

Providing QOS guarantees for disk I/O *

Ravi Wijayarathne¹
A. L. Narasimha Reddy

¹Dept. 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 performance guarantees 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 performance 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 in the context of the proposed approach.

Keywords: Disk Scheduling, VBR streams, QOS, multiple QOS goals, seek optimization.

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

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 each interval.

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 [1, 2, 3, 4, 5, 6]. 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. An evaluation of tradeoffs in a media-on-demand server can be found in [7]. 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.

This paper addresses the problem of providing different levels of service for different classes of requests in a single system. This paper makes the following two significant contributions: (1) an integrated scheme is presented for providing different levels of performance guarantees to different classes of requests. (2) a method is presented for providing deterministic guarantees for VBR streams that exploits statistical multiplexing of resources. The paper also presents an evaluation of tradeoffs in providing deterministic and statistical 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 some of the other related 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 Performance Guarantees

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. Each category of requests employs an

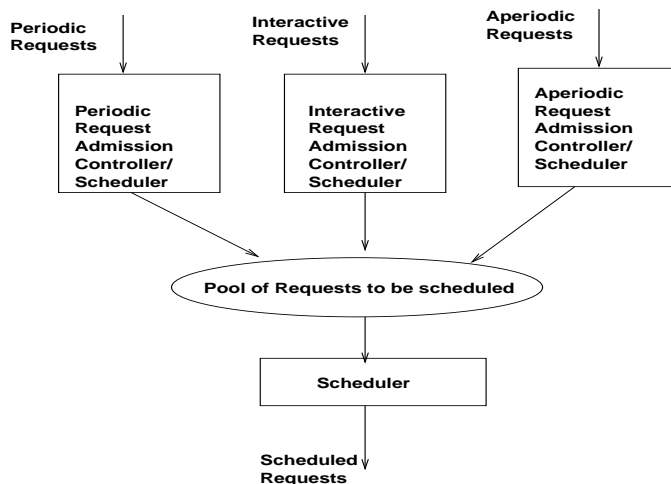


Figure 1: Supporting multiple QOS levels.

admission controller to limit the disk utilization of these requests to their allocated level. 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. Similar approaches have been independently proposed recently in [8, 9]. Both these schemes employ a two-level scheduling approach as proposed here. The work in [8] shares many of the motivations of our work. The scheduler in [9] doesn't support quick response to interactive requests. Our work here also proposes a scheme for allowing statistical multiplexing of VBR streams while providing deterministic guarantees for them.

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.

The admission controller for each class of requests controls the number of requests entering the pool of requests and also the order in which the requests enter the pool. These controllers besides enforcing the bandwidth allocations, control the policy for scheduling the requests in that class of service. This can be generalized to a larger number of request classes, each with its own admission controller/scheduler. The disk level scheduler schedules requests from the request pool to meet the performance criteria of individual requests.

In our system, it is assumed that the requests are identified by their service type at the scheduler. 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.

We first describe the overall functioning of the disk scheduler and then describe how admission control is implemented for each class of requests.

2.1 Scheduling for multiple QOS levels

Since the different classes of 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 can 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.

The disk scheduler uses a round based scheme for scheduling the requests from the candidate pool. Each admission controller schedules the requests in its class and releases them as candidate requests at the beginning of the round. Each admission controller ensures that its class doesn't take any more time than allocated in a round. The disk scheduler combines the requests and serves them together to meet performance goals of individual requests. If all the requests arrive at the beginning of a round, the disk scheduler will not have to worry about deadlines since the admission controllers enforce the time constraints. However, aperiodic and interactive requests arrive asynchronously. To schedule these requests as they arrive (without waiting for the beginning of next round), the disk scheduler uses the notion of a subperiod. The disk scheduler considers the available slack time of periodic requests and adjusts the schedule to incorporate any arriving interactive and aperiodic

requests each subperiod.

