Available online at www.sciencedirect.com

SCIENCE DIRECT®

Performance Evaluation 59 (2005) 47–72

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

Shared-buffer smoothing of variable bit-rate streams

Stergios V. Anastasiadis[a,*], Kenneth C. Sevcik[b], Michael Stumm[c]

[a] *Department of Computer Science, Duke University, Durham, NC 27708, USA*
[b] *Department of Computer Science, University of Toronto, Toronto, ON, Canada M5S 3G4*
[c] *Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada M5S 3G4*

Received 1 July 2003; received in revised form 24 May 2004

Available online 1 September 2004

Abstract

We study network servers that transmit variable bit-rate streams for real-time playback at remote clients. We introduce an algorithm that removes peaks of disk bandwidth by prefetching stored stream data into the shared buffer space of the server. Using a mathematical framework, we show that our algorithm has optimal smoothing effect to the server disk bandwidth over time. Emergence of inexpensive specialized devices makes prevalent the assumption of limited hardware resources for playback clients, and insufficient previous techniques that can only prefetch stream data into the client buffer space. We incorporate our algorithm into a prototype server, and demonstrate significant increase in the number of streams concurrently supported at different system scales. We also extend our algorithm to stripe variable bit-rate streams across heterogeneous disks. We achieve high bandwidth utilization across all the different disks, and improve the server throughput by several factors at high loads.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Variable bit-rate streams; Continuous media streaming; Video servers; Smoothing; Storage systems

1. Introduction

Bit-rate variability of media streams is a feature of media encoding that allows automatically adjusting the rate of media bits generated over time according to some optimization objective. In particular,

* Corresponding author.

E-mail addresses: stergios@cs.duke.edu (S.V. Anastasiadis); kcs@cs.toronto.edu (K.C. Sevcik); stumm@eecg.toronto.edu (M. Stumm)

0166-5316/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2004.06.001

variable bit-rate encoding keeps approximately constant the perceptual quality of the generated stream, and produces bits at a rate that changes instantly but remains on average constant [14]. Alternatively, *constant* bit-rate encoding keeps at a configurable fixed level the generated bit rate, while it changes the perceptual quality according to the information content of the input. Therefore, the difference between the two encoding schemes lies on whether the generated bit rate remains constant on average only or always, and whether the perceptual quality is constant or variable, respectively.

Modern encoders typically support both constant and variable bit-rate encoding. Nevertheless, variable bit-rate encoding of video streams can achieve quality equivalent to that of constant bit-rate encoding while requiring average bit rate that is lower by 40% or more [13,17]. On the other hand, variable bit-rate streams have high variability in their resource requirements, which potentially leads to low utilization of disk and network bandwidth. This occurs because the aggregate bandwidth requirements of concurrently served streams can be significantly higher at particular time instances than on average. Also, the admission control process typically bases its decisions on peak aggregate demand when considering new stream requests for playback.

In order to improve the resource utilization and the throughput of a media streaming system, a number of schemes have been proposed previously. They remove peaks of transfer bandwidth in individual streams by appropriately prefetching stream data during periods of lower bandwidth demand in the system. Such prefetching schemes are better known as *smoothing* techniques, and have been shown to achieve significant performance improvements in several cases.

Previously proposed smoothing techniques have been limited to prefetching data from the media server into the buffer space of the clients. Such techniques can improve the utilization of both the disk and the network bandwidth, but they are 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 within the server only and combining them appropriately with the disk striping and admission control policies of the server. Thus, we introduce a stream smoothing algorithm that prefetches data into the shared buffer space of the server rather than the private buffer space of the client. The algorithm has several important advantages in comparison to previous approaches:

- We reduce the required disk bandwidth in the server, even when serving inexpensive mass-produced clients of minimal buffer space. This is important since historically disk bandwidth has been improving at rates an order of magnitude slower than network link bandwidth [12].
- Since we treat all the clients the same, we only store one stream copy in the system. Thus we reduce the admission control complexity, and the required storage space in the streaming server.

Server-side smoothing targets the required disk bandwidth rather than the network bandwidth, and accepts as input a specification of the data that should be sent to the client over time. When the clients have sufficient buffer space, server smoothing can be complemented with network smoothing techniques, thus taking advantage of prefetching into the buffers of both the server and the client. In order to prevent excessive prefetching from exhausting the available buffer space in the server, we prevent the increased proportion of server buffer required by a stream to exceed the reduced proportion of disk bandwidth (more formally explained later). Thus, the smoothing process is adjusted automatically according to the total memory and disk bandwidth available in the server configuration. We apply the smoothing algorithm off-line to each individual stream, and we use the generated transfer schedule for subsequently storing and accessing the stream data.

Another issue that we study in the present paper is smoothing of *variable* bit-rate streams striped across heterogeneous disks. This is a problem that has not been extensively studied in the past, probably because load-balancing and reliability problems were assumed to restrict the size of disk arrays across which stream data can be striped efficiently [8,32]. Nevertheless, recent research has introduced disk striping schemes that are scalable [2,6]. Operating efficiently a server with heterogeneous disks is important because it enables server installations to be incrementally expanded using the most advanced and cost-efficient storage devices as the system load increases.

Historically, the ratio between disk *storage capacity* and disk *bandwidth* increases by a factor of ten every decade [12]. Thus, disk accesses are becoming more precious, which justifies our decision to focus on the disk bandwidth. We implemented our smoothing scheme into a prototype server. Using experiments with various MPEG-2 streams, we demonstrate 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.

In summary, our main contribution in the present paper is to introduce a general smoothing algorithm that uses the shared buffer space of the server to prefetch stored stream data. We prove that the algorithm optimally smooths the disk bandwidth of the server over time, and prevents the increased buffer utilization from exceeding the reduced disk bandwidth utilization. Additionally, we use the same algorithm to balance the bandwidth utilization of heterogeneous disks under streaming workloads. We experimentally demonstrate significant benefits in the throughput and the resource utilization of the system.

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

2. 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 [29]. Feng and Rexford compare the scheme of Salehi et al. with alternative schemes that minimize the total number of network bandwidth variations over time [10]. McManus and Ross introduce a dynamic programming methodology for scheduling network transfers [22]. All these cases assume a fixed amount of buffer space available for each individual client, while in the present paper we investigate the case of prefetching data into buffer space shared across multiple clients in their server. We made a preliminary presentation of the results described in the present paper elsewhere [4], although not as clearly and self-dependently as we present them here.

Zhao and Tripathi describe a class of algorithms that minimize the maximum required network bandwidth when multiplexing stream network transfers to multiple clients [35]. In later work, that result was extended to actually smooth (rather than just minimize the peak of) multiplexed traffic using buffers shared or partitioned among different clients [1]. Although handling multiplexed traffic is theoretically a more difficult problem than treating each stream individually, which we present here, practically the two problems are complementary. In fact, traffic multiplexing takes place during the time of dynamically serving playback requests and after the streams have been stored on the disks, while individual stream smoothing occurs in advance and affects the way each stream is striped across multiple disks.

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 [27]. Mansour et al. examine several resource tradeoffs in live video smoothing [20]. Sen et al. combine ideas from live video smoothing with prefix caching to smooth streams in network proxies, that offer knowledge about the configuration parameters of the clients unknown to the server [31]. 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 traditional applications [25]. 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 [24]. One difference relative to our server-based smoothing algorithm is that we determine the memory-bandwidth tradeoff before the stream storage, and achieve optimal smoothing of the disk bandwidth. 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 [26]. 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 the system responsiveness to client requests [5,18]. In addition, previous studies are limited to single disk systems with fixed-size transfers only [5]. However, in previous work we have shown that allowing variability in the transfer sizes can increase the system throughput [2].

Sahu et al. [28] find a limited smoothing effect when increasing the round time in variable-size transfers and the block size in fixed-size transfers. They achieve additional smoothing benefit 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. [16] outline some ideas on how to control the tradeoff between buffer and disk bandwidth utilization in stored video streaming. However, their work differs from ours in several aspects, and they specify no concrete algorithm for the problem. For example, they use empirical parameters to divide streams into segments of arbitrary length, and to control data prefetching within each segment. Their disk bandwidth definition ignores the disk head movement delays, and their simulation study is limited to single disk systems. In contrast, we propose a smoothing algorithm that prevents the proportion of server buffer to exceed the maximum proportion of disk bandwidth, and we prove that our 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. Recently, Denehy et al. explored dynamic balancing of heterogeneous disks in RAIDs running traditional workloads [9]. They use online performance measurement of individual disks and dynamically direct block writes to the faster devices. Earlier, Dan and Sitaram suggested that multiple heterogeneous storage devices may coexist in a video server environment [8]. 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 [7]. Santos and Muntz use randomized replication techniques for load balancing of heterogeneous disk arrays [30], while other studies try to achieve that with alternative disk array organizations and striping methods [33,36]. Unlike all these methods, which are applicable to (or have been demonstrated to work with) constant rate streams only, we propose a smoothing technique for efficiently striping variable bit-rate streams across heterogeneous disks.

