Scalable and Fault-Tolerant Support for Variable Bit-Rate Data in the Exedra Streaming Server

STERGIOS V. ANASTASIADIS
University of Ioannina, Greece
and
KENNETH C. SEVCIK and MICHAEL STUMM
University of Toronto, Canada

We describe the design and implementation of the Exedra continuous media server, and experimentally evaluate alternative resource management policies using a prototype system that we built. Exedra has been designed to provide scalable and efficient support for variable bit-rate media streams whose compression efficiency leads to reduced storage space and bandwidth requirements in comparison to constant bit-rate streams of equivalent quality. We examine alternative disk striping policies, and quantify the benefits of innovative techniques for storage space allocation, buffer management, and resource reservation, which we developed to achieve both predictability and high-performance in handling disk and network data transfers of variable size. Additionally, we investigate the differences between diverse data replication schemes over disk arrays, and compare methods for disk access time reservation that enable tolerance of disk failures at minimal cost. Overall, we demonstrate the feasibility of building network media servers that exploit the latest advances in media compression technology towards reducing the cost of wide-scale streaming services for stored data.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: measurement techniques—*Fault tolerance*; C.5.5 [**Computer System Implementation**]: Servers; D.2.11 [**Software Engineering**]: Software Architectures—*domain-specific architectures*; D.4.2 [**Operating Systems**]: Storage Management—*secondary storage*; D.4.5 [**Operating Systems**]: Reliability—*fault-tolerance*; D.4.8 [**Operating Systems**]: Performance—*measurements*; E.5 [**Files**]: organization/structure

General Terms: Design, Experimentation, Measurement, Performance, Reliability

Additional Key Words and Phrases: Content distribution, multimedia compression

Authors' addresses: S. V. Anastasiadis, Department of Computer Science, University of Ioannina, P. O. Box 1186, GR 45110 Ioannina, Greece; email: stergios@cs.uoi.gr; K. C. Sevcik, Department of Computer Science, University of Toronto, 40 St. George Street, Toronto, Ont, M5S 2E4, Canada; email:kcs@cs.toronto.edu; M. Stumm, Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ont. M5S 3G4, Canada; email: stumm@eecg.toronto.edu.

1. INTRODUCTION

Realtime media streaming services are becoming increasingly popular in public and corporate data networks due to the enhanced access to content-rich media content that they offer. In comparison to traditional file downloading, media data streaming allows significantly faster playback initiation, provides guarantees for uninterrupted data decoding, and requires minimal buffering requirements from the client devices. Despite the improved data transfer bandwidth to the end-users thanks to broadband last-mile connectivity, superfluous deployment of backbone network equipment, and dropping cost of disk spindles, low-cost distribution of high-quality media streams remains an elusive target in the telecommunications industry for several years now [Gray and Shenoy 2000].

Several research projects and commercial products of media streaming servers have already established the feasibility of streaming stored files [Ozden et al. 1996; Bolosky et al. 1996; Chang and Zakhor 1996; Shenoy et al. 1998; Muntz et al. 1998]. Quite surprisingly, an entire class of the most efficient data compression techniques are not fully leveraged in current content distribution networks. Even though data networks can handle the resource management issues introduced by using advanced compression techniques for encoding data, the network servers deployed currently to provide storage and realtime streaming access to continuous media data can only handle compression schemes with specific features.

In particular, during the last decade, extensive qualitative and quantitative research has made the case that variable bit-rate encoding of video content provides significant compression advantages with respect to constant bit-rate encoding of equivalent quality. Several experimental studies estimate this advantage to be an order of 40% reduction in the size of the generated encoded files leading to a corresponding improvement in the number of streams that can be multiplexed over a shared network link [Gringeri et al. 1998; Lakshman et al. 1998].

The potential bandwidth savings of handling variable bit-rate streams over data networks has spurred considerable amount of research within the data networking community for achieving quality of service guarantees in packet switching of time-sensitive traffic [Verbiest et al. 1988; Sen et al. 1989]. In the meantime, the majority of proposed commercial and experimental media streaming servers that lie at the edge or middle of the network and provide streaming access to stored or temporarily cached digital media files, can only support constant bit-rate streams. This adversely affects the availability of media content that can take advantage of modern data networking infrastructure in a cost-effective way. Content distribution service providers are limited by the server technology to only offer streaming access to constant bit-rate data [D. Maggs, personal communication, 2002].

