Server-Based Smoothing of Variable Bit-Rate Streams

Stergios V. Anastasiadis
Department of Computer Science
University of Toronto
stergios@cs.toronto.edu

Kenneth C. Sevcik
Department of Computer Science
University of Toronto
kcs@cs.toronto.edu

Michael Stumm
Dept of Electrical and Computer Engineering
University of Toronto
stumm@eecg.toronto.edu

March 5, 2001

ABSTRACT

We introduce an algorithm that uses buffer space available at the server for smoothing disk transfers of variable bit-rate streams. Previous smoothing techniques prefetched stream data into the client buffer space, instead. However, emergence of personal computing devices with widely different hardware configurations means that we should not always assume abundance of resources at the client side. The new algorithm is shown to have optimal smoothing effect under the specified constraints. We incorporate it into a prototype server, and demonstrate significant increase in the number of streams concurrently supported at different system scales. We also extend our algorithm for striping variable bit-rate streams on heterogeneous disks. High bandwidth utilization is achieved across all the different disks, which leads to server throughput improved by several factors at high loads.

1. INTRODUCTION

Variable bit-rate encoding of video streams can achieve quality equivalent to constant bit-rate encoding while requiring average bit rate that is lower by 40% or more [10, 13]. However, variable bit-rate streams have high variability in their resource requirements which can lead to low utilization of disk and network bandwidth in the common case. This occurs because the aggregate bandwidth requirements of concurrently served streams can be significantly higher at particular time instances than on average, and the admission control process typically bases its decisions on peak aggregate demand when considering new stream requests.

In order to improve resource utilization and the throughput of the system, a number of smoothing techniques have been proposed that can remove peaks in the required transfer bandwidth of individual streams by appropriately prefetching stream data during periods of lower bandwidth demand. To date smoothing schemes always prefetched data into the client buffers. Although such an approach can improve the utilization of both disk and network bandwidth, it is dependent on the amount of buffer space available at the client.

In this paper, our goal is to maximize the average number of users supported concurrently in video server systems, by applying smoothing techniques and combining them appropriately with disk striping and admission control policies. Thus, we introduce a stream smoothing algorithm that prefetches data into server buffers, which has several important advantages:

- ability to provide the benefits of smoothing even to clients with minimal memory resources (such as inexpensive mass-produced specialized devices),
- ability to limit the requirements for disk bandwidth, which is estimated to increase at rates an order of magnitude slower than network link bandwidth [9],
- reduced complexity in admission control processing because separate transfer schedules for each individual client type are not required,
- reduced stream replication, since a single retrieval sequence and striping layout suffices for all clients.

Server-side prefetching addresses disk bandwidth and not network utilization. However, our smoothing scheme accepts as input a specification of the quantity of data that should be sent to the client over time. Thus, its operation can be complemented with network smoothing techniques for cases where clients have sufficient buffer resources.

In order to prevent excessive smoothing from exhausting the available buffer space, we apply a novel scheme where data prefetching is done as long as the proportion of server buffer required by each stream does not exceed the corresponding (decreased) proportion of the required disk bandwidth. Thus, the smoothing process is adjusted automatically, according to the total memory and disk bandwidth available in the server configuration.

Another aspect we study in this paper is smoothing of variable bit-rate streams striped across heterogeneous disks, something that has not been done previously as far as we know. In the past, it was assumed that load-balancing and reliability problems restrict the size of disk arrays across which stream data can be striped efficiently [6, 28]. However, more recently disk striping schemes that are scalable have been introduced [1, 4]. It is important to consider the efficient operation of a server with heterogeneous disks because this allows server installations to be incrementally expanded using the most advanced and cost-efficient storage devices as the system load increases. With the ratio between disk *storage capacity* and disk *bandwidth* increasing by a factor of ten every decade [9], disk accesses are becoming more precious. Therefore, bandwidth is the particular disk resource that our approach strives to use best.

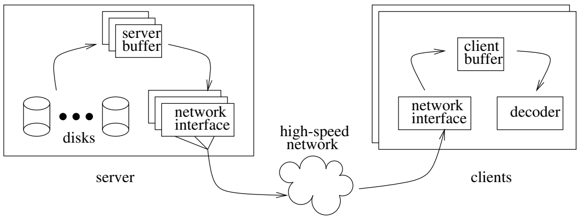

Figure 1: Compressed video streams are stored across multiple disks of the media server. Multiple clients can connect and start playback sessions via a high-speed network.

This smoothing scheme was implemented in a prototype server. Experiments with various MPEG-2 streams demonstrated an increase in the system throughput that can reach 15% with homogeneous disks and can exceed a factor of three with heterogeneous disks in the configuration that we used.

The rest of this paper is structured as follows. In Section 2, we present a high-level description of the system architecture that we choose in our study. In Section 3, we introduce the Server Smoothing algorithm and in Section 4, we describe our experimentation environment, including the stream benchmark used in our experiments. In Section 5, we study the performance of Server Smoothing on homogeneous disks, and in Section 6, we extend the algorithm to apply to heterogeneous disks. In Section 7, we validate our arguments with detailed simulated disk measurements. In Section 8, we summarize previous research, and relate it to our work, and in Section 9 we summarize our conclusions.

2. SYSTEM ARCHITECTURE

2.1 Overview

The system we propose operates according to the server-push model. When a playback session starts, the server periodically sends data to the client until either the end of the stream is reached, or the client explicitly requests suspension of the playback. Data transfers occur in rounds of fixed duration T_{round}. In each round, an appropriate amount of data is retrieved from the disks into a set of server buffers reserved for each active client. Concurrently, data are sent from the server buffers to the client through the network interfaces (Figure 1).

The amount of stream data periodically sent to the client is determined by the decoding frame rate of the stream and the resource management policy of the network. One reasonable policy would send to the client during each round the amount of data that will be needed for the decoding process at the client in the next round; any other policy that does not violate the timing requirements and buffering constraints of the decoding client would be also acceptable.

The streams are compressed according to the MPEG-2 specification, or any other encoding scheme that supports constant quality quantization parameters and variable bit rates. The stream data are stored across multiple disks, as shown in Figure 1. Playback requests arriving from the clients are initially directed to an admission control module, where it is determined whether enough resources exist to activate the requested playback session either immediately or within a limited number of rounds. A schedule database maintains for each stream information on how much data needs to be accessed from each disk in any given round, the amount of server buffer space required, and how much data needs to be transferred to the client. This scheduling information is generated when the media stream is first stored and is used for both admission control and to control data transfers during playback. It is possible that two or more replicas are available for each stream file, with different corresponding schedules.

2.2 Stride-Based Disk Space Allocation

In our experiments, we use a method called *stride-based allocation* for allocating disk space [1]. In stride-based allocation, disk space is allocated in large, fixed-sized chunks called *strides*. The strides are chosen larger than the maximum stream request size per disk during a round. This size is known a priori, since stored streams are accessed sequentially according to a predefined (albeit variable) rate. When a stream is retrieved, only the requested amount of data is fetched to memory during a round, and not the entire stride.

Stride-based allocation eliminates external fragmentation, while internal fragmentation remains negligible because of the large size of the streams relative to strides, and because a stride may contain data of more than one round. Another advantage of this method is that it sets an upper-bound on the estimated disk access overhead during retrieval; since the size of a stream request never exceeds the stride size during a round, at most two partial stride accesses will be required to serve the request of a round on each disk.

2.3 Reservation of Server Resources

A mathematical abstraction of the resource requirements is necessary for scheduling purposes. Initially, we consider a system with D functionally equivalent disks, although a more general case of heterogeneous environments is examined later. In the following sequence definitions, a zero value is assumed outside the specified range.