The aperiodic requests are queued into two separate queues. The first queue holds requests based on the minimum throughput guarantee provided to these requests. Scheduling these requests will not violate any time constraints since these requests are within the allocated bandwidth. 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. Later arriving aperiodic requests (of queue 1) are, if possible, merged into remaining subgroups. The scheduler considers serving interactive requests only at the beginning of a subgroup i.e., the disk SCAN order is not disturbed within a subgroup. To provide quick response times for interactive requests, the SCAN order may be disturbed at the end of subgroups. When possible, the scheduler groups a waiting interactive request into the closest subgroup and serves that group next to minimize the seek overhead in serving these requests. Interactive requests are queued on a first-come first-serve basis to limit the maximum response time of a single request. If sufficient slack time exists, waiting interactive requests are first served before moving to the next subgroup of requests. The response times for interactive requests are hence determined by the burstiness of the interactive requests and the size of the subperiod. The size of the subperiod can be decreased if tighter latency guarantees are required. Arranging requests into subgroups also allows the scheduler to communicate to the device driver in an efficient manner while ensuring that the requests are not reordered by the scheduling algorithm within the disk drive. A more formal description of the scheduler is given in Fig. 2.

Claim: If each group abides by the bandwidth allocation, i.e., service time for group $i \leq T_i$, then the combined schedule for all the requests $\leq \sum_{i=1}^n T_i$, where T_i is the time allocation for group i within a round.

Proof: We only need to consider seek times since other components of the service time don't change due to merging of requests. (Actually, the rotational latencies could change, but since we are using worst-case estimates, they don't impact the estimates). Without loss of generality, consider two groups of requests. Group 1 has requests a, b and Group 2 has requests c, d . There are two possible cases: the groups overlap on the disk surface or not.

Case 1: Overlap The requests are as shown in Fig. 3 on the disk surface. The two groups

```

While(true)
{
    Combine periodic and aperiodic1 requests into SCAN order;
    Break the requests into subgroups;
    slack_time = round_time - service estimate for above requests;
    while(not end of round)
    {
        pick first interactive request if any;
        Combine with one of the remaining subgroups?
        If (no)
        {
            while (estimated service time < slack_time)
                service waiting interactive requests;
            Serve the closest subgroup;
        }
        else serve the merged subgroup;
        Adjust slack_time;
        while (slack_time > 0)
        {
            Combine aperiodic1 requests into existing subgroups;
            Adjust slack_time;
        }
        if (all periodic requests served)
        {
            Continue serving interactive & aperiodic requests until end of round;
        }
    }
}

```

Figure 2: Semi-formal description of the scheduler.

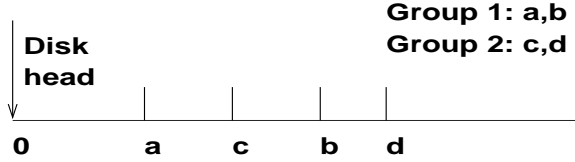


Figure 3: Overlapping Request Groups

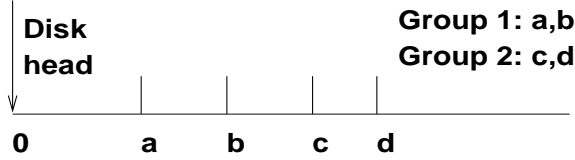


Figure 4: Nonoverlapping Request Groups

calculate the seek times as $S_1 = s_{0a} + s_{ab}$ and $S_2 = s_{0c} + s_{cd}$. When merged, the seek times are $S_{merged} = s_{0a} + s_{ac} + s_{cb} + s_{bd}$. We need to show that $S_1 + S_2 \geq S_{merged}$ or $s_{0a} + s_{ab} + s_{0c} + s_{cd} \geq s_{0a} + s_{ac} + s_{cb} + s_{bd}$. Since $s_{0c} \geq s_{ac}$, $s_{ab} \geq s_{ac}$ and $s_{cd} \geq s_{bd}$, the above is true.