Most of the existing experimental or commercial media servers can only support constant bit-rate streams [Haskin and Schmuck 1996; Bolosky et al. 1996]. Alternatively, they store variable bit-rate streams using either peak-rate resource reservations that may reduce resource utilization but not increase server capacity, or statistical quality-of-service guarantees that may allow the system to get overloaded and miss transfer deadlines occasionally [Martin et al. 1996; Shenoy et al. 1998; Muntz et al. 1998]. The approach of retrieving variable bit-rate streams using constant rates may not solve the general problem either due to arbitrarily long playback initiation latency or large client buffer space requirement [Sen et al. 1997]. Even though the latest editions of commercial video production software support variable bit-rate encoding, the generated files can only be used for local access or downloading, rather than streaming from commercial media servers [Microsoft 2003].

We can attribute the previous trends to the fact that constant bit-rate encoding significantly simplifies resource reservation and bookkeeping issues within media servers and data networks. Additionally, earlier research raised several concerns about the scalable retrieval of variable bit-rate data striped across large disk arrays [Shenoy and Vin 1999]. In the present article, we unify several preliminary results into a solid system architecture that overcomes previous efficiency and scalability problems in storage management of variable bit-rate streams [Anastasiadis et al. 2001a, 2001b, 2002]. We introduce innovative resource allocation techniques which we experimentally evaluate using actual video streams and hardware disks. Thus, we demonstrate high-throughput data transfers and resource reservations that closely match measured device utilization. By focusing on the resource reservation and admission control of the system, we are also able to measure linear scalability in the number of concurrently supported users as a function of the disk array size. Finally, we utilize detailed simulated disk models to show that fault-tolerance against single disk failures is feasible with minimally wasted disk bandwidth.

Therefore, one fundamental question that we examine is that of organizing data transfers into time slots of appropriate length that keep the system operation both practical and efficient. We study issues of allocating server memory and disk storage space in order to maintain high disk access throughput. We also compare measured disk access delays against those predicted to establish efficiency in the admission control. The second question that we investigate has to do with organization of stream data across multiple disks. We exploit the predictable sequential access pattern that is common case in streaming, towards minimizing the number of disks utilized by a stream during every time slot of operation. As the number of disks in the system increases, we show that they are all equally utilized on average, thus avoiding hot spots and achieving load balancing. The third question that we pose is that of tolerating disk failures with minimally wasted resources during normal operation. With the data replication and disk bandwidth reservation schemes that we propose, we significantly improve the system throughput in comparison to simple schemes assumed in previous published research.

The rest of this article is organized as follows. In Section 2, we describe basic architectural definitions underlying our system design. In Section 3, we explain implementation details in the system modules of the prototype that we developed. In Section 4, we go over the experimentation environment that we used for our measurements. In Section 5, we investigate basic system

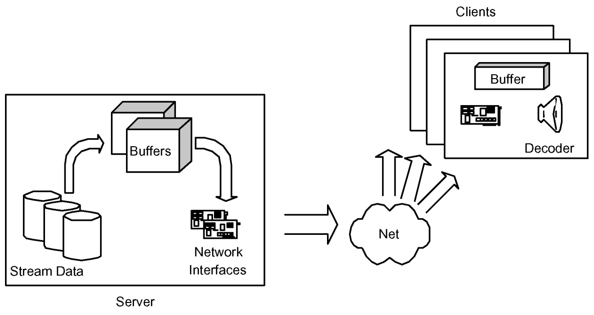


Fig. 1. Data files of digitally encoded media streams are stored across multiple disks of the media server. Decoding clients can connect and start playback sessions via the network.

operation parameters, while in Section 6 we study alternative disk striping policies. In Section 7, we present tradeoffs related to the data replication. The related work is summarized in Section 8, and our conclusions are further clarified in Section 9.

2. ARCHITECTURAL DEFINITIONS

We start with the description of the media server architecture that we propose, the presentation of a new disk space allocation scheme, and the definition of alternative policies for disk striping of stream data.