The stream *Network Sequence*, $\mathbf{S_n}$, of length L_n defines the amount of data, $S_n(i)$, $1 \le i \le L_n$, that the server must send to a particular client during round i of its playback. The *Buffer Sequence*, $\mathbf{S_b}$, of length $L_b = L_n + 1$ defines the server buffer space, $S_b(i)$, $0 \le i \le L_b$, required by the stream data during round i. The *Disk Sequence* $\mathbf{S_d}$ of length $L_d = L_n$ defines the total amount of data, $S_d(i)$, $0 \le i \le L_d - 1$, retrieved from all the disks in round i for the client. We assume that stream data are stored on the disks in logical blocks of fixed size B_l, which is multiple of the physical sector size B_p of the disk. Both the disk transfer requests and the memory buffer reservations are specified in multiples of the block size B_l. The *Disk Striping Sequence* $\mathbf{S_m}$ of length L_d determines the amount of data $S_m(i,k)$, $0 \le i \le L_d - 1$, that are retrieved from disk k, $0 \le k \le D-1$, in round i. It can be easily derived from the corresponding disk sequence $\mathbf{S_d}$ according to the striping method used.

We assume that each disk has edge to edge seek time $T_{fullSeek}$, single-track seek time $T_{trackSeek}$, average rotational latency T_{avgRot}, and minimum internal transfer rate R_{disk}. The stride-based disk space allocation policy enforces an upper bound of at most two disk arm movements per disk for each client per round. The total seek distance can also be limited using a CSCAN disk scheduling policy. Let M_i be the number of active streams during round i of the system operation, and l_j the round of system operation that the playback of stream j, $1 \leq j \leq M_i$, started. Then, the total access time on disk k in round i of the system operation will have an upper-bound of:

$$T_{disk}(i,k) = 2T_{fullSeek} + 2M_i \cdot (T_{trackSeek} + T_{avgRot}) + \sum_{j=1}^{M_i} S_m^j(i - l_j, k)/R_{disk}$$

where $\mathbf{S_m^j}$ is the disk striping sequence of client j. $T_{fullSeek}$ is counted twice due to the disk arm movement from the CSCAN policy, while the factor two in the second term is due to the stride-based allocation scheme we use. The first term should be accounted for only once in the disk time reservation structure of each disk, but each client j incurs an extra access time of

$$T_{disk}^j(i,k) = 2 \cdot (T_{trackSeek} + T_{avgRot}) + S_m^j(i - l_j, k)/R_{disk}$$

on disk k during round i, when $S_m^j(i - l_j, k) > 0$, and zero otherwise. Reservations of network bandwidth and buffer space are more straightforward, and based on the network and buffer sequence of each accepted playback request, respectively.

2.4 Variable-Grain Striping

In *Variable-Grain Striping*, the data retrieved during a round for a client are always accessed from a single disk, and the disks are used in round-robin fashion in successive rounds. The disk striping sequence determines the particular single disk accessed and the exact amount of data retrieved during each round. Comparison with alternative striping techniques has shown significant performance benefit for Variable-Grain Striping [1], and this is the method that we use in the present study.

3. SERVER-BASED SMOOTHING

3.1 Outline

Previous studies have pointed out the potentially low disk utilization (and system throughput) achieved when retrieving variable bit-rate streams, and the need for appropriately prefetching data into the server buffers [15, 22]. A similar utilization problem was also studied in the context of network links carrying variable bit-rate streams, where it was proposed to smooth bit-rate peaks along a stream by prefetching data into client buffers [7, 25]. It was shown that such an approach can improve bandwidth utilization in both disks and network links, but is dependent on the memory configuration of individual clients.

Here, we consider smoothing out disk bandwidth peaks by prefetching stream data into server buffers. One crucial issue with disk prefetching is how to maintain an appropriate balance between disk bandwidth and server buffer space usage. Too aggressive prefetching can limit the number of concurrent streams that can be supported because of excessive server buffer usage [15]. Existing client-based smoothing algorithms do not have this problem, due to their implicit assumption of a fixed available client buffer size. The client buffer space need not be multiplexed among different streams as is the case when buffering is done at the server.

Intuitively, we propose a stream scheduling procedure that specifies for each stream both the variable server buffer and disk bandwidth requirements over time. A disk block b originally scheduled for round i may be prefetched in a previous round j only if: i) the disk bandwidth requirement in round j with the prefetched block does not exceed the original disk bandwidth requirement of round i, and ii) the fraction of server buffer required in each of the rounds j up to $i-1$, after prefetching block b, may not exceed the fraction of disk bandwidth required in round i without b. The first condition is necessary in order for the prefetching to have a smoothing effect on the disk bandwidth requirements over time. The second condition is a heuristic that we apply in order to prevent exhaustion of the server buffer. Both conditions are applied to individual streams, and we experimentally study their effect when serving multiple streams concurrently.

Knowing the data amount that needs to be retrieved from the disks during stream playback is important information that can be used during stream storage. Appropriate striping methods that take advantage of this information have been previously shown to achieve significantly increased system throughput with respect to striping methods of fixed-size block [1]. On the other hand, a retrieval sequence that is fixed a priori might ignore the exact resource tradeoffs that occur during system operation, when different stream playbacks are multiplexed. We evaluate later in detail the resource utilizations that are achieved with our approach of using the retrieval sequence to determine the striping method.

3.2 Limitations of Previous Approaches

For several reasons, previous client-based smoothing algorithms are inadequate for solving the server-based smoothing problem:

1. Unlike network transfer delays, disk access delays include mechanical movement overhead and cannot be accurately expressed in terms of bit rates only.

2. The prefetching constraints that we use span resources of different types and measures (e.g. access delays, data amounts) and it is difficult to describe them using data amounts only. This is not a problem, when the only constraint is the total buffer space available at the client.

3. Our constraints are complex and can only be conveniently expressed as inequalities continuously evaluated during the execution of the algorithm. There is no obvious way to represent them as fixed vectors initialized at the beginning of the algorithm.

Instead, we introduce a new smoothing algorithm that is more general than previous ones, and gives more flexibility and expressibility in representing the required optimization conditions.

3.3 Basic Definitions

We use a "smoothness" criterion that is based on *Majorization Theory* [17, 25]. For any $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, let the square bracket subscripts denote the elements of $\mathbf{x}$ in decreasing order $x_{[1]} \geq \cdots \geq x_{[n]}$. For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\mathbf{x}$ is

```
proc serverSmoothing
input : L_n, S_n[] ( =0 outside [1..L_d] ), B_l
output : L_d, S_d[], L_b, S_b[]
begin
  blockQuantize(L_n, S_n[], B_l) (* see App. B *)
  for t_rnd : 0..L_n-1
    if ( P_buf(S_b(t_rnd)) < P_dsk(S_d(t_rnd)) )
      repeat
        t_min := t_rnd
        P_min := max( P_dsk(S_d(t_rnd)), P_buf(S_b(t_rnd)) )
        t_prv := t_rnd - 1, prefFailed := false
        while ( prefFailed = false AND t_prv >= 0 )
          P_prf = max( P_dsk(S_d(t_prv) + B_l),
                       P_buf(S_b(t_prv) + B_l) )
          P_shf = max( P_dsk(S_d(t_prv)),
                       P_buf(S_b(t_prv) + B_l) )
          (*check for max proportion decrease*)
          if (P_prf < P_min)
            t_min = t_prv, P_min = P_prf
          else if (P_min < P_shf)
            prefFailed := true
          end-if
          t_prv := t_prv - 1
        end-while
        if (t_min < t_rnd) (* update vectors *)
          S_d(t_min) := S_d(t_min) + B_l,
          S_b(t_min) := S_b(t_min) + B_l
          for t_prv := t_min + 1 .. t_rnd - 1
            S_b(t_prv) := S_b(t_prv) + B_l
          end-for
          S_d(t_rnd) := S_d(t_rnd) - B_l
        end-if
      until (t_min >= t_rnd) (*prefetch search failed*)
    end-if
  end-for
end
```

Figure 2: The *Server Smoothing* algorithm generates majorization minimal disk sequence with the disk bandwidth proportion bounding above the corresponding server buffer proportion.

majorized by $\mathbf{y}$, $\mathbf{x} \prec \mathbf{y}$, if:

$$\sum_{i=1}^{k} x_{[i]} \leq \sum_{i=1}^{k} y_{[i]}, \quad k = 1, \cdots, n-1$$

and

$$\sum_{i=1}^{n} x_{[i]} = \sum_{i=1}^{n} y_{[i]}.$$

Then, we consider $\mathbf{x}$ smoother than $\mathbf{y}$, if $\mathbf{x} \prec \mathbf{y}$. Finally, we call a vector $\mathbf{x} \in \mathbb{R}^n$ *majorization-minimal* if there is no other vector $\mathbf{z} \in \mathbb{R}^n$ such that $\mathbf{z} \prec \mathbf{x}$.