Case 2: No Overlap The requests are as shown in Fig. 4 on the disk surface. The two groups calculate the seek times as $S_1 = s_{0a} + s_{ab}$ and $S_2 = s_{0c} + s_{cd}$. When merged, the seek times are $S_{merged} = s_{0a} + s_{ab} + s_{bc} + s_{cd}$. We need to show that $S_1 + S_2 \geq S_{merged}$ or $s_{0a} + s_{ab} + s_{0c} + s_{cd} \geq s_{0a} + s_{ab} + s_{bc} + s_{cd}$, which is clearly true since $s_{0c} \geq s_{bc}$.

If all the requests arrived at the beginning of the round, the above claim is sufficient to prove that guarantees will be met if the individual groups observe the bandwidth allocations. Interactive requests disturb the SCAN schedule and hence are assumed to require worst-case seek time such that servicing these requests won't violate the bandwidth allocations of other requests. However, aperiodic and interactive requests arrive asynchronously. Hence, the slack times need to be considered so as to not violate the deterministic guarantees for VBR streams while scheduling these late arriving requests.

2.2 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. 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 streams 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 ([1]). 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*. Compared to the size of the movie file (about 1 GB for 90 minutes of MPEG-1 quality), the size of the demand trace file is not very significant.

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 proposed approach allows the load across the disks to be "smoothed" across the different streams being served by the system. Individual stream smoothing is considered in a number of studies, for example in [10], to reduce the variations of demand of a single stream. These techniques typically prefetch blocks ahead of time to optimize desired characteristics (reduce peak rate, reduce demand variations, etc.) of an individual stream. It is possible to apply individual stream smoothing techniques in addition to the proposed technique of smoothing demands over different streams. A recent related study [11] showed that individual stream smoothing didn't offer significant additional benefit when applied along with the proposed approach.

2.3 Latency and bandwidth guarantees

Latency guarantees are provided by the disk scheduler as explained earlier. 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 within a second 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. Interactive requests are scheduled in a FIFO order to limit the queuing times of individual requests. We will use maximum response time as a performance measure for these requests. Sophisticated admission controllers (that take request burstiness into account) [12] can be employed if it is necessary to limit the waiting times of interactive requests at the admission controllers.

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. Bandwidth guarantees are provided to aperiodic requests by ensuring that certain minimum number of requests are scheduled every round.

3 Other issues

3.1 Stream scheduling

Stream scheduling deals with the issue of scheduling an arriving VBR stream. 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, $s, s + p, s + 2p, s + 3p \dots$ are considered until the stream can be scheduled. Since p is prime, all the rounds within the latency target window will be considered. All the policies except the minimal load policy, in the worst case, require $O(Ld)$ time, where L is the latency target and d is the length of the demand trace. The minimal load policy, in the worst case, requires $O(Ld + LN)$ time, where the additional $O(LN)$ time is needed for choosing the slot that minimizes the maximal load on the N disks.

Latency target impacts stream throughput in two ways. A larger target allows us to search more slots to find a suitable starting point for 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 [13]: 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 [14].

3.2 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 [15] across the disks for load balancing and to improve throughput available for a single data stream [16]. Data for a VBR stream can be stored in (i) Constant Data Length (CDL) units (ii) Constant Time Length (CTL) units [17, 18]. 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. For example, if a stream requires 2, and 3 blocks in two consecutive rounds, CTL layout will have these blocks on two disks, say A and B, A in round 1 and B in round 2. CDL layout will have data for round 1 on disks A and B and data for round 2 on disks C, D and A if there are 4 disks (A, B, C and D) in the system. However, in both the data layouts, data required by the application in a round is retrieved at once at the beginning of the previous round. 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. 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. Different data layouts are considered in this paper to show that the proposed mechanisms can function well in either data layout.

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

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

[19]. 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. Data block size on the disk is varied from 32KB to 256KB. Data is striped across the eight disks in a round-robin order based on either CDL or BCTL data layout policy. In simulations, it is assumed that the first block of each movie stream is stored on a random disk.

Interactive requests and aperiodic requests are modeled by Poisson arrival. Periodic requests are based on real traces of VBR movies. Periodic request load is varied by requesting more streams to be scheduled. 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.

The admission controller for periodic streams employed the strategy discussed in section 2.2. CDL and BCTL data layout strategies are considered. 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.