2.1 System Overview

We describe a distributed media server architecture that stores video streams on multiple disks. Clients with appropriate stream decoding capability send playback requests and receive stream data via a high-speed network as shown in Figure 1. The stored streams are compressed according to any encoding scheme that supports constant (or variable) quality quantization parameters and variable (or constant) bit rates respectively. The system operates according to the server-push model, where the server periodically sends data to each active client until either the end of the stream is reached, or the client explicitly requests suspension of the playback. The server-push model reduces the control traffic from the client to the server, and facilitates resource reservation at the server side. It is different from the client-pull model of video editing appliances or traditional file servers, where the client explicitly requests each individual data transfer.

In order to keep manageable the bookkeeping of the data transfers, we organize them in rounds of fixed duration T_{round}. In every round an appropriate amount of data is retrieved from the disks into the server buffers; concurrently, data from the server buffers are transmitted to the clients via the network. The server itself consists of multiple nodes connected to the high-speed network through different network interfaces. The amount of stream data periodically sent to the client is determined by the decoding frame rate of the stream and the buffering capacity of the client. The minimal operation requirement is that every client during each round receives the amount of data that will be needed for the decoding process during the next round.

2.2 The *Exedra* Media Server Architecture

The design of the media storage server that we propose is based on standard off-the-shelf components, currently used for data storage and transfer in commodity server systems.[1]

The *Transfer Nodes* are computers responsible for scheduling and initiating all stream data accesses of accepted playback requests. The disks storing stream data are connected to the *Transfer Node* hosts via either standard I/O channels (SCSI), or network switching equipment [Clark 1999]. Data from the disks are temporarily staged in the *Buffer* memory of the Transfer Node on their way to the client through the network interfaces. The system bus bandwidth within each Transfer Node is a critical resource that determines the number and capacity of the attached network and disk channel interfaces.

Playback requests arriving from the clients are initially directed to an *Admission Control Node*, where it is determined whether sufficient resources exist to activate the requested playback session either immediately or within a few rounds. The computational complexity of the general stream scheduling problem is combinatorial in the number of streams considered for activation, their length, and the number of reserved resources [Garofalakis et al. 1998]. Practically, we assume a limited acceptable initiation latency, and use a simple scheduling algorithm linear in the number of the stream rounds and the reserved resources. Should the admission control process become a bottleneck due to the incoming load and the needed detail of resource reservation, the admission control may be distributed across multiple nodes (Figure 2). We don't examine here the nontrivial concurrency control issues that arise in that case, though.

In traditional storage systems, it is relatively difficult to determine disk striping parameters customized to the needs of a constantly changing workload and system configuration [Alvarez et al. 2001]. However, for the common case of read-only sequential accesses in video streaming, the system load requirements are more predictable, and appropriate disk striping parameters can be determined a priori. The *Schedule Descriptor* stores the amount of stream data that needs to be retrieved during each round from each disk (Figure 3). Additionally, it also specifies the buffer space required and the amount of data sent to the client by the Transfer Nodes during each round. Such scheduling information is generated before a stream is first stored and is used both for admission control and for specifying data transfers during playback. Since this information changes infrequently, it can be replicated to avoid potential bottlenecks.

[1]Exedra means stage for live performances in modern Greek.

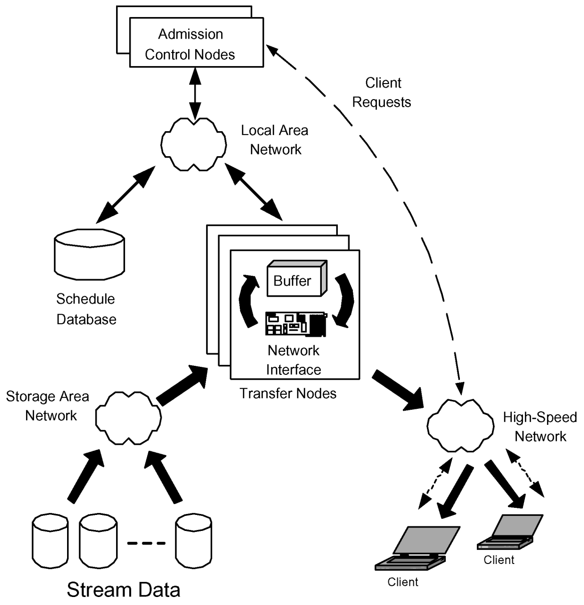


Fig. 2. In the *Exedra* media server architecture, stream data are retrieved from the disks and sent to the clients through the Transfer Nodes. Both the admission control and the data transfers make use of stream scheduling information maintained in the Schedule Database.