From section 2.3, the disk time reservation for a disk transfer of X bytes is approximately equal to:

$$T_{disk}(X) = 2 \cdot (T_{trackSeek} + T_{avgRot}) + X/R_{disk}.$$

Before further explaining the algorithm, we introduce the following definitions.

Definition 1 Let the *Disk Time Proportion* of X bytes, $P_d(X)$, be the fraction of the round time T_{round} that the disk time reservation $T_{disk}(X)$ occupies: $P_d(X) = T_{disk}(X)/T_{round}$. Further let the *Buffer Space Proportion* of X bytes, $P_b(X)$, be the fraction of the buffer space for each disk, B_{disk},[1] that X bytes occupy in a round: $P_b(X) = X/B_{disk}$. Then, the *Maximum Resource Proportion* in round i, is the maximum of the corresponding disk time and buffer space proportions in that round: $max(P_d(S_d(i)), P_b(S_b(i)))$.

Definition 2 The *Deadline Round* for a block is the latest round at which the block can be accessed from the disk without incurring a real-time violation at the network transfer. Then, with respect to a specific block, all rounds before the deadline round are considered *Candidate Rounds* and the one actually chosen for prefetching is called the *Prefetch Round.* All the rounds between the deadline and a prefetch round are called *Shift Rounds.*[2]

Definition 3 We define as the *Maximum-Proportion Constraint* the requirement that the maximum resource proportion of the deadline round is *no less than* the maximum resource proportion of the corresponding (if any) prefetch and shift rounds.

3.4 The Algorithm

In the rest of this section we describe an algorithm that, given a stream network sequence $\mathbf{S_n}$ and a target server configuration, generates a smoothed disk sequence $\mathbf{S_d}$. We show that the generated disk sequence is majorization-minimal under the specified constraints. The generated disk sequence can be subsequently transformed into a striping sequence $\mathbf{S_m}$ according to some disk striping method, such as the Variable-Grain Striping.

The *Server Smoothing* algorithm of Figure 2 initially invokes the `blockQuantize` procedure (see appendix B) that generates disk and buffer sequences with data transfer sizes that are integral multiples of the logical block B_l. Network transfers are specified in byte granularity for increased flexibility (if necessary, they could be quantized too). Then, rounds of the generated sequences are visited in increasing order starting from round zero. For every logical block to be retrieved from the disk in the currently visited round, previous rounds are examined linearly in decreasing order towards round zero for potential prefetching of the block. Each such search completes successfully when a prefetch round is found such that the maximum resource proportion of the current round decreases while remaining *higher* than those of the prefetch and shift rounds. Below, we show that the algorithm works correctly.

Lemma 1 *The* Server Smoothing *algorithm chooses the prefetch round for each block in a way that satisfies the following properties: i) No network transfer timing violation occurs. ii) No maximum-proportion constraint violation occurs. iii) It has the lowest possible disk time proportion. iv) It is closest to the deadline round. Property (iii) prevails when in conflict with property (iv).*

Proof: Available in appendix A.

Definition 4 If $\beta_1 \geq \cdots \geq \beta_n$ are integers and $\beta_i > \beta_j$, then the transformation $\beta'_i = \beta_i - 1$, $\beta'_j = \beta_j + 1$, $\beta'_k = \beta_k$, for all $k \neq i, j$ is called a *transfer from i to j.*[3]

[1] B_{disk} is the total server buffer divided by the number of disks D.

[2] These definitions only affect the number of blocks accessed in each round, since stream block accesses are done sequentially during playback.

[3] The term *transfer* that we borrow from the original definition [19], should not be confused with regular data transfers.

Lemma 2 *(Muirhead, 1903 [19]) If $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n$ are integers and $\alpha \prec \beta$, then α can be derived from β by successive applications of a finite number of transfers.*

Proof: See Marshall and Olkin [17], pg. 135.□

In the presentation that follows *transfer* units correspond to logical blocks of size B_l as opposed to individual bytes.

Lemma 3 *The Server Smoothing algorithm produces disk sequence that satisfies the properties of Lemma 1, and has no transfer that would not violate them.*

Proof: Available in appendix A.

Theorem: 1 *The Server Smoothing algorithm generates a majorization-minimal disk sequence that satisfies the properties of Lemma 1.*

Proof: From Lemma 3, the disk sequence generated by the Server Smoothing algorithm satisfies the properties of Lemma 1 and has no *transfer* that would not violate them. Then, from Lemma 2, there is no other sequence that satisfies the properties of Lemma 1 and is majorized by the disk sequence generated by the *Server Smoothing* algorithm. If such a sequence existed, additional block *transfers* would be acceptable.□

The computational complexity of the *Server Smoothing* algorithm is $O(\frac{\sum_{i=1}^{L_n} S_n(i)}{B_l} L_n^2)$, where $\mathbf{S_n}$ is the input network sequence and L_n is its length. Although it may be possible to reduce this complexity, practically the application of this algorithm on a video stream of 30 minutes completes in tens of seconds in our experiments on a 133MHz RISC processor with $B_l = 16KB$, $L_n = 1,800$ and $\sum_{i=1}^{L_n} S_n(i) = 1.12 \cdot 10^9$. Since the schedule generation is done off-line, the above execution time is acceptable. The higher computational complexity relative to the $O(L_n)$ complexity of network smoothing algorithms [25] is the extra cost for avoiding the "hard-wired" fixed client buffer constraint. The *Server Smoothing* algorithm can generate majorization-minimal disk sequence with several buffer constraints, including the fixed buffer of network smoothing algorithms as a special case.

4. EXPERIMENTATION ENVIRONMENT

4.1 Prototype Overview

We have designed and built a media server experimentation platform, in order to evaluate the resource requirements of alternative stream scheduling techniques. The modules are implemented in about 12,000 lines of C++/ Pthreads code on AIX4.1. The code is linked either to the University of Michigan DiskSim disk simulation package [8], which incorporates advanced features of modern disks such as on-disk cache and zones for simulated disk access time measurements, or to hardware disks through their raw device interfaces [2]. The indexing metadata are stored as regular Unix files, and during operation are kept in main memory.

The basic responsibilities of the media server include file naming, resource reservation, admission control, logical to physical metadata mapping, buffer management, and disk and network transfer scheduling.

With appropriate configuration parameters, the system can operate at different levels of detail. In *Admission Control* mode, the system receives playback requests, does admission control and resource reservation, but no actual data transfers take place. In *Simulated Disk* mode, all the modules become functional, and disk request processing takes place using the specified DiskSim [8] disk array. There is also the *Full Operation* mode, where the system accesses hardware disks and transfers data to fixed client network addresses. For the experiments in the current study, we mostly used the Admission Control mode of our system, except for the validation in section 7 where Simulated Disk Mode was used.

Content Type	Avg Bytes per Round	Max Bytes per Round	CoV per Round
Science Fiction	624,935	1,201,221	0.383
Music Clip	624,728	1,201,221	0.366
Action	624,194	1,201,221	0.245
Talk Show	624,729	1,201,221	0.234
Adventure	624,658	1,201,221	0.201
Documentary	625,062	625,786	0.028

Table 1: We used six MPEG-2 video streams of 30 minutes duration each. The coefficient of variation shown in the last column changes according to the content type.