3. System architecture

In the present section, we outline important aspects of the system architecture that we assume in our algorithm. We have presented elsewhere [2,3] more details about the design and the implementation of a system that satisfies these requirements.

3.1. Overview

We assume a system that 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 are organized in rounds of fixed duration T_{round}, during which we retrieve an appropriate amount of data from the disks into a set of buffers that we reserve in the server. Concurrently, we send data from the server to the client buffer through the network interfaces (Fig. 1). During each round, a client must receive the amount of data that will be needed by the playback during the next round. Any transfer scheduling policy that does not violate this timing requirement and does not overflow the buffers of the client is acceptable for our purposes.

The streams are compressed according to any encoding scheme (e.g. MPEG-2) that supports streams of constant perceptual quality at variable bit-rate. We store the stream data across multiple disks, as shown in Fig. 1. Playback requests arriving from the clients are initially directed to an admission control module, where we determine whether enough resources exist to activate a playback session either immediately or within a limited number of rounds. A schedule database maintains for each stream detailed information

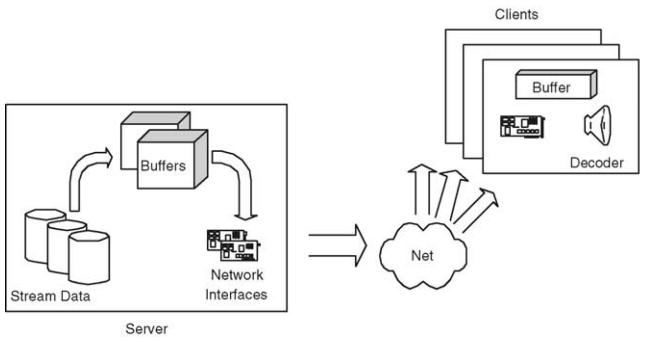

Fig. 1. Variable bit-rate video streams are stored across multiple disks of the media server. Multiple clients can connect and start playback sessions via a high-speed network.

about the data that should be accessed from each disk at any given round, the amount of server buffer space required, and the data that need 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 control of data transfers during playback. It is also possible that two or more replicas are available for each stream file, with different storage layout and retrieval schedules.

3.2. Stride-based disk space allocation

In our experiments, we allocate disk space in large fixed-sized chunks, called *strides*, according to the *stride-based allocation* method [2]. We choose the stride to be larger than the maximum stream request size per disk during a round. We know this size because we access stored streams sequentially according to a predefined (potentially variable) rate. During a playback round, we only fetch into memory the requested amount of stream data, and not the entire stride. Stride-based allocation eliminates external fragmentation, because the strides have fixed size. It also keeps negligible the internal fragmentation since we only waste disk space in the last stride of each stream; streams are large in size relative to strides, while a stride may contain data of more than one round. Another advantage of stride-based allocation is to set 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. We describe in the next section, how to allocate strides when we distribute data across multiple disks.

3.3. Reservation of server resources

In order to describe the resource reservation scheme that we use, we introduce a mathematical abstraction of the allocated resources in the system. We summarize the symbols that we use for our definitions in Table 1. For now, we consider a system with D functionally equivalent disks, although later we examine a more general case of heterogeneous environments. In the following sequence definitions, we assume a zero value outside the specified index ranges.

The stream *Network Sequence*, S_n, of length L_n specifies the amount of data, $S_\mathrm{n}(i)$, $1 \le i \le L_\mathrm{n}$, that the server must send to a particular client during round i of its playback. The *Buffer Sequence*, S_b, of length $L_\mathrm{b} = L_\mathrm{n} + 1$ defines the server buffer space, $S_\mathrm{b}(i)$, $0 \le i \le L_\mathrm{b} - 1$, required by the stream during round i. The *Disk Sequence* S_d of length $L_\mathrm{d} = L_\mathrm{n}$ defines the total amount of data, $S_\mathrm{d}(i)$, $0 \le i \le L_\mathrm{d} - 1$, that we retrieve 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* S_m of length L_d determines the amount of data $S_\mathrm{m}(i, k)$, $0 \le i \le L_\mathrm{d} - 1$, that we retrieve from disk k, $0 \le k \le D - 1$, in round i. We can easily derive it from the corresponding disk sequence S_d, according to the striping method that we use. In particular, with *Variable-Grain Striping* [2] the disk striping sequence is defined as follows:

$$S_\mathrm{m}^{v}(i, k) = (K^{v}(i) - K^{v}(i-1)) \cdot B_\mathrm{l},\ K^{v}(i) = \left\lceil \frac{\sum_{0 \le j \le i} S_\mathrm{d}(j)}{B_\mathrm{l}} \right\rceil,$$

Table 1
Summary of symbols used in the descriptions of our resource reservations schemes

Symbol	Description
D	Number of disks
L_n	Length of network sequence
$S_\mathrm{n}(i), 1 \le i \le L_\mathrm{n}$	Bytes sent at round *i*
L_b	Length of buffer sequence
$S_\mathrm{b}(i), 0 \le i \le L_\mathrm{b} - 1$	Buffer space at round *i*
L_d	Length of disk sequence
$S_\mathrm{d}(i), 0 \le i \le L_\mathrm{d} - 1$	Bytes read at round *i*
$S_\mathrm{m}(i, k), 0 \le i \le L_\mathrm{d} - 1, 0 \le k \le D - 1$	Striping sequence
B_p	Physical block size of disk
B_l	Logical block size of system
B_s	Stride size
T_fullSeek	Disk full-seek time
$T_\mathrm{trackSeek}$	Disk single-track seek time
T_avgRot	Average disk rotational latency
R_disk	Disk transfer capacity
R_net	Network transfer capacity
M_i	Active streams during *system* round *i*
$l^j, 1 \le j \le M_i$	Starting round of stream *j*
$T_\mathrm{disk}(i, k)$	Access time on disk *k* in round *i*
R_net	Server network link capacity
$T_\mathrm{net}^j(i)$	Transmit time of client *j* at round *i*
$B^j(i)$	Buffer space of client *j* at round *i*

when $i \pmod D = k$, and $S_\mathrm{m}^v(i, k) = 0$ when $i \pmod D \neq k$. Intuitively, the variable amount of data retrieved during a round for a client is always accessed from a single disk, and the disks are used round-robin in successive rounds. The disk striping sequence determines the particular single disk accessed and the exact amount of data retrieved during each round. In comparisons with alternative striping techniques, we have shown significant performance benefit for Variable-Grain Striping [2], which makes it the preferred method in the present study.

We assume that each disk has full-stroke (edge-to-edge) seek time T_fullSeek, single-track seek time $T_\mathrm{trackSeek}$, 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. Additionally, the C-SCAN disk scheduling policy limits the total distance travelled by the disk head per round to two full-stroke seeks. 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*, $1 \le j \le M_i$, started. Then, the total access time on disk *k* in round *i* of the system operation can be approximated by the expression:

$$T_\mathrm{disk}(i, k) = 2T_\mathrm{fullSeek} + 2M_i \cdot (T_\mathrm{trackSeek} + T_\mathrm{avgRot}) + \sum_{j=1}^{M_i} \frac{S_\mathrm{m}^j(i - l^j, k)}{R_\mathrm{disk}}, \tag{1}$$

where S_m^j is the disk striping sequence of client *j*. In Eq. (1), we include the factor 2 in the first and second terms due the C-SCAN policy and the stride-based allocation scheme, respectively. In the total time that we reserve for disk k during round i, for each additional client j we account extra access time:

$$T^j_{\text{disk}}(i, k) = 2 \cdot (T_{\text{trackSeek}} + T_{\text{avgRot}}) + \frac{S^j_{\text{m}}(i - l^j, k)}{R_{\text{disk}}} \tag{2}$$

when $S^j_{\text{m}}(i - l^j, k) > 0$. Reservations of network bandwidth and buffer space are simply based on the network and buffer sequence of each accepted playback request, respectively. For example, if R_{net} is the total network bandwidth available to the server, then the corresponding network transmission time reserved for client j in round i becomes $T^j_{\text{net}}(i) = S^j_n(i - l^j)/R_{\text{net}}$, where S_{n} is the network sequence of client j. Also, the total server memory buffer reserved for client j in round i becomes $B^j(i) = S^j_b(i - l^j)$, where S_{b} is the buffer sequence of client j.