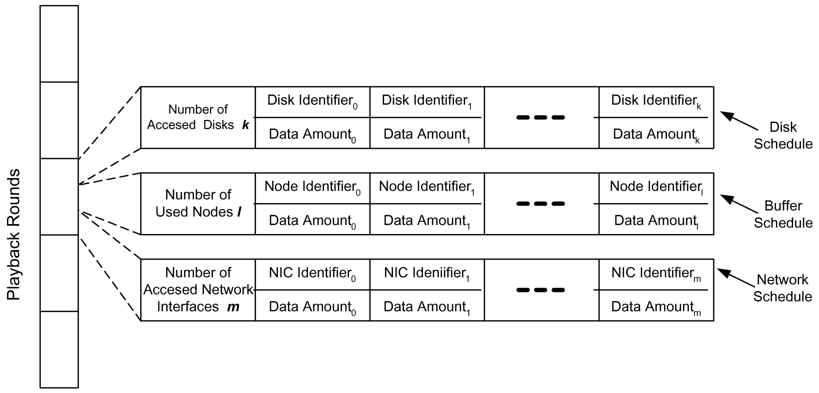


Fig. 3. Stream schedule descriptor for a stream stored on system with multiple transfer nodes. For each playback round, it identifies each disk accessed and the corresponding amount of transferred data, the buffer space reserved on each node, and the amount of data that has to be sent to the client through each network interface.

2.3 Stride-Based Disk Space Allocation

Stored streams are accessed sequentially according to a predefined (potentially variable) rate, while the maximum amount of data accessed from a disk during a round for a stream is known a priori. We exploit these features into the

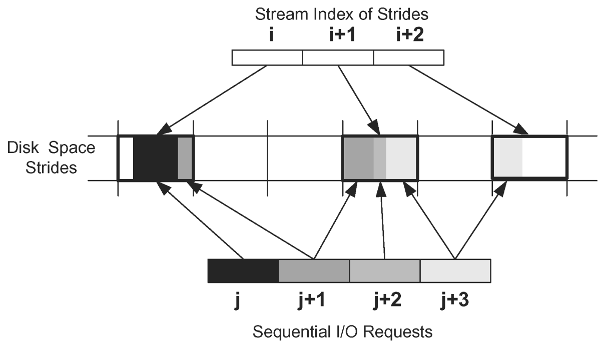


Fig. 4. The stride-based allocation of disk space is illustrated on one disk. A stream is stored in a sequence of generally non-consecutive fixed-size strides with a stride possibly containing data of more than one round. Sequential requests of one round are smaller than the stride size and thus require at most two partial stride accesses.

stride-based disk space allocation.[2] According to this scheme, disk space is allocated in large fixed-sized chunks called *strides* with size chosen larger than the maximum stream I/O request per disk during a round. Stride-based allocation eliminates external fragmentation, while internal fragmentation remains negligible because of the large size of the streams, and because a stride may contain data of more than one round (Figure 4).

Although stride-based allocation seems similar to extent-based [McVoy and Kleiman 1991] and other previous allocation methods [Shenoy et al. 1998], one difference is that strides have fixed size. Also, when a stream is retrieved, only the requested amount of data is fetched to memory, not the entire stride. 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 stream on each disk in a round. This allows us to avoid the arbitrary number of actuator movements required by previously proposed allocation methods [Chang and Zakhor 1996].

Arguably, storing the data of each disk request contiguously would further reduce the disk overhead to a single seek and rotation delay. However, due to external fragmentation, contiguous disk space allocation requires periodic compaction of the allocated disk space, which is a time-consuming process. Alternatively, stride padding could be used for storing a stream request on a single stride of a disk. This would prevent spanning of a stream request across two strides, and would lead to one disk head movement at most for each stream request. However, stride padding would waste disk storage space. With stride-based allocation we get most of the disk efficiency, while avoiding the extra overhead of alternative methods (see also Section 5.3).

2.4 Reservation of Server Resources

We consider system of N network interfaces, D disks, and Q transfer nodes. The *Network Striping Sequence* S_{mn} defines the amount of stream data, $S_{mn}(i, u)$,

[2]Stride-based allocation should not be confused with the term *strided access* used to describe regularity in I/O requests of parallel scientific applications [Nieuwejaar et al. 1996].