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

The simulations are carried out in two different phases. In the first phase, we only considered the VBR streams to study the effectiveness of the proposed scheduler for VBR streams. In the second phase, we considered integrated service of three different types of requests.

4.1.1 Traces

MPEG traces from University of Wuerzburg [20] 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 (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.

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.

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

VBR admission control policy: Fig. 5 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 peak-rate based scheme utilizes peak demand of the stream over a round for determining admissibility of that stream. 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. The proposed approach improves the stream throughput compared to a peak-rate based scheme in both the data layouts.

Fig. 6 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. 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 other 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 multiple types of requests. This result shows the need for bandwidth allocation among different classes of requests.

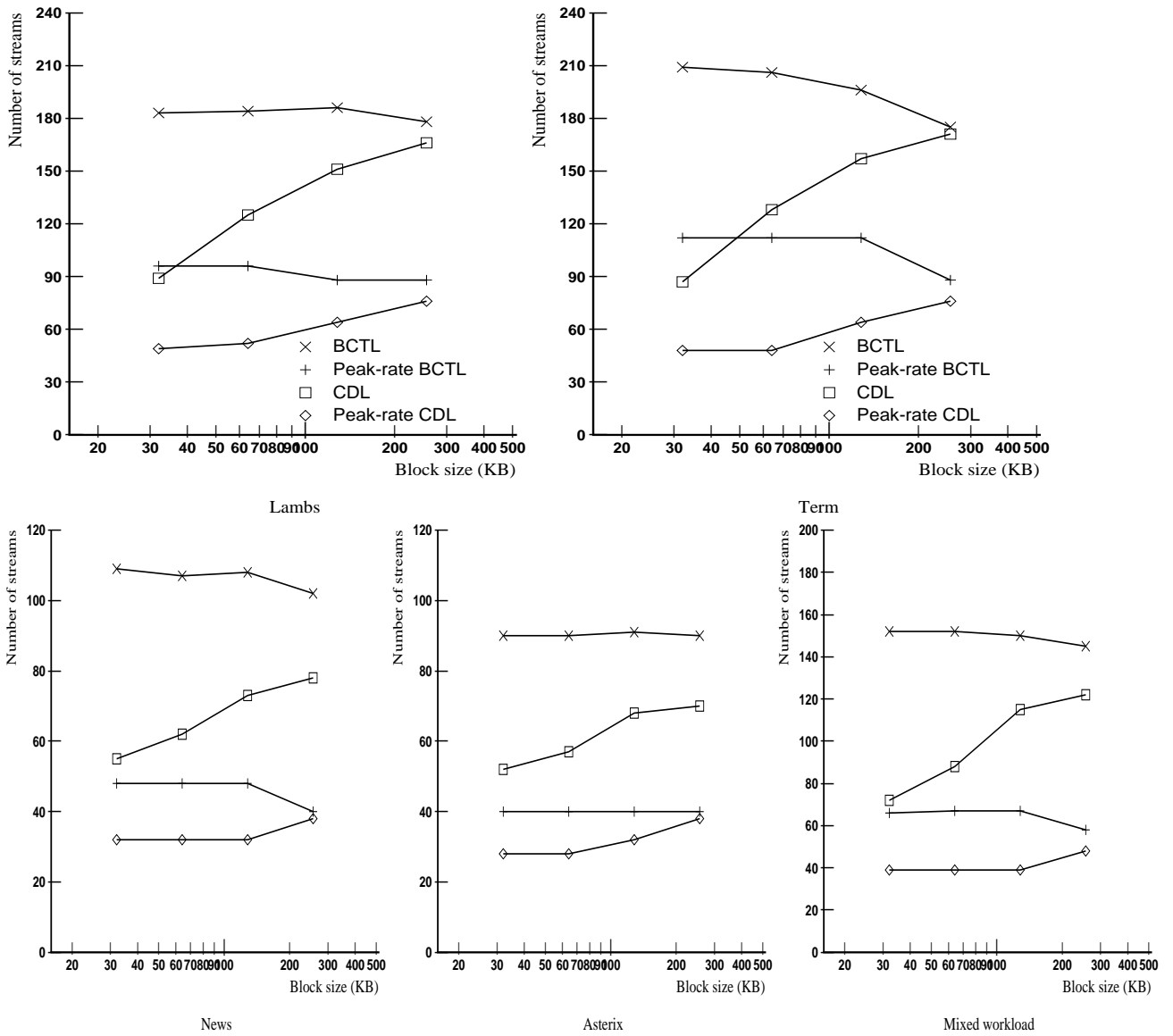


Figure 5: Impact of data layout and block size on VBR streams.