4. Shared-buffer Smoothing

4.1. Outline

Previous studies have pointed out the potentially low disk utilization and system throughput achieved when retrieving variable bit-rate streams from disks, and the need for appropriately prefetching data into server buffers [19,26]. For a similar problem studied in the context of network links carrying variable bit-rate streams, it was proposed to smooth bitrate peaks along a stream by prefetching data into client buffers [10,29]. Such an approach can improve bandwidth utilization in network links (and disk channels as a side-effect), but is dependent on the memory configuration of individual clients.

Here, we consider smoothing the disk bandwidth peaks by prefetching stream data into server buffers. Note that the data access order remains sequential during normal playback, and prefetching only affects the amount of data accessed over time. 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 [19]. Existing client-based smoothing algorithms do not have this problem, due to their implicit assumption of a fixed buffer size available to each client. Unlike the case of prefetching data into the server buffer, the client buffer space is not multiplexed among different streams.

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 the individual rounds j up to $i - 1$, after prefetching block b, may not exceed the fraction of disk bandwidth required in round i. 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. Essentially, we try to balance the two resources and prevent one of them to be exhausted much earlier than the other as multiple streams are concurrently served. We apply offline these constraints to individual streams, and experimentally study their effect when serving multiple streams concurrently.

We should point out that merely replacing the buffer bounding heuristic that we use in our algorithm with other constraints can create an entire class of smoothing algorithms that can satisfy different optimization objectives under alternative operating conditions. However, in our present study, we only examine a specific buffer bounding constraint that we found to perform acceptably for our purposes, and leave for future investigation the comparative study of any alternatives.

Knowing the data amount that we need to retrieve from the disks during stream playback is important information that we may use during stream storage. As we already explained, variable-grain striping distributes data across a disk array so that we can retrieve from a single disk the amount of data required during the playback of a fixed time period. We have shown in previous related work the significant improvement in system throughput as a result of using such striping methods instead of the traditional approach of disk striping using fixed-size blocks [2]. On the other hand, a retrieval sequence that is fixed a priori ignores the exact resource tradeoffs that occur during system operation, when different stream playbacks are multiplexed. We evaluate in detail the resource utilizations that are achieved with our approach.

4.2. Basic definitions

Before explaining the algorithm, we have to introduce several definitions. Our main challenge is to capture with appropriate measures the two resources that we consider, namely disk transfer bandwidth and server buffer space. Subsequently, we use these measures to approximately balance the occupancy of these two resources. For that purpose, we use the utilization of the disk bandwidth and server buffer space incurred by the stream under consideration. We also need to specify the specific range of the stream sequence within which data prefetching takes place (Fig. 2).

The disk time reservation for each extra disk transfer of X bytes is approximated by Eq. (2) of Section 3.3. We repeat it here for reading convenience:

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

We point out that Eq. (2) is only part of the total disk access delay given by Eq. (1). We divide this estimated delay by the round duration T_{round}, and get the bandwidth utilization incurred by the input stream during a round.

Definition 1. Let the *disk time proportion* of X bytes, $P_{\mathrm{d}}(X)$, be the fraction of the round time T_{round} that the disk time reservation $T_{\mathrm{disk}}(X)$ occupies: $P_{\mathrm{d}}(X) = T_{\mathrm{disk}}(X)/T_{\mathrm{round}}$. Further, let the *buffer space proportion* of X bytes, $P_{\mathrm{b}}(X)$, be the fraction of the buffer space for each disk, B_{disk}, that X bytes occupy in a round: $P_{\mathrm{b}}(X) = X/B_{\mathrm{disk}}$. Then, the *maximum resource proportion* of round i, *MRP*(i), is the maximum of the corresponding disk time and buffer space proportions at round i: $\max(P_{\mathrm{d}}(S_{\mathrm{d}}(i)), P_{\mathrm{b}}(S_{\mathrm{b}}(i)))$.

Definition 2. The *deadline round* of a block is the latest round at which the block can be accessed from the disk without incurring a real-time violation of the corresponding network transfer. Then, with respect to a specific block, we call *candidate rounds* all the rounds before the deadline round, and *prefetch round*

```
input : $L_n$, $B_l$, $S_n[]$ ( $= 0$ $outside$ $[1..L_n]$ )
output : $L_d$, $S_d[]$, $L_b$, $S_b[]$
// initialize disk and buffer sequences from network sequence
$S_d[] = 0$, $S_b[] = 0$, $L_d = L_n$, $L_b = L_n + 1$, $curSum = 0$
for $i_{dln}$ : $0..L_n$
    $prvSum = curSum$, $curSum += S_n(i_{dln} + 1)$
    $S_d(i_{dln}) = B_l \cdot (\lceil curSum/B_l \rceil - \lceil prvSum/B_l \rceil)$
    $S_b(i_{dln})$ = $S_b(i_{dln} - 1)$ + $S_d(i_{dln})$ - $S_n(i_{dln} - 1)$
end-for
// visit each round in increasing order
for $i_{dln}$ : 0..$L_n$ - 1
    // skip rounds with MRP(buffer) >= MRP(disk)
    if ( $P_{buf}(S_b(i_{dln}))$ >= $P_{dsk}(S_d(i_{dln}))$ ) continue endif
    repeat // consider each block of the current round
        $i_{min} = i_{dln}$, $P_{min} = max( P_{dsk}(S_d(i)), P_{buf}(S_b(i_{dln})) )$
        for $i_{cnd}$ : $i_{dln} - 1 .. 0$   // examine candidates backwards
            // find MRP($i_{cnd}$) for $i_{cnd}$ as prefetch round
            $P_{prf} = max( P_{dsk}(S_d(i_{cnd}) + B_l), P_{buf}(S_b(i_{cnd}) + B_l) )$
            // find MRP($i_{cnd}$) for $i_{cnd}$ as shift round
            $P_{shf} = max( P_{dsk}(S_d(i_{cnd})), P_{buf}(S_b(i_{cnd}) + B_l) )$
            if ($P_{prf} < P_{min}$)     // if $i_{cnd}$ possible prefetch round
                $i_{min} = i_{cnd}$, $P_{min} = P_{prf}$
            else if ($P_{min} < P_{shf}$) // if $i_{cnd}$ cannot be shift round
                break
            end-if
        end-for
        if ($i_{min} < i_{dln}$)                          // on search success
            $S_d(i_{dln}) -= B_l$                        // update deadline round
            $S_d(i_{min}) += B_l$, $S_b(i_{min}) += B_l$ // update prefetch round
            for $i_{shf}$ : $i_{min} + 1 .. i_{dln} - 1$       // update shift rounds
                $S_b(i_{shf}) += B_l$
            end-for
        end-if
    until ($i_{min} >= i_{dln}$)  // no more prefetch rounds found
end-for
```

Fig. 2. The *Shared-buffer Smoothing* algorithm generates majorization minimal disk sequence with the disk bandwidth proportion limiting the server buffer proportion.

the one actually chosen for prefetching. All the rounds between the deadline and the prefetch round are called *shift rounds*.

4.3. The Algorithm

Main goal of the present study is to explore methods of making as uniform as possible over time the bitrate of stream data transferred through a disk channel. In order to mathematically formalize this objective, we use as "smoothness" criterion one that is based on *Majorization Theory* [21,29]. 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 *majorized* by $\mathbf{y}$, $\mathbf{x} \prec \mathbf{y}$, if there is k, $1 \leq k \leq n$, such that

$$\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i,$$

$$x_{[i]} = y_{[i]}, \quad \text{for} 1 \leq i < k,$$

and

$$x_{[k]} < y_{[k]}.$$

Then, we consider $\mathbf{x}$ smoother than $\mathbf{y}$, if $\mathbf{x} \prec \mathbf{y}$, and 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 $\mathbf{z} \prec \mathbf{x}$.