4.2 Performance Evaluation Method

We assume that playback initiation requests arrive independently of one another, according to a Poisson process. The system load can be controlled through the arrival rate λ of playback initiation requests. Assuming that the disk transfers form the bottleneck resource, we consider the ideal system, with no disk overhead when accessing disk data, as "perfectly efficient system." Then, we choose the maximum arrival rate $\lambda = \lambda_{max}$ of playback requests equal to the mean stream completion rate in that perfectly efficient system. This creates enough system load to show the performance benefit of arbitrarily efficient data striping policies. The mean stream completion rate μ, expressed in streams per round, for streams of average data size S_{tot} bytes becomes:

$$\mu = \frac{D \cdot R_{disk} \cdot T_{round}}{S_{tot}} \frac{streams}{round}. \tag{1}$$

The corresponding system load becomes: $\rho = \frac{\lambda}{\mu} \leq 1$, where $\lambda \leq \lambda_{max} = \mu$.

In the admission control process, when a playback request arrives, it is checked whether available resources exist for every round during playback. The test considers the exact data transfers of the requested playback for every round and also the corresponding available disk transfer time, network transfer time and buffer space in the system. If the request cannot be initiated in the next round, the test is repeated for each round up to $\lceil \frac{1}{\lambda} \rceil$ rounds into the future, until the first round is found where the requested playback can be started with guaranteed sufficiency of resources. Checking $\lceil \frac{1}{\lambda} \rceil$ rounds into the future achieves most of the potential system capacity as was shown previously [1]. If not accepted, the request is rejected rather than being kept in a queue.

As the basic performance metric we choose the average number of active playback sessions that can be supported by the server. The objective is to make this number as high as possible.

Seagate Cheetah ST-34501N	
Data Bytes per Drive	4.55 GB
Average Sectors per Track	170
Data Cylinders	6,526
Data Surfaces	8
Zones	7
Buffer Size	0.5 MB
Track to Track Seek(read/write)	0.98/1.24 msec
Maximum Seek(read/write)	18.2/19.2 msec
Average Rotational Latency	2.99 msec
Internal Transfer Rate Inner Zone to Outer Zone Burst Inner Zone to Outer Zone Sustained	 122 to 177 Mbit/s 11.3 to 16.8 MB/s

Table 2: Features of the Seagate SCSI disk assumed in our experiments.

4.3 Experimentation Setup

We used six different VBR MPEG-2 streams of 30 minutes duration each. Each stream has 54,000 frames with a resolution of 720x480 and 24 bit color depth, 30 frames per second frequency, and a $IB^2PB^2PB^2PB^2PB^2$ 15 frame Group of Pictures structure. The encoding hardware that we use allows the generated bit rate to take values between 1Mbit/s and 9.6Mbit/s. The statistical characteristics of the clips are given in Table 1. The coefficients of variation of bytes per round lie between 0.028 and 0.383, depending on the content type. In our *mixed* basic benchmark, the six different streams are submitted round-robin. Where appropriate, experimental results from individual stream types are also shown.

For experimentation with homogeneous disks, we assumed Seagate Cheetah SCSI disks, with the features shown in Table 2.[4] Except for the much larger storage capacity in the latest models, the rest of the performance numbers are typical of today's high-end drives. The logical block size B_l was set to $16KB$ bytes, while the physical sector size B_p was equal to 512 bytes. The stride size B_s in the disk space allocation was set to 2 MB. The server memory is organized in buffers of fixed size $B_l = 16KB$ bytes each, with a typical total space of 256 MB for every extra disk. (The effect of buffer space is examined later.) The available network bandwidth was assumed to be infinite, leaving contention for the network outside the scope of the current work.

In our experiments, the round time was set equal to one second.[5] We used a warmup period of 3,000 rounds and calculated the average number of active streams from round 3,000 to round 9,000. The measurements were repeated until the half-length of the 95% confidence interval was within 5% of the estimated mean value of the number of active streams.

5. STUDY OF HOMOGENEOUS DISKS

We start with a study of disk arrays consisting of functionally equivalent disks. In Figure 3, we depict the disk time and buffer space proportions in each round for a particular stream. Without smoothing (Fig. 3(a)), the occupied buffer space is the minimum necessary for data staging during disk and network transfers. With *Server Smoothing* (Fig. 3(b)), data are prefetched into the server buffer according to the maximum-proportion constraint. This keeps the maximum buffer space proportion to be no more than the maximum disk time proportion (8.7% in this example).

In Figure 4 we can see the number of active streams that can be sustained at different system loads and array configurations with between 4 and 64 disks using the mixed workload. In all the cases shown, the stream data have been striped according to the Variable-Grain Striping method. The Server-Smoothed plots show the performance benefit of applying the *Server Smoothing* algorithm assuming 256 MB of available server buffer space for each extra disk (we try later other server buffer sizes). At moderate load ($\rho = 50\%$),

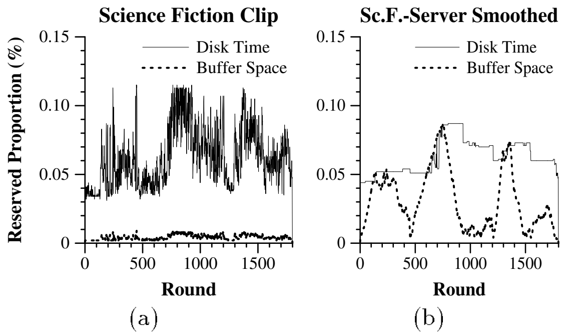

Figure 3: Example of applying the *Server Smoothing* algorithm. We depict the disk time and buffer space proportion in the system (a) before, and (b) after applying *Server Smoothing*. The maximum disk time proportion drops from 11.5% to 8.7%, while the maximum buffer space proportion increases from less than 1% to 8.7%. The Seagate Cheetah disk parameters are assumed and server buffer of 256 MB per disk.

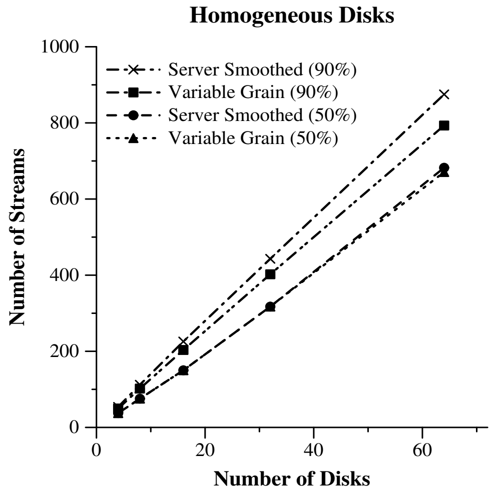

Figure 4: With the mixed workload and Variable-Grain Striping (with/without *Server Smoothing*), the sustained number of active streams increases almost linearly with the number of disks. Although at system load 50% all the submitted streams are accepted, at load 90% *Server Smoothing* increases the number of active streams by about 10%. This benefit is maintained across different numbers of disks.

[4]Note that one megabyte (megabit) is considered equal to 2^{20} bytes (bits), except for the measurement of transmission rates and disk storage capacities where it is assumed equal to 10^6 bytes (bits) instead [11].

[5]We found this round length to achieve most of the system capacity with reasonable initiation latency. This choice also facilitates comparison with previous work in which one second rounds were used.

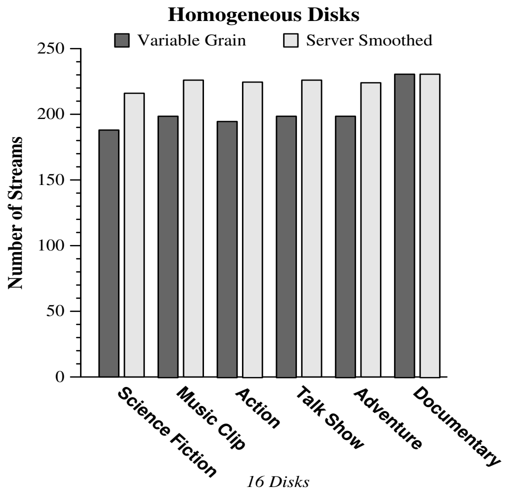

Figure 5: The advantage of *Server Smoothing* when applied to Variable Grain Striping can exceed 15% (Action) depending on the stream type. The load was set to 90% on sixteen disks and 256 MB per disk were assumed.