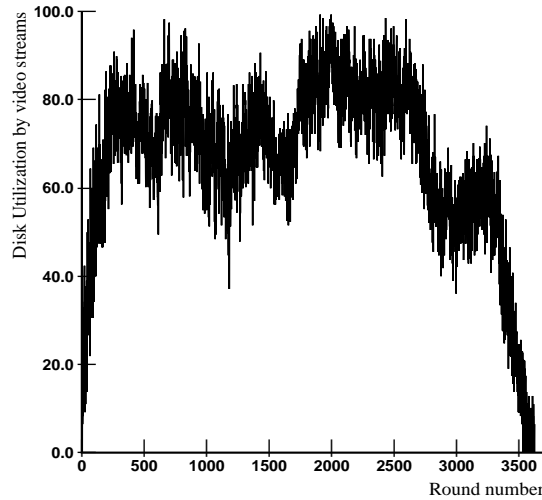


Figure 6: Disk utilization by VBR streams.

Stream Scheduling: Fig. 7 shows the impact of stream scheduling policies on the stream throughput at various block sizes. The results in Fig. 7 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. 8 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. More extensive results on stream scheduling can be found in [21].

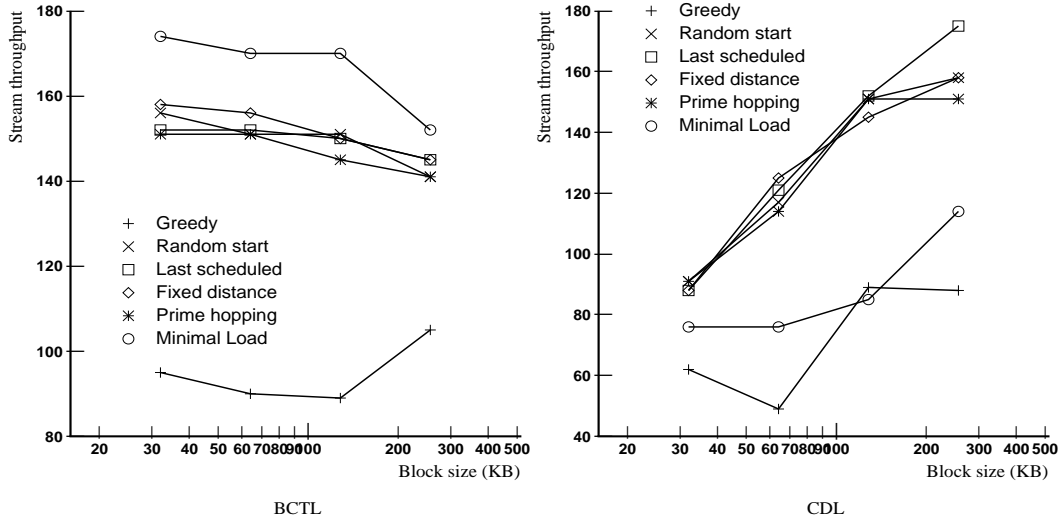


Figure 7: Impact of stream scheduling.