$1 \leq i \leq L_n$, $0 \leq u \leq N-1$, that the server sends through network interface u to a particular client during round i. Correspondingly, the *Network Sequence* S_n, $S_n(i) = \sum_{u=0}^{N-1} S_{mn}(i,u)$, specifies the total amount of data that the server sends to the client over time. Similarly, the stream *Buffer Striping Sequence* S_{mb} defines the server buffer space $S_{mb}(i,q)$, $0 \leq i \leq L_b$, $0 \leq q \leq Q-1$, with $L_b = L_n + 1$, that a client occupies on transfer node q during round i. The total buffer space occupied by the client over time is given by the *Buffer Sequence* S_b, with $S_b(i) = \sum_{q=0}^{Q-1} S_{mb}(i,q)$. Buffer space is reserved for the time period starting at the point that data are requested from the disk until the corresponding network data transfer has finished.

We assume that data are organized on the disks in strides. The stride size B_s is multiple of the *logical block* size B_l, which is multiple of the *physical sector* size B_p of the disk. Both disk transfer requests and memory buffer reservations are specified in multiples of the logical block size B_l. The *Disk Striping Sequence* S_{md}, $0 \leq k \leq D-1$, $0 \leq i \leq L_d - 1$ with $L_d = L_n$, determines the amount of data $S_{md}(i,k)$, that are retrieved from disk k in round i. The *Disk Sequence* S_d defines the total amount of data retrieved from the disks for a client over time. It can be derived from the network sequence S_n after applying quantization in terms of logical blocks. The cumulative number of blocks B_l retrieved from all the disks until round i is equal to

$$K^d(i) = \left\lceil \frac{\sum_{0 \leq j \leq i} S_n(j+1)}{B_l} \right\rceil. \tag{1}$$

Then, the disk sequence of a client at round i is defined as follows:

$$S_d(i) = (K^d(i) - K^d(i-1)) \cdot B_l. \tag{2}$$

The disk striping sequence S_{md} can be generated directly from the disk sequence S_d according to the striping policy used.

We assume that disk k has edge-to-edge seek time $T^k_{fullseek}$, single-track seek time $T^k_{trackseek}$, average rotation latency T^k_{avgrot}, and minimum internal transmission rate R^k_{disk}. For every client, the stride-based allocation policy guarantees that at most two disk-arm movements are needed per disk in each round. We keep bounded the total seek distance with the circular scan (C-SCAN) disk scheduling policy. At round i of the system operation, let M_i be the number of active streams. If we initiate the playback of stream j, $1 \leq j \leq M_i$, at round l_j of the system operation, the total access time on disk k has upper bound

$$T_{disk}(i,k) = 2T^k_{fullSeek} + 2M_i \cdot (T^k_{trackSeek} + T^k_{avgrot}) + \sum_{j=1}^{M_i} S^j_{md}(i - l_j, k)/R^k_{disk}, \tag{3}$$

where S^j_{md} is the disk striping sequence of client j. We count twice the $T^k_{fullSeek}$ parameter due to the disk-arm movement from the C-SCAN policy, while we apply a factor of two in the second term due to the stride-based method. The first term is applied once in the disk time reservation structure of each disk k. Additionally, each client j incurs during round i on disk k maximum access time

$$T_{disk}^{j}(i,k) = \begin{cases} 2 \cdot (T_{trackSeek}^{k} + T_{avgRot}^{k}) + S_{md}^{j}(i - l_j, k)/R_{disk}^{k}, & S_{md}^{j}(i - l_j, k) > 0 \\ 0, & otherwise \end{cases} \tag{4}$$

Let R_{net}^{u} be the bandwidth available at network interface u. If S_{mn} is the network striping sequence of client j, then the corresponding network transmission time reserved for client j in round i becomes $T_{net}^{j}(i,u) = S_{mn}^{j}(i - l_j, u)/R_{net}^{u}$. Similarly, if S_{mb}^{j} is the buffer striping sequence of client j, then the buffer space reserved for client j at transfer node q in round i becomes $B^{j}(i,q) = S_{mb}^{j}(i - l_j, q)$.