We receive as input the logical block size B_l, and a stream network sequence S_n of data amounts needed every time unit of playback. The algorithm runs offline and produces as output (i) the smoothed disk sequence S_d of data amounts transferred into the server buffer from the disk over time, and (ii) the corresponding buffer sequence S_b. 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 that specifies the amount of data retrieved from (after being stored across) the disks every time unit of playback.

Initially, the algorithm quantizes the disk transfer and buffer sizes into multiples of the logical block size B_l, but keeps the network transfers expressed in bytes. We visit the consecutive deadline rounds of the generated sequences in increasing order starting from round zero. For every logical block to be fetched from the disk, we examine the preceding rounds in decreasing order towards round zero for potential block prefetching.

We are striving to reduce the MRP of the current round by retrieving one or more of its blocks earlier at a prefetch round with lower MRP. The search for prefetch round succeeds when we can reduce the MRP of the current round but keep it higher than or equal to the MRP of the shift and prefetch rounds. Among the candidate rounds that satisfy this requirement we choose the one with the lowest MRP that is closest to the current round.

At the prefetch round we retrieve the data block from the disk into the server memory. At the shift rounds we keep the fetched data in memory. When the deadline round arrives, we transmit the specified data block from the server memory to the client buffers. Thus, we leave the client unaware of the fact that we prefetched into the server memory the transmitted data. We repeat the search for prefetch round with each block of the current round as long as the search succeeds. When the search fails, we restart the search procedure at the subsequent deadline round. Below, we show that the algorithm works correctly.

Lemma 1. *The Shared-buffer Smoothing algorithm chooses as prefetch round the candidate that satisfies the following properties in order of decreasing priority*:

1. ***Real-Time***: *Satisfy the network transfer deadlines.*
2. ***Resource-Balance***: *Prevent the MRP of the prefetch and shift rounds from exceeding the MRP of the deadline round.*
3. ***Min-Smooth***: *Pick the candidate with the lowest MRP as prefetch round.*

Proof. The Real-Time property is satisfied at line 10 by choosing the prefetch round among the candidate rounds that precede the deadline round. We preserve the Resource-Balance property at lines 20 and 22, respectively, where we explicitly keep the MRP of the prefetch and shift rounds no more than that of the deadline round. We use the variable $P_{\min}$ at lines 20–21 to keep track of the candidate round with minimal MRP. We break ties in the choice of the prefetch round at lines 15 and 20 by searching in a decreasing loop with strict inequality. □

The Real-Time property maintains valid the playback of the generated disk transfer sequence. The Resource-Balance property keeps the buffer space proportion bounded by the disk time proportion of the deadline round. The Min-Smooth property with the tie-breaking rule leads to a disk sequence that is smoother than the original, and only increases the buffer occupancy if necessary.

Definition 3. 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*.[1]

Lemma 2 (Muirhead [23]). *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 [21, p. 135]. □

In the presentation that follows *transfer* units correspond to logical blocks of size B_{l} as opposed to individual bytes.

Lemma 3. *The Shared-buffer Smoothing algorithm produces disk sequence with no additional transfer to satisfy the properties of Lemma 1.*

Proof. 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 $L_{\mathrm{n}} = 1$, with round 0 to access the data from disk and round 1 to send the data over the network. We assume that at $L_{\mathrm{n}} = k$ the claim is valid. We show that it is also valid for $L_{\mathrm{n}} = k + 1$. Let us assume that we get the sequence of the k first disk accesses $0, \ldots, k - 1$ to satisfy the lemma claim, before starting to deal with the disk access of round k. Due to the Real-Time property of Lemma 1, it is not possible to schedule at round k block accesses from the previous rounds. Therefore, the only acceptable block *transfer* is moving blocks of the round k to previous rounds. An exhaustive search is done in the for-loop of the algorithm at lines 15–25. We visit each of the preceding rounds, and keep record of the latest round where we can prefetch a block with minimal disk time proportion. We repeat the above search using the repeat-loop of lines 13–33 until no more *transfers* of logical blocks belonging to round k are possible. The first time that the search for prefetch round fails at round k, we know there are no more Lemma 1 *transfers* possible starting from either round k or rounds $1, \ldots, k - 1$. We conclude so because each earlier Lemma 1 *transfer* moved a block to

[1] The term *transfer* that we borrow from the original definition [23], should not be confused with regular data transfers.

prefetch round with lowest MRP, and only increased the buffer proportion of the related prefetch and shift rounds. □

Theorem 1. *The Shared-buffer Smoothing algorithm uses Lemma 1 transfers to generate a majorization-minimal disk sequence.*

Proof. From Lemma 3, the disk sequence S_d generated by our algorithm has no additional Lemma 1 *transfer*. Then, from Lemma 2, there is no sequence S'_d majorized by S_d. If such a sequence existed, additional block *transfers* would be possible after the termination of our algorithm. □

As a corollary of the above theorem, the Shared-buffer Smoothing algorithm generates disk sequence that satisfies the data transfer deadlines of the original network sequence, and prevents the buffer space proportion from exceeding the maximum disk time proportion.

4.4. Analysis

For several reasons, previous client-based smoothing algorithms are inadequate for solving the Shared-buffer 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 assume are (a) smoothing out the required disk transfer bandwidth, while at the same time (b) keeping under control the occupied buffer space. These constraints span resources of different types and measures (access delays in disks and occupied buffer space in memory), and are difficult to describe using data amounts only. This problem is much simpler 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.

As a result of the above, 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. The computational complexity of our algorithm is $\mathrm{O}((\sum_{i=1}^{L_\mathrm{n}} S_\mathrm{n}(i)/B_\mathrm{l})L_\mathrm{n})$, where S_n is the input network sequence and L_n is its length in rounds. This complexity follows from the fact that we visit once each of the $\mathrm{O}((\sum_{i=1}^{L_\mathrm{n}} S_\mathrm{n}(i))/B_\mathrm{l})$ blocks, and consider for each block at most $\mathrm{O}(L_\mathrm{n})$ alternative prefetch rounds. It may be possible to reduce this complexity, although practically the execution of the algorithm completes in tens of seconds. We ran our experiments using 30-min video streams as input on a 133 MHz RISC processor with $B_\mathrm{l} = 16\,\mathrm{kB}$, $L_\mathrm{n} = 1,800$ and $\sum_{i=1}^{L_\mathrm{n}} S_\mathrm{n}(i) = 1.12E9$. Since we generate the smoothed schedule off-line, we find the above execution time acceptable. As we already pointed out in Section 4.1, our smoothing algorithm can generate majorization-minimal disk sequences with alternative buffer bounding constraints. One specific case is the fixed buffer size typically assumed in network smoothing algorithms [29]. The higher computational complexity of our algorithm relative to $\mathrm{O}(L_\mathrm{n})$ of existing algorithms

[29] is the extra cost that we pay to avoid the fixed buffer constraint typically “hardwired” in network smoothing.

5. Experimentation environment

5.1. Prototype overview

We have designed and built a media server experimentation platform, in order to evaluate the resource requirements of alternative stream scheduling techniques. Our code communicates with either the DiskSim package [11] for simulated disk access time measurements (with advanced features of modern disks such as on-disk cache and zones), or hardware disks through their raw interface [3]. 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 does not transfer stream data. In *Simulated Disk* mode, the system processes disk requests using a DiskSim [11] disk array. Alternatively, the system can access hardware disks and transfer data to client network destinations. For the experiments in the current study, we mostly used the Admission Control mode, except for the validation in Section 8 where we used the Simulated Disk mode.

5.2. 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 are the bottleneck resource, we consider a hypothetical system that accesses disk data with zero seek and rotational delays. Then the stream completion rate becomes equal to the maximum arrival rate of playback requests $\lambda = \lambda_{\max}$. This rate creates enough system load to show the performance benefit of arbitrarily efficient data striping policies. Then, the mean stream completion rate μ for streams of average total size A_s bytes becomes

$$\mu = \frac{D \cdot R_{\text{disk}} \cdot T_{\text{round}}}{A_S} \text{streams/round}. \tag{3}$$

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

When a playback request arrives, the admission control process checks whether available resources exist for every round of the playback, namely disk transfer time, network transfer time and buffer space in the system. A critical question is how long a request can be delayed to start after it arrives into the system. Normally, a client prefers that a request is served immediately, although this may be too expensive in terms of resource requirements for a target system throughput. Alternatively a playback request can be delayed for a limited number of rounds, until available resources become available for the playback to start and continue uninterrupted. In our experiments, if we cannot initiate a request in the next round, we repeat the admission test at each round up to $\lceil 1/\lambda \rceil$ rounds into the future. Previously [2], we verified experimentally that checking $\lceil 1/\lambda \rceil$ rounds into the future achieves most of the potential system capacity. If we are unable to accept the request in the consid-

Table 2
We used six MPEG-2 video streams of 30 min duration each

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

The coefficient of variation shown in the last column changes according to the content type.