Statistical guarantees: Fig. 9 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. In our scheme, the admission controller decides which blocks of a stream get dropped i.e., the blocks to be dropped are determined at the time of admission. Otherwise, it would be difficult to provide stream isolation i.e., a stream requesting statistical guarantees can force another stream requesting deterministic service to drop blocks at the time of retrieval. 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. 9 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

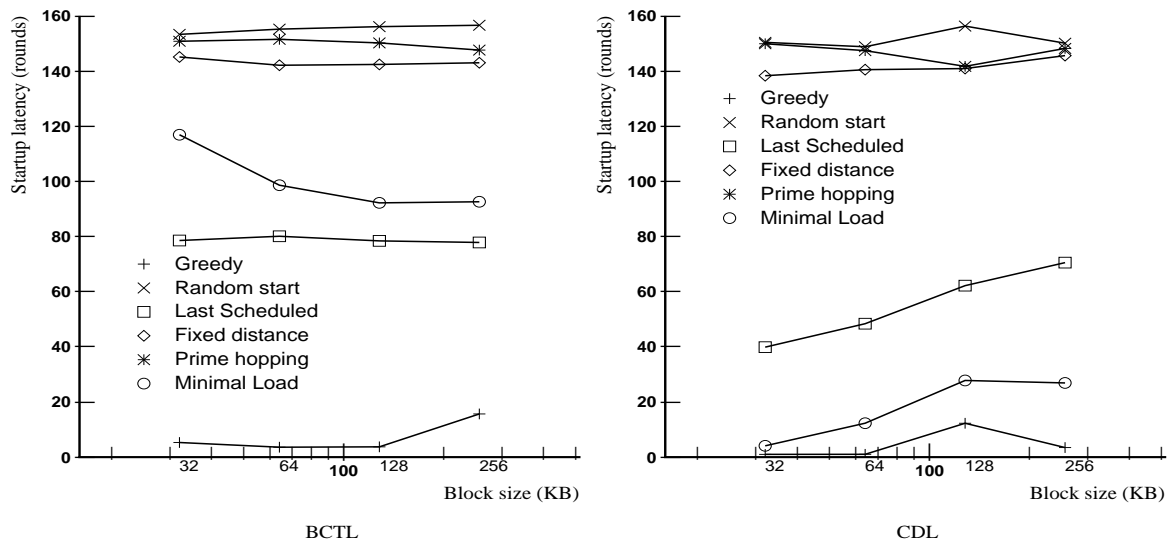


Figure 8: Average startup latency.

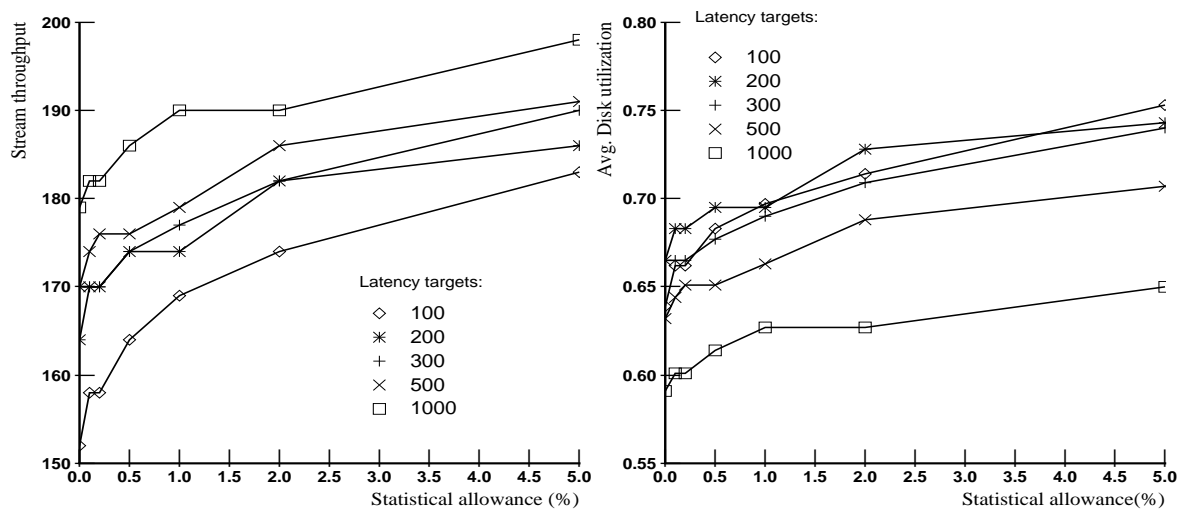


Figure 9: Effect of statistical allowances.