2.5 Definition of Striping Techniques

2.5.1 *Fixed-Grain Striping.* This method uses data blocks of fixed size B_f, a multiple of the logical block size B_l. It stripes the data blocks round-robin across the disks as shown in Figure 5(a). Let *stripe* be any sequence of D consecutive blocks of size B_f each, where the first block is stored on the first disk of the array. Let the cumulative number of blocks retrieved until round i for a specific client be

$$K^{f}(i) = \left\lceil \frac{\sum_{0 \le j \le i} S_d(j)}{B_f} \right\rceil. \tag{5}$$

If $\lfloor K^{f}(i)/D \rfloor - \lfloor K^{f}(i-1)/D \rfloor = 0$, all the blocks accessed for the client during round i lie on the same stripe, and the disk striping sequence S_{md}^{f} becomes

$$S_{md}^{f}(i,k) = S_{ps}^{f}(i,k) \cdot B_f \tag{6}$$

where

$$S_{ps}^{f}(i,k) = \begin{cases} 1, & \text{if } K^{f}(i-1)\ mod\ D < k\ mod\ D \le K^{f}(i)\ mod\ D \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

specifies whether disk k is accessed during round i or not.

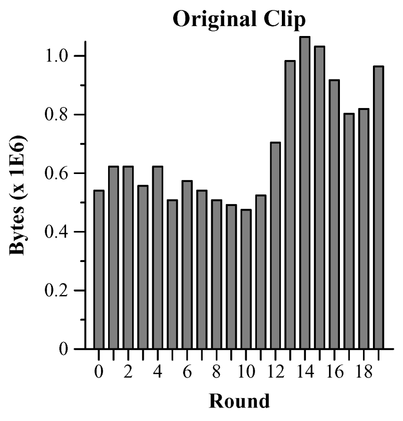
If $\lfloor K^{f}(i)/D \rfloor - \lfloor K^{f}(i-1)/D \rfloor > 0$, the blocks accessed for the client during round i lie on more than one stripe, and the disk striping sequence becomes

$$S_{md}^{f}(i,k) = (\lfloor K^{f}(i)/D \rfloor - \lfloor K^{f}(i-1)/D \rfloor - 1) \cdot B_f + S_{ps}^{f}(i,k) \cdot B_f, \tag{8}$$

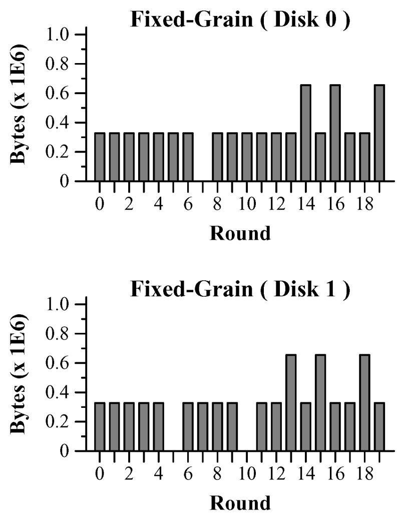
where

$$S_{ps}^{f}(i,k) = \begin{cases} 2, & \text{if } K^{f}(i-1)\ mod\ D < k\ mod\ D \le K^{f}(i)\ mod\ D \\ 1, & \text{if } k\ mod\ D > max(K^{f}(i-1)\ mod\ D, K^{f}(i)\ mod\ D) \\ 1, & \text{if } k\ mod\ D \le min(K^{f}(i-1)\ mod\ D,\ K^{f}(i)\ mod\ D) \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

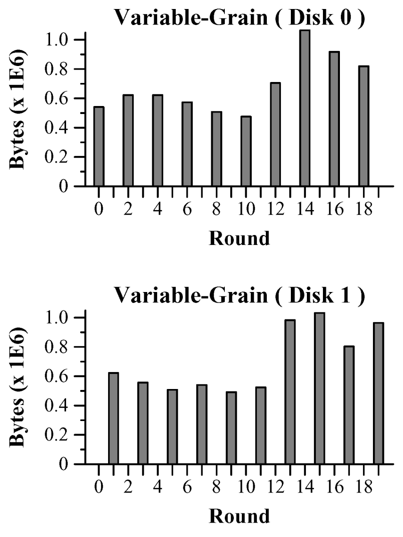
The first term in Eq. (8) refers to stripes fully accessed in round i, while the second term covers stripes partially accessed by the client. Depending on whether disk k occurs zero, once, or twice in the partially accessed stripes, $S_{ps}^{f}(i,k)$ takes the value 0, 1 or 2, respectively.