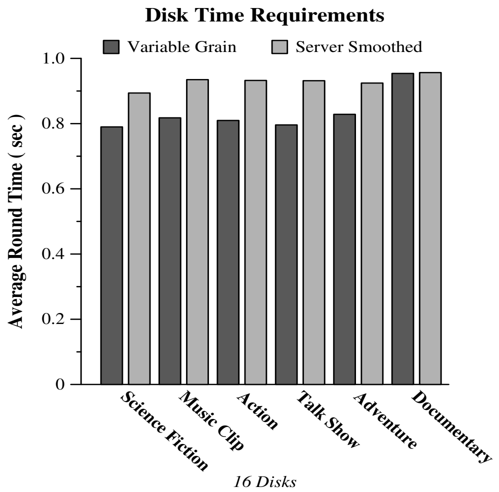

Figure 6: With sixteen disks and 90% system load, the average disk time reserved each round increases from less than 80% to over 90% with *Server Smoothing* and server buffer 256 MB per disk. Although the disk time shown corresponds to one of the disks (Disk 0) it was similar (typically within 2%) across the disks of the array.

Variable-Grain Striping with no smoothing allows all stream requests to be accepted. At a higher load ($\rho = 90\%$) the *Server Smoothing* can improve throughput by over 10%. The corresponding rejection rate (not shown) is 25% with Server Smoothing, and 41% with plain Variable Grain Striping, at 90% load.

Results for individual stream types are shown in Figure 5. We find that the benefit of *Server Smoothing* depends on the variability of data transfers across different rounds. Thus, although smoothing adds no benefit at streams with negligible variability (e.g. Documentary), as variation becomes higher, the increase in the number of streams can exceed 15% (Action).

Figure 6 shows the average total reserved disk time $T_{disk}(i, k)$ (expressed as percentage of round time) on a particular disk ($k = 0$) during the measurement period $3,000 \leq i < 9,000$. While for most stream types the average disk time hardly exceeds 80% of the round time with plain Variable-Grain Striping, it consistently approaches 90% and in several cases exceeds 93% (Action, Music Clip, Talk Show) when *Server Smoothing* is applied. This is remarkably high when compared to the 96% achieved by Documentary that has very low variation of data transfer sizes across different rounds.

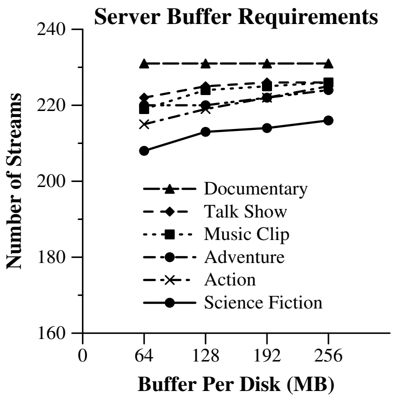

Figure 7: With buffer space (per disk) set to 64 MB, more than half of the total benefit of *Server Smoothing* can be achieved (see also Figure 5). Increasing the buffer space to 256 MB farther improves the number of streams in Science Fiction and Action types although at a diminishing degree.

Statistics gathered across the different stream workloads showed that the average occupied proportion of the available buffer space was about 50%, and the maximum hardly exceeded 60% at 90% load. Although in individually smoothed streams the buffer space requirement is allowed to reach that of disk bandwidth (in terms of proportions), the aggregate buffer space requirement turns out to be lower. This is a result of the way resource requirements of individual streams are multiplexed during system operation. The original constraint of preventing excessive prefetching from overflowing the available buffer space is still satisfied. Further increasing the aggregate buffer demand, without the buffer becoming a potential bottleneck in the system, would require incorporating into the algorithm information about the way stream requests are multiplexed.

In our experiments until now, we have assumed a server buffer of 256 MB per disk. From Figure 7 we can conclude that more than half of the *Server Smoothing* benefit is achieved with server buffer size as low as 64 MB per disk. We still believe that our assumption of 256 MB server memory (per disk) in the smoothing process is reasonable, however. The additional performance benefit from extra memory is sustained across different system sizes as was shown in Figure 4, with the purchase and administration cost of server memory being only a fraction of the costs associated with high-end disk drives.

6. STUDY OF HETEROGENEOUS DISKS

It has traditionally been assumed that disk arrays consist of homogeneous disks, presumably in order to keep the system complexity manageable. With the scalability of stream

HP-C3323A	
Data Bytes per Drive	1,052,491,776
Data Sectors per Track	72 to 120
Data Cylinders	2,910
Data Surfaces	7
Zones	8
Buffer Size	0.5 MB
Track to Track Seek	< 2.5 msec
Maximum Seek	22 msec
Rotational Latency	5.56 msec +/- 0.5%
Internal Transfer Rate Inner to Outer Zone Burst Inner to Outer Zone Sustained	 4.0 to 6.6 MB/s 2.8 to 4.7 MB/s

Table 3: Features of the HP SCSI disk that is included in the experiments for heterogeneous environments.

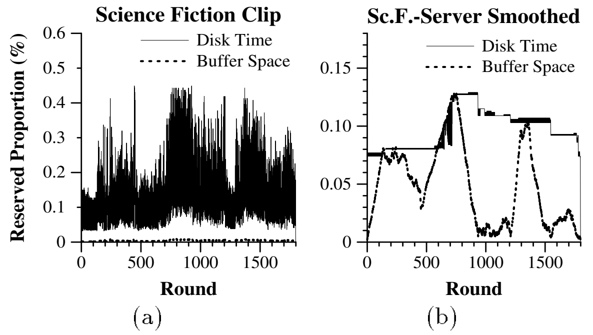

Figure 8: Example of stream *Server Smoothing*, in disk array configuration with Seagate and HP disks in alternating order. The much lower bandwidth of the older HP disk model leads to disk time proportion exceeding 40% in some of the rounds. When Server Smoothing is applied, disk transfers are appropriately adjusted to smooth out the peaks and keep the maximum reservation below 13%. At the same time, the maximum server buffer proportion is constrained to never exceed the maximum disk time proportion.

striping demonstrated recently [4, 1] and the sequential access of stored video making things somewhat simpler, it is interesting to investigate systems with different disk types that might have been upgraded incrementally with the newest disk technology. Newer disk models typically achieve higher transfer rates and have larger storage capacities.

In this section, we study the case of striping stream data across heterogeneous disk arrays. Our objective is to maximize the number of active streams by increasing the disk bandwidth utilization across all the disks. This might lead to suboptimal storage capacity utilization, which we assume is affordable given the current technology trends [9].

In our experiments, we assume disk arrays consisting of Seagate (Table 2) and older HP disks in alternating order. The features of the HP disks are shown in Table 3. We note a striking difference in the minimum internal transfer rate, which is 2.8 MB/s for the HP disks, one fourth as much as the 11.3 MB/s of the Seagate disks. Such a difference only makes the balancing of the system load more challenging. Although the experiments in this section assume an equal number of disks of each type, we also tried other ratios in the number of disk types and obtained similar results.

.

In Figure 8(a) we depict an example of a stream striped

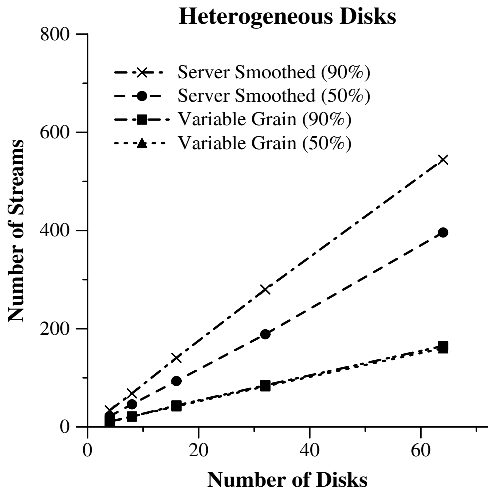

Figure 9: With the mixed workload and plain Variable-Grain Striping, the sustained number of active streams remains the same as the load is raised from 50% to 90%. Instead, when Variable-Grain Striping is combined with *Server Smoothing* the number of streams increases by a factor of 2 at 50% and more than a factor of 3 at 90% load, when compared to that achieved by plain Variable-Grain Striping. Server buffer space of 256 MB per disk has been assumed.

across an heterogeneous disk array. The lower transfer rate of the HP disks creates peaks of disk time proportion that can exceed 40%. In order to alleviate this problem, we extend the *Server Smoothing* algorithm to handle heterogeneous disks. In particular, we redefine the disk time $T_{disk}(X)$ and the disk time proportion function $P_d(X)$ to accept a second disk type argument k that specifies the particular disk parameters to be used $P_d(X,k) = \frac{T_{disk}(X,k)}{T_{round}}$. During the operation of the *Server Smoothing* algorithm, the disk type k assumed in each round i can be derived using a simple rule, such as $k = i(mod\ D)$, where D is the total number of the disks. Finally, if R^k_{disk} is the minimum internal transfer rate of disk k, the service rate definition of Eq. (1) becomes: $\mu = (T_{round} \cdot \sum_{k=0}^{D-1} R^k_{disk})/S_{tot}\ \frac{streams}{round}$.