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. 9 also shows the impact on the average disk utilization as a function of the statistical allowances and latency targets. As higher statistical allowances are made, the disk utilizations are improved as more 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%. The primary reason for this is that the proposed technique achieved significant statistical multiplexing of streams while providing deterministic guarantees.

4.2.2 Integrated service

Fig. 10 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,

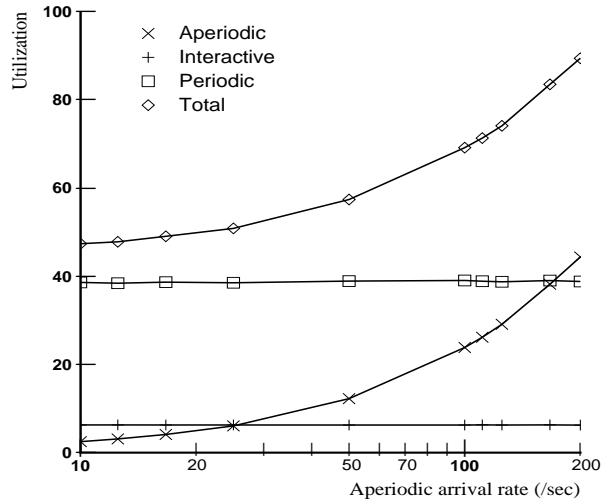


Figure 10: Average disk utilizations across request categories.

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. 11 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 (260 ms maximum interactive response time compared to 1600 ms for aperiodic requests both at 50 reqs/sec). Both average and maximum response times are better for interactive requests than for aperiodic 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. 12 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

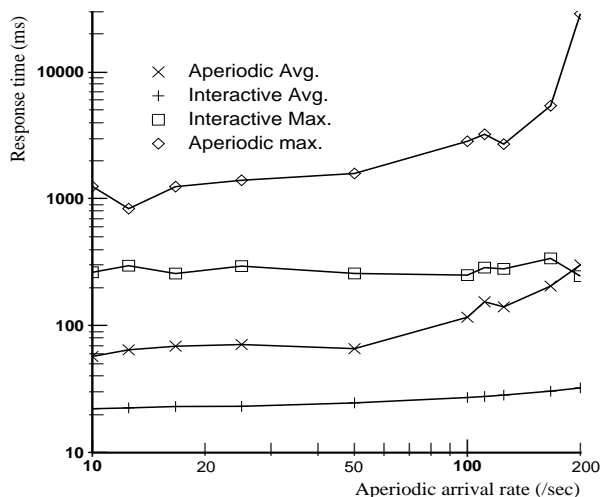


Figure 11: Impact of aperiodic arrival rate on response times.

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. 13 shows a comparison of the proposed scheduling algorithm and a variant of SCAN scheduling algorithm used in most of the current disks [22]. The figure shows the average and maximum response times of interactive requests as the aperiodic request rate is varied. The proposed method achieves better average and maximum response times compared to SCAN. As the aperiodic arrival rate is increased, both the maximum and average response times of interactive requests get impacted with SCAN scheduling policy. The proposed method isolates the request categories and maintains the performance of the interactive requests independent of the aperiodic arrival rate.

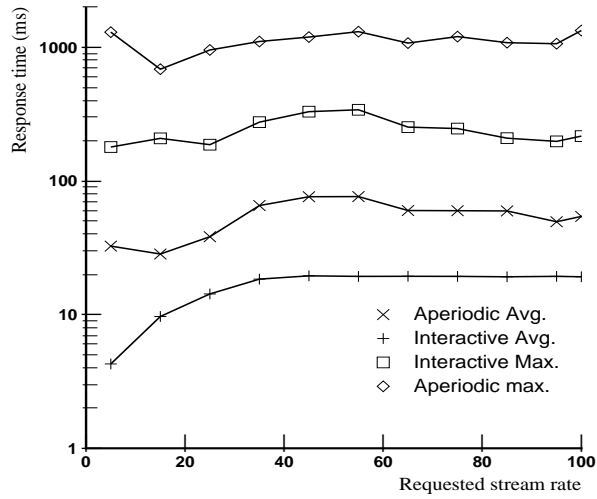


Figure 12: Impact of requested stream rate on response times.