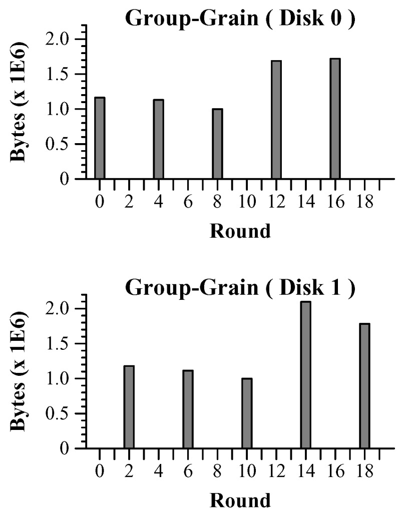
(a) Required Data Transfers



(b) Fixed-Grain Striping (B_f = 327, 680)



(c) Variable-Grain Striping



(d) Group-Grain Striping (G=2)

Fig. 5. Figure (a) shows the data requirements of twenty consecutive rounds in an MPEG-2 clip. With Fixed-Grain Striping (b), the needed blocks of size B_f are retrieved round-robin from the disks every round. In Variable-Grain Striping (c), a different disk is accessed in each round, according to the byte requirements of the original clip. In Group-Grain Striping (d) with G = 2, stream data worth of two rounds are accessed from a different disk every two rounds.

2.5.2 *Variable-Grain Striping.* The data retrieved during a round for a client are always accessed from a single disk that changes round-robin over time, as shown in Figure 5(b). Equivalently, the striping sequence $S^v_{md}(i,k)$ takes value zero or not during round i, depending on whether disk k is skipped or accessed by the client during that round. Therefore, we have

$$S^v_{md}(i,k) = \begin{cases} \big(K^v(i) - K^v(i-1)\big) \cdot B_l, & i\ mod\ D = k \\ 0, & otherwise \end{cases} \tag{10}$$

where

$$K^v(i) = \left\lceil \frac{\sum_{0 \le j \le i} S_d(j)}{B_l} \right\rceil. \tag{11}$$

is the cumulative number of logical blocks retrieved for the client until round i.

2.5.3 *Group-Grain Striping.* In this method, the amount of data required by a client over a period of G rounds is retrieved every Gth round from one disk that changes round-robin over time. We call *Group Size* the parameter G, $G \ge 1$, as shown in Figure 5(c). Then, the striping sequence for Group-Grain Striping is given by the expression

$$S^g_{md}(i,k) = \begin{cases} (K^v(i+G-1) - K^v(i-1)) \cdot B_l, & i\ mod\ G = 0 \wedge \lfloor i/G \rfloor mod\ D = k \\ 0, & \text{otherwise,} \end{cases} \tag{12}$$

where $K^v(i)$ is defined in Eq. (11). Essentially, every time round i becomes a multiple of G, an amount of data equal to $(K^v(i+G-1) - K^v(i-1)) \cdot B_l$ is retrieved from the single disk $k = \lfloor i/G \rfloor\ mod\ D$. In fact, Group-Grain Striping degenerates to Variable-Grain Striping for $G = 1$.

We should note that, if two playbacks begin from the same disk at rounds i and j ($i \not\equiv j (mod\ G)$), the playbacks do not have any disk transfers occurring in the same round. As a result, increasing G reduces the total number of disk requests and the access overhead, while allowing different playbacks to be served in separate rounds. In comparison, larger B_f in Fixed-Grain Striping results in accesses from different streams randomly coinciding on the same disk in the same round, which saturates the system with fewer streams. Also, longer round length achieves aggregation of disk transfers, but results in increased buffer space and playback initiation latency (Section 5.1). Therefore, Group-Grain Striping reduces the disk access overhead, without incurring the negative effects of alternative transfer aggregation methods.