Table 3
Features of the Seagate SCSI disk assumed in our experiments

Seagate Cheetah ST-34501N	
Data bytes per drive	4.55 GB
Average sectors per track	170
Data cylinders	6526
Data surfaces	8
Zones	7
Buffer size	0.5 MB
Track to track seek(read/write)	0.98/1.24 ms
Maximum seek(read/write)	18.2/19.2 ms
Average rotational latency	2.99 ms
Internal transfer rate	
Inner zone to outer zone burst	122–177 Mbit/s
Inner zone to outer zone sustained	11.3–16.8 MB/s

ered time window, we reject the request instead of keeping it in a queue for further reconsideration later.

Our basic performance metric is 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.

5.3. Experimentation setup

We used six different VBR MPEG-2 clips with the statistical characteristics shown in Table 2. Each of them lasts 30 min, and consists of 54,000 frames with resolution 720 × 480 pixels, color depth 24 bits, playback frequency 30 frames/s, and group structure $IB^2PB^2PB^2PB^2PB^2$. The encoding hardware that we used generates bits at rates in the range between 1 Mbit/s and 9.6 Mbit/s. Depending on the clip, the bitrate coefficient of variation takes values in the range between 0.028 and 0.383. In the *mixed* benchmark, we distribute the playback requests uniformly across the six different streams. Additionally, we show experimental results from serving each individual stream type in the system.

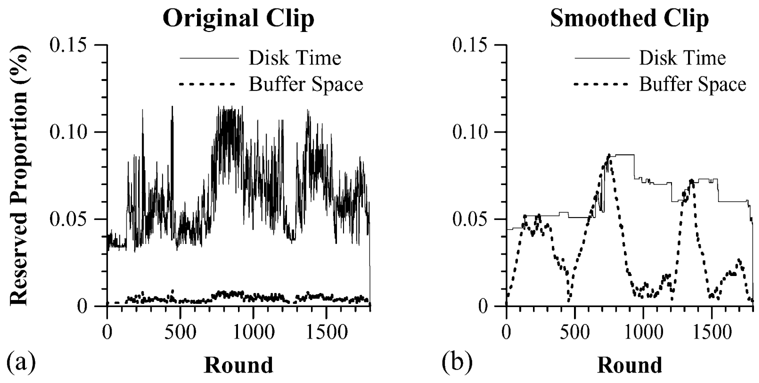

Fig. 3. We depict the disk time and buffer space proportion in the system (a) before, and (b) after applying *Shared-buffer Smoothing*. The peak of disk time proportion drops from 11.5% to 8.7%, while the peak of buffer space proportion increases from less than 1% to 8.7%. We assume the disk parameters of Seagate Cheetah and server buffer of 256 MB per disk.

For experimentation with homogeneous disks, we assumed Seagate Cheetah SCSI disks with the features shown in Table 3.[2] 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. We set the logical block size B_l equal to 16 KB bytes, the physical sector size B_p to 512 bytes, and the stride size B_s of the disk space allocation to 2 MB. We organized the server memory in buffers of fixed size $B_l = 16$ KB bytes each, with total space of 256 MB for every extra disk. (We examine later the effect of buffer space to the system performance.) We assume infinite network bandwidth, thus leaving the contention for network resources outside the scope of the current work.

In our experiments, we set the round length equal to one second. We found this length to achieve most of the system capacity with reasonable initiation latency. This choice also facilitates comparison with previous work where rounds of one second were used. We used a warmup period of 3000 rounds and calculated the average number of active streams from round 3000 to round 9000. We repeat the measurements until the half-length of the 95% confidence interval lies within 5% of the estimated mean value of the number of active streams.

6. Study of homogeneous disks

We start with a study of disk arrays consisting of functionally equivalent disks. In Fig. 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 *Shared-buffer Smoothing* (Fig. 3(b)), data are prefetched into the server buffer according to the resource-balance constraint. This keeps the maximum buffer space proportion to be no more than the maximum disk time proportion (8.7% in this example).

In Fig. 4 we can see the sustained number of active streams at different system loads and disk array sizes. In all the cases, we stripe the stream data using the Variable-Grain Striping method. The smoothed plots

[2] 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 [15].

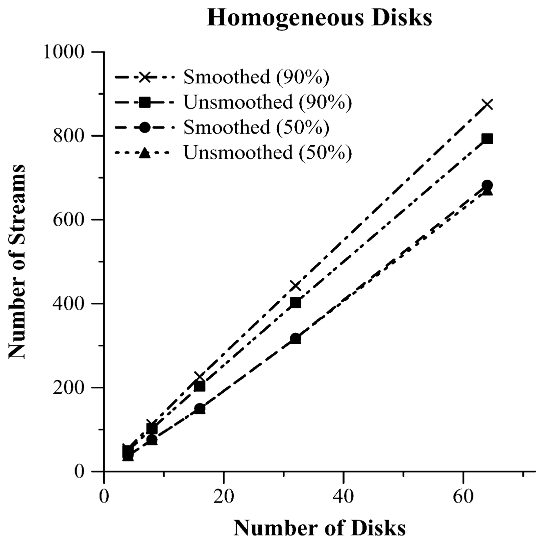

Fig. 4. Using mixed workload, the sustained number of active streams increases almost linearly with the number of disks. Although all the submitted streams are accepted at system load 50%, at system load 90% *Shared-buffer Smoothing* increases the number of active streams by about 10%. This benefit is maintained across different numbers of disks. Note that the lines of unsmoothed and smoothed streams overlap at load 50%.

show the performance benefit of applying our algorithm assuming 256 MB of available server buffer space for each extra disk. At moderate load of $\rho = 50\%$, Variable-Grain Striping with no smoothing allows all stream requests to be accepted. At a higher load of $\rho = 90\%$ the *Shared-buffer Smoothing* can improve throughput by over 10%. The corresponding rejection rate (not shown) at 90% load is 25% with Shared-buffer Smoothing, and 41% with plain Variable-Grain Striping.

We show results for individual stream types in Fig. 5. We find that the benefit of *Shared-buffer Smoothing* depends on the variability of data transfers across different rounds. Thus, although smoothing adds no benefit to streams with negligible variability (e.g. Documentary), higher variation leads to benefit of 15% more streams (Action). Fig. 6 shows the average reserved busy time $T_{\text{disk}}(i, k)$ expressed as percentage of round time on a disk. For most stream types the average disk time hardly exceeds 80% of the round time with plain Variable-Grain Striping. However, it consistently approaches 90% and in several cases exceeds 93% (Action, Music Clip, Talk Show) when applying *Shared-buffer Smoothing*. This is remarkably high when compared to the 96% achieved by Documentary and demonstrates very low variation of data transfer sizes across different rounds.

The statistics that we gathered across the different stream workloads show that the average occupied proportion of the available buffer space is about 50%, and the maximum hardly exceeds 60% at 90% load. Although in individually smoothed streams the buffer space proportion is allowed to reach that of disk bandwidth, 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 satisfied. Further increase of the aggregate buffer demand without the buffer becoming a potential bottleneck in the system, would require incorporating into the algorithm information about the playback request multiplexing.