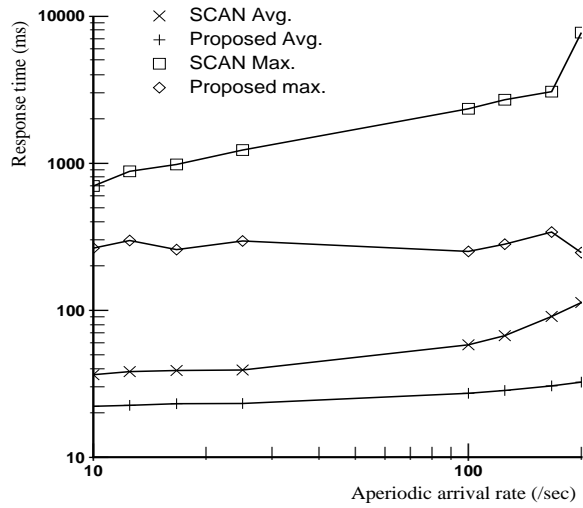


Figure 13: Comparison with SCAN.

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 also proposed a method for providing deterministic guarantees for VBR streams that exploited statistical multiplexing of different 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. We showed that startup latency is an effective tradeoff parameter for improving stream throughput. For the workloads considered, statistical allowances on top of the proposed approach 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.

6 Acknowledgements

Reviewers comments have greatly contributed to the improved presentation of the paper.

References

- [1] A. L. N. Reddy and J. Wyllie. I/O issues in a multimedia system. *IEEE Computer*, Mar. 1994.
- [2] 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.
- [3] D. J. Gemmell and S. Christodoulakis. Principles of delay-sensitive multimedia storage and retrieval. *ACM Trans. on Info. Systems*, pages 51–90, 1992.
- [4] 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.
- [5] A. Molano, K. Juvva, and R. Rajkumar. Guaranteeing timing constraints for disk accesses in rt-mach. *Proc. of Real time systems Symposium*, Dec. 1997.
- [6] 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.
- [7] 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.
- [8] P.J.Shenoy and H. M. Vin. Cello: A disk scheduling framework for next generation operating systems. *Proc. of ACM SIGMETRICS*, June 1998.
- [9] M. M. Budhikot, X. J. Chen, D. Wu, and G. M. Parulkar. Enhancements to 4.4 BSD UNIX for efficient networked multimedia in project MARS. *Proc. of IEEE Multimedia Computing and Systems Conf.*, pages 326–337, June 1998.
- [10] 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.
- [11] A. L. Narasimha Reddy and R. Wijayarathne. Techniques for improving the throughput of VBR streams. *Proc. of ACM/SPIE Conf. on Multimedia Computing and Networking*, Jan. 1999.
- [12] H. Zhang and D. Ferrari. Rate-controlled static-priority queueing. *Proc. IEEE INFOCOM*, Apr. 1993.

- [13] R. Golding, E. Shriver, T. Sullivan, and J. Wilkes. Attribute-managed storage. *Proc. of Workshop on modeling and specification of I/O*, Oct. 1995.
- [14] 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.
- [15] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Conference*, June 1988.
- [16] S. Berson, S. Ghandeharizadeh, and R. Muntz. Staggered striping in multimedia information systems. *Proc. of SIGMOD*, 1994.
- [17] 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.
- [18] 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.
- [19] Seagate Corp. Disk drive product info. <http://www.seagate.com/>, 1997.
- [20] O. Rose. Mpeg trace data sets. <ftp-info3.informatik.uni-wuerzburg.de>, 1995.
- [21] A. L. Narasimha Reddy and R. Wijayarathne. On providing deterministic guarantees for VBR streams. *Tech. rep. TAMU-ECE-9701, Texas A&M University*, Apr. 1997.
- [22] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling algorithms for modern disk drives. *Proc. of ACM SIGMETRICS*, pages 241–251, May 1994.