We applied the extended *Server Smoothing* algorithm on the stream example of Figure 8(a). The generated transfer sequence, shown in Figure 8(b), has its buffer space proportion bounded by the disk time proportion, as before. In addition the maximum disk time proportion dropped from over 40% to less than 13%, after the transfer sizes across different rounds were appropriately adjusted according to the available disk bandwidth.

In Figure 9, we compare the performance of plain Variable-Grain Striping to that of Variable-Grain Striping with Server Smoothing in a range of heterogeneous disks between 4 and 64. Although the number of streams always increases almost linearly with the number of disks, *Server Smoothing* can achieve an advantage that exceeds a factor of 2 and 3 at loads of 50% and 90%, respectively. The reason is that the limited disk bandwidth of the HP disks, prevents the Seagate disks from attaining high bandwidth utilization without appropriate adjustment of the disk transfers. A similar behavior is also demonstrated across different stream types in Figure 10. With plain Variable-Grain Striping, the number of supported streams on sixteen disks hardly exceeds 50;

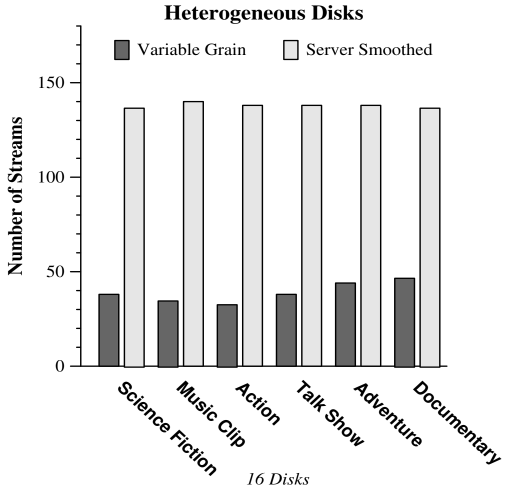

Figure 10: Applying *Server Smoothing* on different stream types, can lead to an increase in the number of streams by more than a factor of 3 at 90% load and server buffer 256 MB per disk.

when Server Smoothing is added the number of streams gets close to 140.

In Figure 11, we depict the average time that the Seagate and HP disks are expected to be busy respectively during each round. We show the statistics for disks 0 and 1 only, since the statistics for the rest of the disks were similar. As we see, the average time that the Seagate disks are busy is less than 25% of the round time with plain Variable-Grain Striping. The reason is the low bandwidth of the HP disks, whose corresponding average time varies between 50% and 80%; it cannot get higher due to the relatively large data requirements of the individual streams. When Server Smoothing is applied, a high average reserved disk time that gets close to 90% is achieved across all the disks of the disk array. This becomes possible with appropriate data prefetching that distributes data accesses across the disks according to the bandwidth that they can support.

In the previous experiments we set the server buffer to 256 MB per disk. However, as we can see from Figure 12, having only 64 MB per disk is sufficient to get most of the benefits of Server Smoothing for the particular streams included in our benchmark.

7. VALIDATION IN SIMULATED DISK MODE

In order to keep the computation time reasonable, the previous experiments were conducted with our system in Admission Control mode, where resource reservations are made for arriving playback requests, but without actual time measurement of the individual disk transfers. In the present section, we use Simulated Disk Mode to compare the statistics of the disk time reservations with the statistics gathered over the access times of all individual data transfers involved, using the DiskSim representation of the Seagate Cheetah and HP C3323A disks [8]. A four-disk array model is used with the disk types in alternating order. Each disk is presumed to be attached to a separate 20 MB/sec SCSI bus, and no contention is assumed on the host system bus connecting the SCSI buses. The statistics are gathered during 6,000 rounds after a warmup period of 3,000 rounds, as before and the mixed workload is used. The server can support 9.74 active streams with plain Variable-Grain Striping and 33.87 active streams with Server-Smoothed Variable-Grain Striping corresponding to a 90% load.

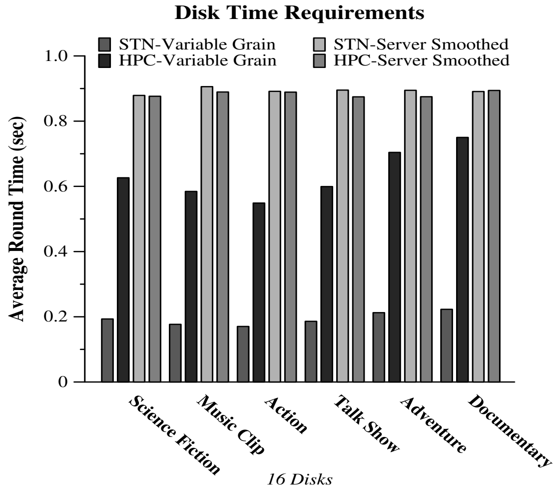

Figure 11: The two leftmost bars of each stream type, show the average reserved disk time for the Seagate (STN) and HP (HPC) disks, assuming plain Variable Grain Striping and 90% load. The lower transfer bandwidth of the HP disk creates a bottleneck keeping the reserved disk time of the Seagate disk to less than 25% of the round time. As is shown by the two rightmost bars though, with Server Smoothing both disk types attain average disk time close to 90% of the round time.

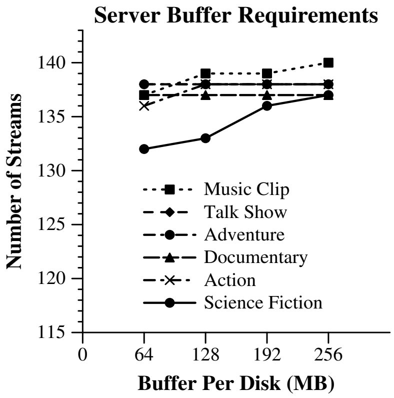

Figure 12: With the server buffer (per disk) set to 64 MB, most of the benefit of *Server Smoothing* can be attained (see also Figure 10). Increasing the server buffer from 64 MB to 256 MB increases only marginally (less than 5%) the sustained number of active streams.

As can be seen from Figure 13, in both the average and maximum case, the reserved disk time is no more than 8% higher than the corresponding measurements on the Seagate disk model by Ganger et al.[8]. This gap can be attributed to the fact that our disk time reservation assumes a minimum disk transfer rate and ignores on-disk caching. The corresponding gap for the HP disks gets close to 15% with

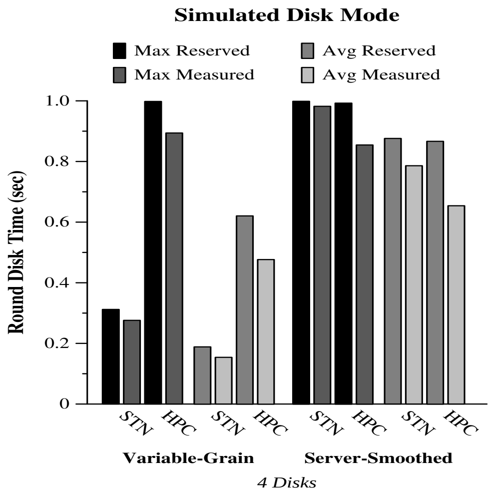

Figure 13: In an array of four disks, Seagate (STN) and HP (HPC) models are used in alternating order. The average and maximum reserved and measured time is shown for the disks 0 and 1 of the array with the mixed workload at 90% load. On the STN disks, the reserved statistics are no more than 8% higher than the measured. On the HPC disk, the corresponding difference can get up to 20%. The measurements have been done using the detailed disk simulation models by Ganger et al.

plain striping and 20% with *Server Smoothing*. Possible reasons for the larger discrepancy with the HP disks are the increased on-disk cache locality due to the smaller disk capacity, and the higher probability that only one head movement is required with the smaller data transfers (smoothed case).