In our experiments until now, we have assumed a server buffer of 256 MB per disk. From Fig. 7 we can conclude that more than half of the *Shared-buffer Smoothing* benefit is achieved with server buffer size as low as 64 MB per disk. Nevertheless, our previous assumption of 256 MB server memory per disk is justified by the fact that the advantage from extra server memory is obtained at purchase and administration cost that is only fraction of that required for additional high-end disk

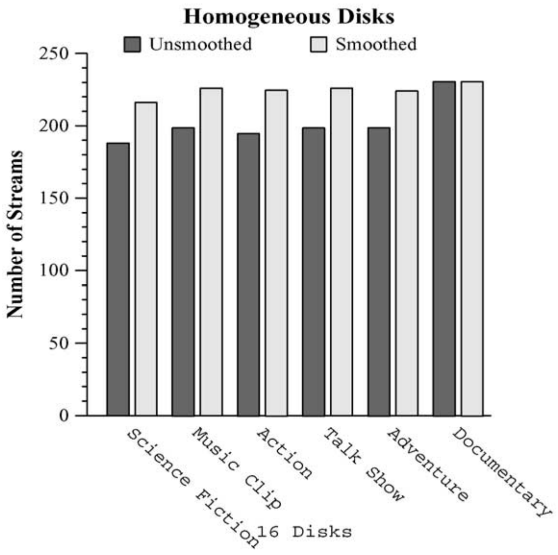

Fig. 5. The advantage of *Shared-buffer Smoothing* when combined with Variable-Grain Striping can exceed 15% (Action) depending on the stream type. We set the load to 90% on 16 disks, and assumed buffer space 256 MB per disk in the system.

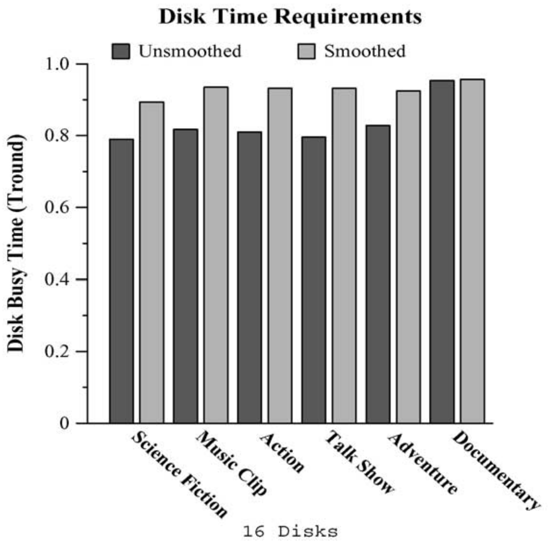

Fig. 6. With 16 disks and 90% system load, the average disk time reserved each round increases from less than 80% to over 90% with *Shared-buffer Smoothing* and server buffer 256 MB per disk. The disk time, that we show for only one of the disks, was similar (typically within 2%) across the different disks of the array.

drives.

7. Study of heterogeneous disks

System designers and administrators traditionally assume that disk arrays consist of homogeneous disks, presumably in order to keep the system complexity manageable. However, the demonstrated scalability of stream striping [6,2] makes also interesting to investigate systems with different disk types that are potentially upgraded incrementally [9]. 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

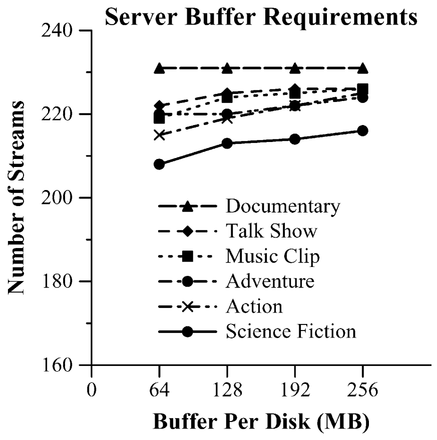

Fig. 7. With 64MB buffer space per disk, more than half of the total benefit of *Shared-buffer Smoothing* can be achieved (see also Fig. 5). Increasing the buffer space to 256 MB further improves the number of streams in Science Fiction and Action types although at a diminishing degree.

Table 4
Features of the HP SCSI model that is included in the experiments for heterogeneous disk arrays

HP-C3323A	
Data bytes per drive	1,052,491,776
Data sectors per track	72–120
Data cylinders	2910
Data surfaces	7
Zones	8
Buffer size	0.5 MB
Track to track seek	<2.5 ms
Maximum seek	22 ms
Rotational latency	5.56 ms ± 0.5%
Internal transfer rate	
Inner to outer zone burst	4.0–6.6 MB/s
Inner to outer zone sustained	2.8–4.7 MB/s

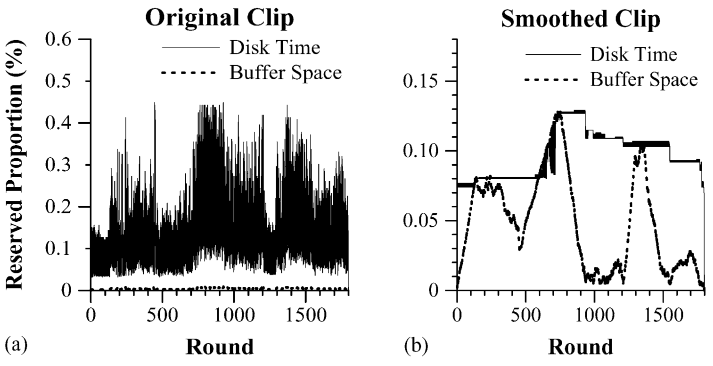

Fig. 8. Example of stream stored in an array consisting of Seagate and HP disks in alternating order. The low bandwidth of the HP model leads to disks busy over 40% during some rounds. When Shared-buffer Smoothing is applied, disk transfers are adjusted to smooth out the peaks and keep the maximum reservation below 13%. At the same time, the peak server buffer proportion is constrained to never exceed the peak disk time proportion.

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 [12].

In our experiments, we assume disk arrays consisting of Seagate (Table 3) and older HP disks in alternating order. The features of the HP disks are shown in Table 4. 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.

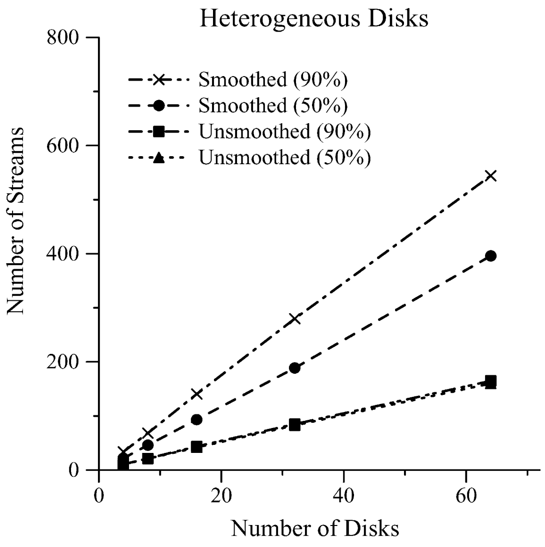

Fig. 9. Using the mixed workload, the sustained number of active streams remains the same as the load is raised from 50 to 90% with plain Variable-Grain Striping. In comparison, with *Shared-buffer Smoothing* the number of streams increases by a factor of 2 at 50% and more than a factor of 3 at 90% load. We assume server buffer space of 256 MB per disk.

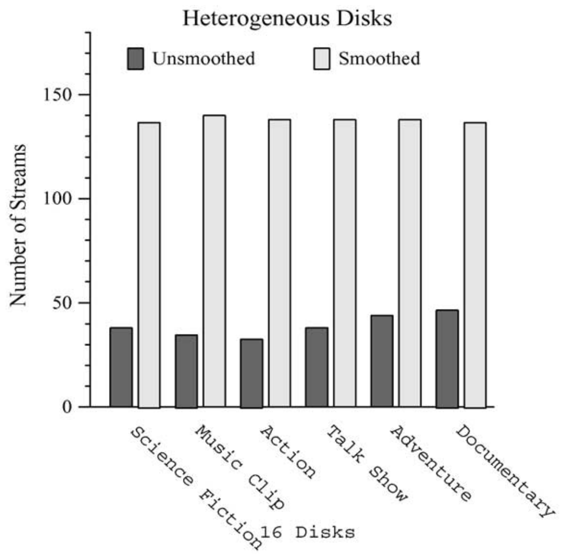

Fig. 10. Applying *Shared-buffer Smoothing* to different stream types, can lead to an increase in the number of accepted streams by more than a factor of 3. We assume load equal to 90%, and server buffer 256 MB per disk.

