varied while maintaining the interactive request rate at 50 requests/sec (request rates are measured over the entire system of 8 disks). It is observed that the average utilization of the periodic streams stays below 50%. Because of variations in demand over time, more periodic streams could not be admitted. The utilizations of periodic and interactive requests are unaffected by the aperiodic request rate. It is also observed that as we increase the aperiodic request rate, aperiodic requests take up more and more bandwidth and eventually utilize more than the allocated 25% bandwidth. When periodic and interactive requests don't make use of the allocated bandwidth, aperiodic requests make use of any available bandwidth (25% is the minimum available) and hence can achieve more than the allocated 25% utilization of the disk. This shows that disks are not left idle when aperiodic requests are waiting to be served.

Fig. 6. shows the average and maximum response times of aperiodic and interactive requests as a function of the aperiodic request rate. The number of streams is kept at the maximum allowed by the system and the interactive arrival rate is kept at 50 requests/sec. Interactive response times are not considerably affected by the aperiodic arrival rate and the maximum interactive response time stays relatively independent of the aperiodic arrival rate. It is also observed that the interactive requests achieve considerably better response times than aperiodic requests (260ms maximum interactive response time compared to 1600ms for aperiodic requests both at 50 reqs/sec). Both average and maximum response times are better for interactive requests than for aperiodic

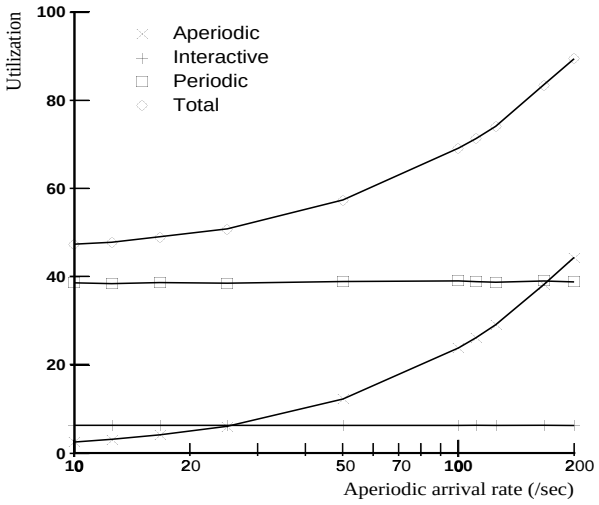

Fig. 5. Average disk utilizations across request categories.

requests even at lower aperiodic arrival rates. We observed that the maximum interactive response times are only dependent on the burstiness of arrival of interactive requests and the bandwidth allocated to them. Zero percentage of periodic requests missed deadlines as aperiodic request rate is varied.

Fig. 7. shows the response times of aperiodic requests and interactive requests (both at 25 requests/sec) as the number of requested streams in the system is varied from 5 to 100. With the considered allocation of bandwidths, the system could support a maximum of 33 streams. Hence, even when more number of streams are requested, the system admits only 33 streams. This shows that the periodic request rate is contained to allow aperiodic requests and interactive requests to achieve their performance goals. We observe that the maximum response times of interactive requests are not considerably impacted by the number of requested streams in the system.

Fig. 8. shows the performance of interactive requests when interactive arrival rate is varied. It is observed that the response times of interactive requests stay bounded at the scheduler irrespective of the arrival rate. Beyond the supported rate, the interactive requests are forced to wait longer in the leaky-bucket controller. Hence, beyond the supported rate, the total delay experienced by an interactive request can exceed the desired latency bound. If higher interactive request rate needs

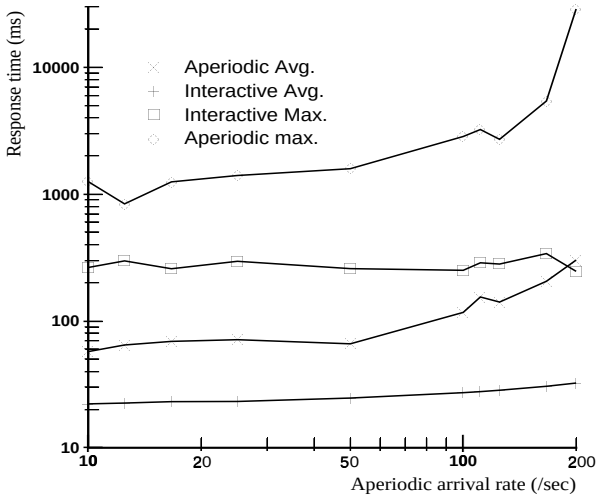

Fig. 6. Impact of aperiodic arrival rate on response times.