In general, we believe that the achieved accuracy in the disk time predicted by the resource reservation is adequate. In fact, to improve these estimates, it would probably be necessary to have extra disk geometry information that is not readily available for commercial disk drives [30].

8. RELATED WORK

Several smoothing techniques deal with network link transfers of stored video streams. Salehi et al. describe a network smoothing technique to minimize the variability of network bandwidth requirements assuming a fixed-size client buffer [25]. Feng and Rexford compare the scheme of Salehi et al. with alternative schemes that minimize the total number of network bandwidth increases or decreases [7]. McManus and Ross introduce a dynamic programming methodology for scheduling network transfers [18]. Zhao and Tripathi describe a class of algorithms that minimize the maximum required network bandwidth when multiplexing stream network transfers to multiple clients [31]. All of these studies are complementary to our work, since our algorithm focuses on the management of the server disk bandwidth and server buffer space, and can accommodate any valid specification of data amounts that should be sent to the client over time.

Other related research tries to improve network link utilization for live video. Rexford et al. use client data prefetching for smoothing live video streams, where the stream requirements are known only for a limited period of time instead of the entire playback period [23]. Mansour et al. examine several resource tradeoffs in live video smoothing [16]. Ideas from live video smoothing are combined with prefix caching for smoothing streams in network proxies, assuming extra knowledge of quality of service parameters about the client not usually available at the server [27]. This study is consistent with our own assumptions about limited knowledge of client resources at the server.

Several studies have considered server-side resource management. Patterson et al. apply a cost-benefit analysis in order to control the disk bandwidth versus data buffer size tradeoff for general applications [21]. Paek and Chang propose an approach that, given a set of streams, optimizes a "general objective function" by controlling the maximum disk bandwidth and buffer space available to each stream [20]. One difference relative to our server-based smoothing algorithm is that we determine the memory-bandwidth tradeoff before stream storage, and we use it during striping. Reddy and Wijayaratne have experimented with the effect of client-based smoothing on alternative disk striping methods; they point out the need for also studying server-based prefetching techniques in their future work [22]. Other schemes that have been proposed require that a certain amount of data be retrieved into the server buffer before playback can start, which reduces responsiveness [3, 14]. In addition, previous studies are limited to single disk systems with fixed-size transfers only [3]. It has been shown that allowing variability in the transfer sizes can increase system throughput [1].

Sahu et al. [24] find a limited smoothing effect when increasing the round time in variable-size transfers and the block size in fixed-size transfers. Additional smoothing benefit is achieved with deadline-based scheduling of disk transfers that minimize the maximum required buffer space and disk bandwidth. However, that study is limited to single-disk systems. It remains unclear how deadline-based algorithms for admission control and disk scheduling are extended and actually perform when data are striped across multiple disks. The striping method itself can affect significantly the disk access efficiency and the data prefetching flexibility for reducing the load imbalance across the system.

Kim et al.[12] outline some ideas on how to control the tradeoff between buffer and disk bandwidth utilization in stored video streaming. Their work differs from ours in several aspects, and they specify no concrete algorithm for the problem. Their approach divides the stream into arbitrary length segments according to an "empirical threshold", alpha. Prefetching is done only within a segment, and it is controlled by an "empirical threshold", beta. Their disk bandwidth definition ignores the disk arm movement delays and the round duration, and their simulation study is limited to single disk systems only. In contrast, we propose a smoothing algorithm that prevents the required proportion of server buffer from exceeding the correspondingly decreased proportion of the required disk bandwidth. We prove that the algorithm achieves optimally smoothed disk transfer sequence under the specified constraints.

A significant amount of previous work also addresses efficient retrieval of stream data from heterogeneous disks. Dan and Sitaram suggest that multiple heterogeneous storage devices may coexist in a video server environment [6]. Considering the complexity of striping data across all the devices, they propose clustering homogeneous devices into groups and describe a dynamic data placement policy to keep the bandwidth and storage space utilization high. Chou et al. propose dynamic object replication techniques for improving utilization across different groups of disks [5]. Santos and Muntz use randomized replication techniques for load balancing of heterogeneous disk arrays [26], while other studies try to achieve that with alternative disk array organizations and striping methods [29, 32]. Unlike all these methods, which are applicable (or have been demonstrated to work) only for constant rate streams, we propose a smoothing technique for efficiently striping variable bit-rate streams across heterogeneous disks.

9. CONCLUSIONS

Recently, thin client devices have emerged that are soon expected to outnumber powerful desktop computers. They motivate the development of resource management policies that make minimal assumptions about the available client capabilities.

In this paper, we introduce the *Server Smoothing* algorithm that uses data prefetching into server buffers for smoothing disk transfers of variable bit-rate streams. The algorithm is greedy and is shown to have optimal smoothing effect under the specified constraints. Experimentation with homogeneous disk arrays and moderate server buffer space demonstrates that *Server Smoothing* can achieve more than 15% increase in the number of streams that can be supported by the server. This benefit is sustained across different numbers of disks that we examine.

We also use the *Server Smoothing* algorithm for striping variable bit-rate streams across arrays of heterogeneous disks. When plain disk striping is used, disks with lower transfer rates prevent the system from reaching high utilization. With *Server Smoothing*, the average reserved disk access time can get as high as 90% of the round time across the different disks. The corresponding benefit in the number of streams accepted by the server exceeds a factor of three for the particular disk array configuration that we use.

One important issue that remains open is designing appropriate replication techniques for tolerating disk failures in heterogeneous disk environments. In our future work, we also plan to consider potential contention in the network, and include in our experimentation, alternative network smoothing techniques as well.

10. REFERENCES

[1] Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. Disk Striping Scalability in the Exedra Media Server. In *ACM/SPIE Multimedia Computing and Networking Conf.* (San Jose, CA, Jan. 2001), pp. 175–189.

[2] Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. Modular and Efficient Resource Management in the Exedra Media Server. In *USENIX Symposium on Internet Technologies and Systems* (San Francisco, CA, Mar. 2001). (to appear).

[3] Biersack, E. W., and Hamdi, M. Cost-optimal Data Retrieval for Video Servers with Variable Bit Rate Video Streams. In *Intl. Work. Network and Operating System Support for Digital Audio and Video* (Cambridge, UK, July 1998), pp. 295–302.

[4] Bolosky, W. J., Barrera, J. S., Draves, R. P., Fitzgerald, R. P., Gibson, G. A., Jones, M. B., Levi, S. P., Myhrvold, N. P., and Rashid, R. F. The Tiger Video Fileserver. In *Intl. Work. on Network and Operating System Support for Digital Audio and Video* (Zushi, Japan, Apr. 1996), pp. 97–104.

[5] Chou, C. F., Golubchik, L., and Lui, J. C. S. A Performance Study of Dynamic Replication Techniques in Continuous Media Servers. In *ACM SIGMETRICS* (Atlanta, GA, May 1999), pp. 202–203.

[6] Dan, A., and Sitaram, D. An Online Video Placement Policy based on Bandwidth to Space Ratio (BSR). In *ACM SIGMOD* (San Jose, CA, May 1995), pp. 376–385.

[7] Feng, W.-C., and Rexford, J. A Comparison of Bandwidth Smoothing Techniques for the Transmission of Prerecorded Compressed Video. In *IEEE INFOCOM* (Kobe, Japan, Apr. 1997), pp. 58–66.

[8] Ganger, G. R., Worthington, B. L., and Patt, Y. N. The DiskSim Simulation Environment: Version 2.0 Reference Manual. Tech. Rep. CSE-TR-358-98, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, Dec. 1999.