In Fig. 8(a) we depict an example of a stream striped across an heterogeneous disk array. The lower transfer rate of the HP disks creates peaks of disk time proportion that can exceed 40%. Essentially, heterogeneous disks introduce variation in the disk bandwidth that is available to a striped stream over time. In order to take advantage of all the bandwidth capacity in the system, we extend the *Shared-buffer*

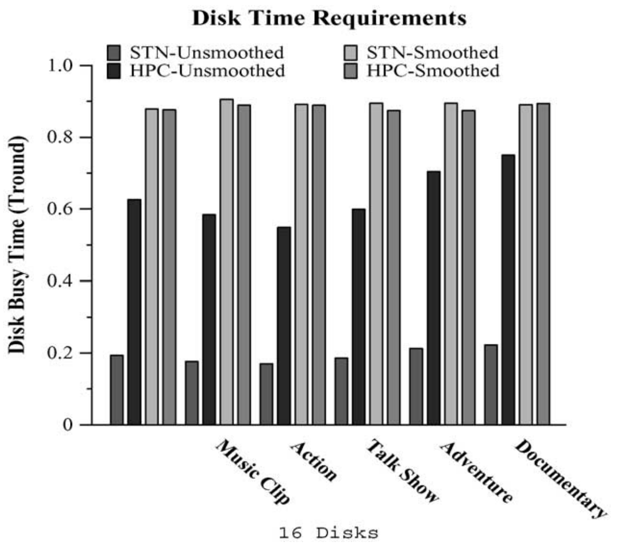

Fig. 11. The two leftmost bars of each stream show the average reserved disk time for the Seagate (STN) and HP (HPC) disks, assuming plain Variable-Grain Striping and 90% load. The lower transfer bandwidth of the HP disk creates a bottleneck keeping the reserved disk time of the Seagate disk to less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

Smoothing algorithm with knowledge about the available disk bandwidth during each round. For that purpose, we redefine the disk time $T_{\text{disk}}(X)$ and the disk time proportion function $P_{\text{d}}(X)$ to accept a second argument k that specifies the disk type used in each round: $P_{\text{d}}(X, k) = T_{\text{disk}}(X, k)/T_{\text{round}}$. During the operation of the *Shared-buffer Smoothing* algorithm, we derive the disk type k of each round i using a simple rule, such as $k = i \pmod D$, where D is the total number of the disks. Subsequently, we incorporate the average disk transfer capacity into the expression of the request interarrival time. If R^k_{disk} is the minimum internal transfer rate of disk k, then the service rate definition of Eq. (3) becomes: $\mu = (T_{\text{round}} \cdot \sum_{k=0}^{D-1} R^k_{\text{disk}})/\sum_{i=1}^{L_{\text{n}}} S_{\text{n}}(i)\, streams/round$.

We applied the extended *Shared-buffer Smoothing* algorithm to the stream example of Fig. 8(a). The generated transfer sequence, shown in Fig. 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 Fig. 9, we compare the performance of unsmoothed streams with that of smoothed streams in a range of heterogeneous disks between 4 and 64. Although the number of streams always increases linearly with the number of disks, *Shared-buffer Smoothing* can achieve an advantage that exceeds a factor of 2 and 3 at loads of 50 and 90%, respectively. The reason is that only a small number of streams is sufficient to saturate the limited disk bandwidth of the HP disks. As a result, the Seagate disks are prevented from attaining high bandwidth utilization without appropriate adjustment of the disk transfers. A similar behavior is also demonstrated across different stream types in Fig. 10. With plain Variable-Grain Striping, the number of supported streams on 16 disks hardly exceeds 50; when Shared-buffer Smoothing is added the number of streams gets close to 140.

In Fig. 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 the first and second disks only, since the statistics for the rest of the disks were similar. As we see, the average busy time of the Seagate disks remains less than 25% of the round time with plain Variable-Grain Striping. The reason is the saturation of the HP disks, whose corresponding busy time varies between 50 and 80% (it is less than 100% due to the relatively high

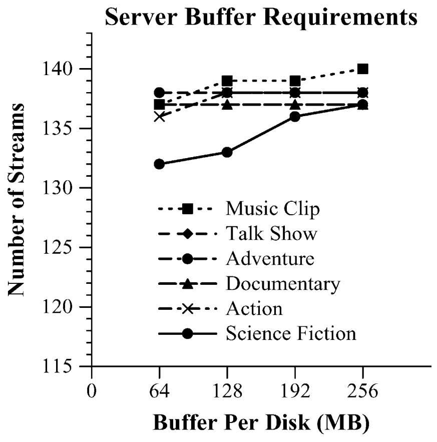

Fig. 12. When we set the server buffer per disk to 64 MB, we obtain most of the benefit of *Shared-buffer Smoothing* (see also Fig. 10). Scaling the server buffer from 64 to 256 MB increases only marginally the sustained number of active streams.

bitrate of the individual streams). When Shared-buffer Smoothing is applied, a high average reserved disk time close to 90% is achieved across all the disks of the disk array. This becomes possible with 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 Fig. 12, having only 64 MB per disk is sufficient to get most (close to 95%) of the benefits of Shared-buffer Smoothing for the particular streams included in our benchmark.

8. System validation

In order to keep the computation time reasonable, the previous experiments were conducted with our system in Admission Control mode. This means that we make all the necessary resource reservations when playback requests arrive into the system, but we omit the actual data transfers that normally take place between the disks and the network. Although this mode adds flexibility in specifying the number of the disks in the system, it leaves open the question of whether low-level resource management issues are handled correctly by the system.

In previous work, we verified the correct operation of the system in small scale using hardware disks [3]. In the present section, we additionally use the Simulated Disk Mode to compare the statistics of the disk time reservations against those of the access times of the individual data transfers involved, using the DiskSim representation of the Seagate Cheetah and HP C3323A disks [11]. For that purpose, we use a four-disk array model consisting of the two disk types in alternating order. We presume that each disk is attached to a separate SCSI bus 20 MB/s, and there is no contention on the host system bus connecting

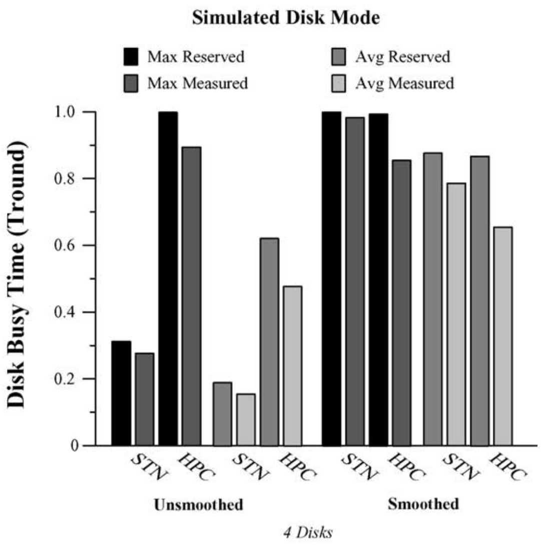

Fig. 13. We create an array of four disks with Seagate (STN) and HP (HPC) models in alternating order. We show the average and maximum reserved and measured time for the first and second disk of the array with the mixed workload at 90% load. For the STN disks, the reserved statistics are no more than 8% higher than the measured. For the HPC disk, the corresponding difference can get up to 20%. We made the measurements using the detailed disk simulation models by Ganger et al.

the SCSI buses. The statistics are gathered during 6000 rounds after a warmup period of 3000 rounds, as before and the mixed workload is used. With the load set to 90%, the server can support 9.74 active streams with plain Variable-Grain Striping and 33.87 active streams with smoothing.

As can be seen from Fig. 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. [11]. 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 plain striping and 20% with *Shared-buffer 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 fewer head movements are required with the smaller data transfers in the 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 be probably necessary to make use of extra disk geometry information that is not readily available for commercial disk drives [34].

9. Conclusions

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

In this paper, we introduce the *Shared-buffer 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 *Shared-buffer 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 *Shared-buffer 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 *Shared-buffer 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 study.

Comparative study of alternative buffer bounding constraints is a useful topic for future work. An interesting theoretical question would be to generalize our algorithm to smooth functions of multiple resources instead of individual resources only. Another open issue to explore further is the design of appropriate replication techniques for tolerating disk failures in heterogeneous disk environments.

References

[1] S. Anastasiadis, P. Varman, J.S. Vitter, K. Yi, Lexicographically optimal smoothing for broadband traffic multiplexing, in: ACM Symposium on Principles of Distributed Computing, Monterey, CA, July 2002, pp. 68–77.

[2] S.V. Anastasiadis, K.C. Sevcik, M. Stumm, Disk striping scalability in the exedra media server, in: ACM/SPIE Multimedia Computing and Networking Conf., San Jose, CA, January 2001, pp. 175–189.