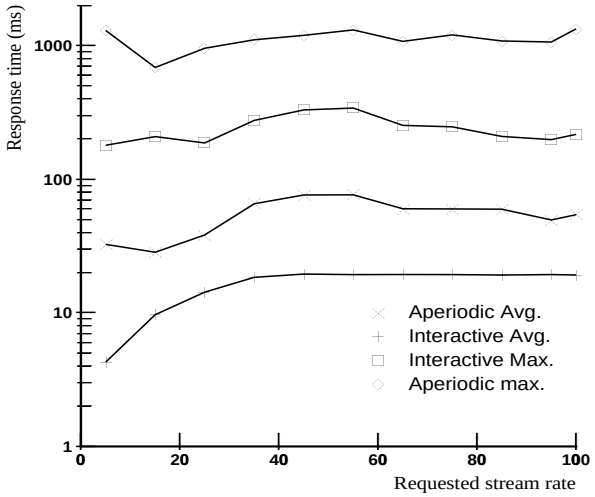

Fig. 7. Impact of requested stream rate on response times.

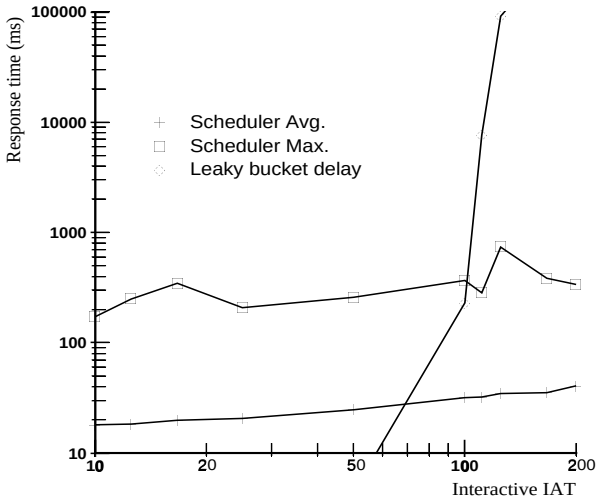

Fig. 8. Impact of interactive arrival rate on response times.

to be supported, these requests have to be allocated higher bandwidth so as to limit the delays at the leaky-bucket controller. We observed that the maximum number of streams supported by the system stayed constant (at 33) irrespective of the arrival rate of the interactive requests and zero percentage of these requests missed deadlines as interactive request rate is varied.

5 Conclusions and Future work

In this paper, we addressed the problem of providing different performance guarantees in a disk system. The proposed approach uses admission controllers and an appropriate scheduler to achieve the desired performance goals. We showed that through proper bandwidth allocation and scheduling, it is possible to design a system such that one type of requests do not impact the performance of another type of requests. We proposed a scheduling policy that allows seek optimization while achieving the performance goals.

We proposed a method for providing deterministic guarantees for VBR streams. We showed that

the proposed approach provides 130%-195% more throughput than peak-rate allocation. We also evaluated the impact of data layout on the performance. Larger block sizes improved performance significantly with the CDL data layout while larger blocks reduced performance slightly with the BCTL data layout. We showed that startup latency is an effective tradeoff parameter for improving stream throughput. For the workloads considered, dropping 1% of the blocks did not provide significant improvement in stream throughput.

In the work presented here, we used static bandwidth allocations to achieve performance goals. We are currently investigating issues in dynamic allocation and adaptive performance guarantees. We are also studying ways of describing an application load on the disks more concisely than a load trace. A WindowsNT based system employing these ideas is being implemented.

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, Kluwer-Publishers*, 1996.

[4] T. Niranjan, T. Chiueh, and G. A. Schloss. Implementation and evaluation of a multimedia file system. *Proc. of IEEE Conf. on Multimedia Computing and Systems*, pages 269–276, June 1996.

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

[6] A. Molano, K. Juvva, and R. Rajkumar. Guaranteeing timing constraints for disk accesses in rt-mach. *Proc. of Real time systems Symposium*, Dec. 1997.

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

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

[9] 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.

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

[11] 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.

[12] D. Jadav and A. Choudhary. Designing and implementing high-performance media-on-demand servers. *IEEE Trans. on Parallel and Distributed Tech.*, pages 29–39, Summer 1995.

[13] A. L. N. Reddy and J. Wyllie. I/O issues in a multimedia system. *IEEE Computer*, Mar. 1994.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] A. L. Narasimha Reddy and R. Wijayaratne. On providing deterministic guarantees for vbr streams. *Tech. rep. TAMU-ECE-9701, Texas A&M University*, Apr. 1997.

[19] R. Golding, E. Shriver, T. Sullivan, and J. Wilkes. Attribute-managed storage. *Proc. of Workshop on modeling and specification of I/O*, Oct. 1995.

[20] P. Goyal, X. Guo, and H. Vin. A hierarchical CPU scheduler for multimedia operating systems. *Proc. of Symp. on Operating System Design and Implementation*, 1996.

[21] C. Waldspurger and W. Weihl. Lottery Scheduling: Flexible proportional-share resource management. *Proc. of Symp. on Operating System Design and Implementation*, Nov. 1994.

[22] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Conference*, June 1988.

[23] 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.

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

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