[9] Gray, J., and Shenoy, P. Rules of Thumb in Data Engineering. In *IEEE Intl. Conf. Data Engineering* (San Diego, CA, Feb. 2000), pp. 3–10.

[10] Gringeri, S., Shuaib, K., Egorov, R., Lewis, A., Khasnabish, B., and Basch, B. Traffic Shaping, Bandwidth Allocation, and Quality Assessment for MPEG Video Distribution over Broadband Networks. IEEE Network, 6 (Nov/Dec 1998), 94–107.

[11] *The IBM Dictionary of Computing*. McGraw-Hill, New York, NY, 1994.

[12] Kim, I.-H., Kim, J.-W., Lee, S.-W., and Chung, K.-D. VBR Video Data Scheduling using Window-Based Prefetching. In *IEEE Multimedia Computing and Systems* (Florence, Italy, June 1999), pp. 159–164.

[13] Lakshman, T. V., Ortega, A., and Reibman, A. R. VBR Video: Tradeoffs and Potentials. *Proceedings of the IEEE* **86**, 5 (May 1998), 952–973.

[14] Lee, D.-Y., and Yeom, H. Y. Tip Prefetching : Dealing with the bit rate variability of video streams. In *IEEE Multimedia Computing and Systems* (Florence, Italy, June 1999), pp. 352–356.

[15] Makaroff, D., Hutchinson, N., and Neufeld, G. An Evaluation of VBR Disk Admission Algorithms for Continuous Media File Servers. In *ACM Multimedia* (Seattle, WA, May 1997), pp. 143–154.

[16] Mansour, Y., Patt-Shamir, B., and Lapid, O. Optimal Smoothing Schedules for Real-Time Streams. In *ACM Principles of Distributed Computing* (Portland, OR, July 2000).

[17] Marshall, A. W., and Olkin, I. *Inequalities: Theory of Majorization and its Applications*. Academic Press, New York, 1979.

[18] McManus, J., and Ross, K. A Dynamic Programming Methodology for Managing Prerecorded VBR Sources in Packet-Switched Networks. Telecommunications Systems **9** (1998), 223–247.

[19] Muirhead, R. F. Some methods applicable to identities and inequalities of symmetric algebraic functions of n letters. In *Proc. Edinburgh Mathematical Society* (1903), vol. 21, pp. 144–157.

[20] Paek, S., and Chang, S.-F. Video Server Retrieval Scheduling for Variable Bit Rate Scalable Video. In *IEEE Multimedia Computing and Systems* (Hiroshima, Japan, June 1996), pp. 108–112.

[21] Patterson, R. H., Gibson, G. A., Ginting, E., Stodolsky, D., and Zelenka, J. Informed Prefetching and Caching. In *ACM Symp. Operating Systems Principles* (Copper Mountain Resort, CO, Dec. 1995), pp. 79–95.

[22] Reddy, A. L. N., and Wijayaratne, R. Techniques for improving the throughput of VBR streams. In *ACM/SPIE Multimedia Computing and Networking* (San Jose, CA, Jan. 1999), pp. 216–227.

[23] Rexford, J., Sen, S., Dey, J., Feng, W., Kurose, J., Stankovic, J., and Towsley, D. Online Smoothing of Live, Variable-Bit-Rate Video. *IEEE Trans. on Multimedia* **2**, 1

(Mar. 2000), 37–48.

[24] Sahu, S., Zhang, Z.-L., Kurose, J., and Towsley, D. On the Efficient Retrieval of VBR Video in a Multimedia Server. In *IEEE Multimedia Computing and Systems* (Ottawa, Canada, June 1997), pp. 46–53.

[25] Salehi, J. D., Zhang, Z.-L., Kurose, J. F., and Towsley, D. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. In *ACM SIGMETRICS* (Philadelphia, PA, May 1996), pp. 222–231.

[26] Santos, J. R., and Muntz, R. Performance Analysis of the RIO Multimedia Storage System with Heterogeneous Disk Configurations. In *ACM Multimedia* (Bristol, UK, Sept. 1998), pp. 303–308.

[27] Sen, S., Rexford, J., and Towsley, D. Proxy Prefix Caching for Multimedia Streams. In *IEEE INFOCOM* (New York, NY, Mar. 1999), pp. 1310–1319.

[28] Shenoy, P. J., and Vin, H. M. Efficient Striping Techniques for Multimedia File Servers. In *Intl. Work. Network and Operating Systems Support for Audio and Video* (St. Louis, MO, May 1997), pp. 25–36. An expanded version appears in Performance Evaluation Journal, vol. 38, June 1999, pg. 175-199.

[29] Wang, Y., and Du, D. H. Weighted Striping in Multimedia Servers. In *IEEE Multimedia Computing and Systems* (Ottawa, Canada, June 1997), pp. 102–109.

[30] Worthington, B. L., Ganger, G. R., Patt, Y. N., and Wilkes, J. On-Line Extraction of SCSI Disk Drive Parameters. In *ACM SIGMETRICS* (Ottawa, Canada, May 1995), pp. 146–156.

[31] Zhao, W., and Tripathi, S. K. Bandwidth-Efficient Continuous Media Streaming Through Optimal Multiplexing. In *ACM SIGMETRICS* (Atlanta, GA, June 1999), pp. 13–22.

[32] Zimmermann, R., and Ghandeharizadeh, S. Continuous Display Using Heterogeneous Disk-Subsystems. In *ACM Multimedia* (Seattle, WA, Nov. 1997), pp. 227–328.

APPENDIX

A. PROOFS OF LEMMAS 1 AND 3

Proof of Lemma 1: The property (i) comes from lines 10-11,22 of the algorithm, which limit the range of prefetching rounds to those preceding the current one. The property (ii) is due to lines 17,19 which ensure that the maximum resource proportion of the deadline round is no less than that of the prefetch and shift rounds respectively. The candidate round with minimal disk time proportion (iii) is kept track of through variable P_{min} in line 18. Finally, the closeness to the deadline (iv) is controlled by the descending loop at lines 22, and the strict inequality in line 17. □

Proof of Lemma 3: The algorithm is "greedy" and we will use induction on the network sequence length L_n. The generated disk sequence trivially satisfies the lemma claim at $L_n = 1$, with round 0 to access the data from disk and round 1 to send the data over the network. We assume that at $L_n = k$ the claim is valid. We show that it is also valid for $L_n = k + 1$. Let us assume that we get the sequence of the k first disk accesses $0 \dots k - 1$ to satisfy the lemma claim, before starting to deal with the disk access of round k. Due to the property (i) of Lemma 1, it is not possible to schedule in round k, block accesses from the previous rounds. Therefore, the only acceptable *transfer* of blocks that remains is moving blocks of the round k to previous rounds. An exhaustive search is done in the while-loop of the lines 11-23 of the algorithm. Each of the previous rounds is visited, and a record is kept of the closest round that can prefetch a block with minimal total disk time proportion from property (iii). Property (ii) guarantees no violation of the maximum-proportion constraint. The above search is repeated by the repeat-loop of lines 7-32 until no more *transfers* of logical blocks belonging to round k are possible. (The decision to choose prefetch rounds closest to the deadline round, from property (iv), leads to buffer occupancy minimization.) □

B. THE BLOCKQUANTIZE PROCEDURE

```
proc blockQuantize
input : L_n, S_n[] ( =0 outside [1..L_n] ), B_l
output : L_d, S_d[], L_b, S_b[]
begin
  S_d[] := 0, S_b[] := 0
  L_d := L_n, L_b := L_n + 1
  totSn := 0
  for t_rnd : 0..L_n
    prvSn := totSn, totSn := totSn + S_n(t_rnd + 1)
    (* we use function ceil() for the ⌈ ⌉ operation *)
    S_d(t_rnd) := B_l · ( ceil(totSn/B_l) − ceil(prvSn/B_l) )
    S_b(t_rnd) := S_b(t_rnd − 1) + S_d(t_rnd) - S_n(t_rnd − 1)
  end-for
end
```