[3] S.V. Anastasiadis, K.C. Sevcik, M. Stumm, Modular and efficient resource management in the exedra media server, in: USENIX Symposium on Internet Technologies and Systems, San Francisco, CA, March 2001, pp. 25–36.
[4] S.V. Anastasiadis, K.C. Sevcik, M. Stumm, Server-based smoothing of variable bit-rate streams, in: ACM International Conference on Multimedia, Ottawa, ON, October 2001, pp. 147–158.
[5] E.W. Biersack, M. Hamdi, Cost-optimal data retrieval for video servers with variable bit rate video streams, in: International Workshop on Network and Operating System Support for Digital Audio and Video, Cambridge, UK, July 1998, pp. 295–302.
[6] W.J. Bolosky, J.S. Barrera, R.P. Draves, R.P. Fitzgerald, G.A. Gibson, M.B. Jones, S.P. Levi, N.P. Myhrvold, R.F. Rashid, The tiger video fileserver, in: International Workshop on Network and Operating System Support for Digital Audio and Video, Zushi, Japan, April 1996, pp. 97–104.
[7] C.F. Chou, L. Golubchik, J.C.S. Lui, A performance study of dynamic replication techniques in continuous media servers, in: ACM SIGMETRICS, Atlanta, GA, May 1999, pp. 202–203.
[8] A. Dan, D. Sitaram, An online video placement policy based on bandwidth to space ratio (BSR), in: ACM SIGMOD, San Jose, CA, May 1995, pp. 376–385.
[9] T.E. Denehy, A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, Bridging the information gap in storage protocol stacks, in: USENIX Annual Technical Conference, Monterey, CA, June 2002, pp. 177–190.
[10] W.-C. Feng, J. Rexford, A comparison of bandwidth smoothing techniques for the transmission of prerecorded compressed video, in: IEEE INFOCOM, Kobe, Japan, April 1997, pp. 58–66.
[11] G.R. Ganger, B.L. Worthington, Y.N. Patt, The DiskSim Simulation Environment: Version 2.0 Reference Manual. Tech. Re CSE-TR-358–98, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, December 1999.
[12] J. Gray, P. Shenoy, Rules of thumb in data engineering, in: IEEE International Conference on Data Engineering, San Diego, CA, February 2000, pp. 3–10.
[13] S. Gringeri, K. Shuaib, R. Egorov, A. Lewis, B. Khasnabish, B. Basch, Traffic shaping, bandwidth allocation, and quality assessment for mpeg video distribution over broadband networks, IEEE Netw. 6 (1998) 94–107.
[14] D.T. Hoang, J.S. Vitter, Efficient Algorithms for MPEG Video Compression, Wiley, New York, 2002.
[15] The IBM Dictionary of Computing, McGraw-Hill, New York, 1994.
[16] I.-H. Kim, J.-W. Kim, S.-W. Lee, K.-D. Chung, VBR video data scheduling using window-based prefetching, in: IEEE Multimedia Computing and Systems, Florence, Italy, June 1999, pp. 159–164.
[17] T.V. Lakshman, A. Ortega, A.R. Reibman, VBR video: tradeoffs and potentials, in: Proceedings of the IEEE 86, 5 May 1998, pp. 952–973.
[18] D.-Y. Lee, H.Y. Yeom, Tip prefetching: dealing with the bit rate variability of video streams, in: IEEE Multimedia Computing and Systems, Florence, Italy, June 1999, pp. 352–356.
[19] D. Makaroff, N. Hutchinson, G. Neufeld, An evaluation of VBR disk admission algorithms for continuous media file servers, in: ACM Multimedia, Seattle, WA, May 1997, pp. 143–154.
[20] Y. Mansour, B. Patt-Shamir, O. Lapid, Optimal smoothing schedules for real-time streams, in: ACM Principles of Distributed Computing, Portland, OR, July 2000.
[21] A.W. Marshall, I. Olkin, Inequalities: Theory of Majorization and its Applications, Academic Press, New York, 1979.
[22] J. McManus, K. Ross, A dynamic programming methodology for managing prerecorded vbr sources in packet-switched networks, Telecommun. Syst. 9 (1998) 223–247.
[23] R.F. Muirhead, Some methods applicable to identities and inequalities of symmetric algebraic functions of *n* letters, in: Proceedings of the Edinburgh Mathematical Society, vol. 21, 1903, pp. 144–157.
[24] S. Paek, S.-F. Chang, Video server retrieval scheduling for variable bit rate scalable video, in: IEEE Multimedia Computing and Systems, Hiroshima, Japan, June 1996, pp. 108–112.
[25] R.H. Patterson, G.A. Gibson, E. Ginting, D. Stodolsky, J. Zelenka, Informed prefetching and caching, in: ACM Symp. Operating Systems Principles, Copper Mountain Resort, CO, December 1995, pp. 79–95.
[26] A.L.N. Reddy, R. Wijayaratne, Techniques for improving the throughput of VBR streams, in: ACM/SPIE Multimedia Computing and Networking, San Jose, CA, January 1999, pp. 216–227.
[27] J. Rexford, S. Sen, J. Dey, W. Feng, J. Kurose, J. Stankovic, D. Towsley, Online smoothing of live, variable-bit-rate video, IEEE Trans. Multimedia 2 (1) (2000) 37–48.
[28] S. Sahu, Z.-L. Zhang, J. Kurose, D. Towsley, On the efficient retrieval of vbr video in a multimedia server, in: IEEE Multimedia Computing and Systems, Ottawa, Canada, June 1997, pp. 46–53.

[29] J.D. Salehi, Z.-L. Zhang, J.F. Kurose, D. Towsley, 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.
[30] J.R. Santos, R. Muntz, Performance analysis of the RIO multimedia storage system with heterogeneous disk configurations, in: ACM Multimedia, Bristol, UK, September 1998, pp. 303–308.
[31] S. Sen, J. Rexford, D. Towsley, Proxy prefix caching for multimedia streams, in: IEEE INFOCOM, New York, March 1999, pp. 1310–1319.
[32] P.J. Shenoy, H.M. Vin, Efficient striping techniques for multimedia file servers, in: International Workshop in Network and Operating Systems Support for Audio and Video, St. Louis, MO, May 1997, pp. 25–36. An expanded version appears in Perform. Evaluation 38 (1999) 175–199.
[33] Y. Wang, D.H. Du, Weighted striping in multimedia servers, in: IEEE Multimedia Computing and Systems, Ottawa, Canada, June 1997, pp. 102–109.
[34] B.L. Worthington, G.R. Ganger, Y.N. Patt, J. Wilkes, On-line extraction of SCSI disk drive parameters, in: ACM SIGMETRICS, Ottawa, Canada, May 1995, pp. 146–156.
[35] W. Zhao, S.K. Tripathi, Bandwidth-efficient continuous media streaming through optimal multiplexing, in: ACM SIGMETRICS, Atlanta, GA, June 1999, pp. 13–22.
[36] R. Zimmermann, S. Ghandeharizadeh, Continuous display using heterogeneous disk-subsystems, in: ACM Multimedia, Seattle, WA, November 1997, pp. 227–328.

Stergios V. Anastasiadis received BSc (1994) degree from the Department of Computer Engineering and Informatics, University of Patras, Greece, and MSc (1996) and PhD (2001) degrees from the Department of Computer Science, University of Toronto. Subsequently, he joined the Deparment of Computer Science, Duke University as visiting assistant professor. His research interests include performance evaluation of computer systems, design and implementation of systems software architectures, and development of algorithms for data caching and prefetching problems.

Kenneth C. Sevcik is a professor of computer science with a cross-appointment in Electrical and Computer Engineering at the University of Toronto. He is Director of the Computer Systems Research Institute, and past Chairman of the Department of Computer Science. He holds degrees from Stanford (BS, mathematics, 1966) and the University of Chicago (PhD, information science, 1971). His primary area of research interest is in developing techniques and tools for performance evaluation, and applying them in such contexts as distributed systems, database systems, local area networks, and parallel computer architectures.

Michael Stumm received a diploma in mathematics and a PhD in computer science from the University of Zurich in 1980 and 1984, respectively. He is a professor in the Department of Electrical and Computer Engineering and the Department of Computer Science at the University of Toronto, Toronto, Canada. Dr. Stumm's research interests are in the areas of computer systems, in particular, operating systems for multiprocessors.