2.6 Fault-Tolerance Issues

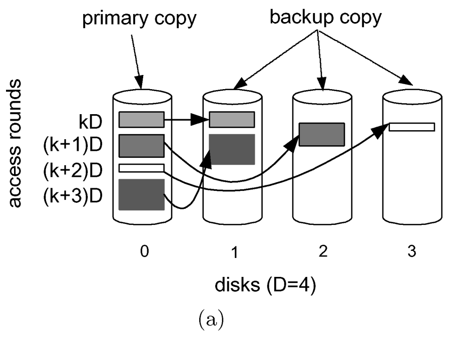
Achieving efficiency in disk space allocation and predictability in disk access delays can be a challenging task as a result of variability in the system resource requirements over time. Fault tolerance over disk arrays introduces the additional need for balancing the utilized storage space and data transfer bandwidth across different disks under conditions of normal operation and disk failure. In the present section, we describe a number of data redundancy and bandwidth reservation schemes that can tolerate single disk failures without service interruption.

2.6.1 *Data Redundancy Policies.* Due to the large number of components involved in a typical commercial server installation, it is necessary to assume device failures during its lifetime. Although the disks are likely to be distributed across multiple independent servers, building a single large disk array from distributed components has also been demonstrated in the past for media streaming services [Bolosky et al. 1996; Haskin and Schmuck 1996].

In order to improve the system reliability efficiently, we can use data redundancy techniques that minimize the extra computation, storage and bandwidth requirements with respect to the nonredundant case. In the present study, we focus on single disk failures rather than multiple disk failures that are less likely to occur simultaneously [Chen et al. 1994]. In the past, several parity-based techniques have been proposed that use error-correcting codes on the surviving disks to recover the missing data blocks from a failed disk [Chen et al. 1994]. Parity-based techniques trade extra bandwidth or memory buffer for reduced storage space. Since disk storage space is the least expensive resource of the three, mirroring rather than parity is becoming the preferred fault-tolerance technique [Bolosky et al. 1996; Gray and Shenoy 2000], and the one that we examine here.

With mirroring techniques, the data of each disk are replicated on one or more different disks. We refer to the original copy of the data as *primary* and the additional copy as *backup*. When one disk fails, its data remain available by retrieving their backup replicas from the rest of the disks. The total required storage space is roughly doubled across the disk array. The needed bandwidth on each disk can vary between $100\frac{n}{n-1}\%$ and 200% that of the nonredundant case for n disks, depending on the bandwidth reservation policy used.

2.6.2 *Replica Placement.* Although mirroring has previously been only used with data striped using fixed-size blocks, in principle it could be applied to variable-grain striping as well. During sequential playback of a media file with no failed disks, in a disk array of size D each disk is accessed every D rounds. In order to preserve the load-balancing property when a disk fails, data of a media file stored on consecutive disks could be replicated round-robin across the remaining disks (or a subset of them). The unit of replication corresponds to data retrieved by a client during one round of playback. We call *Deterministic Replica Placement* this mirroring approach. For example, Figure 6(a) shows disk 0 to store stream data requested during rounds $k \cdot D$, $(k+1) \cdot D$, $(k+2) \cdot D$ and $(k+3) \cdot D$, while the respective replicas are distributed round-robin among disks 1, 2 and 3. An alternative replication approach would use some pseudo-random sequence for specifying the disks that store the backup copies of one disk's primary data. We call that mirroring technique *Random Replica Placement* (Figure 6(b)).



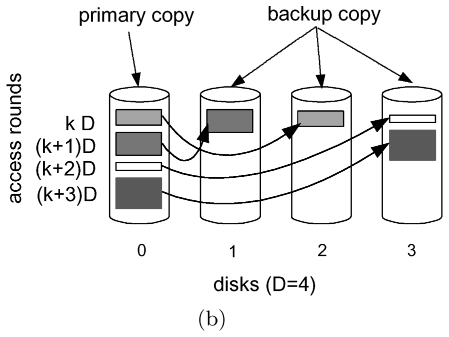
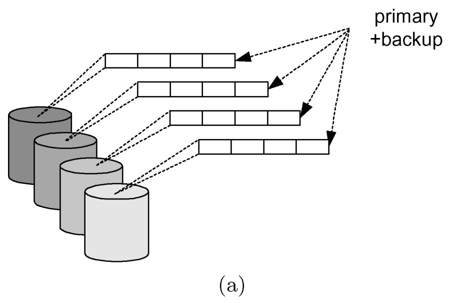


Fig. 6. (a) *Deterministic Replica Placement.* Data of a media file stored on one disk are replicated round-robin across the other disks. (b) *Random Replica Placement.* Data of a media file stored on one disk are replicated on three other disks randomly chosen.



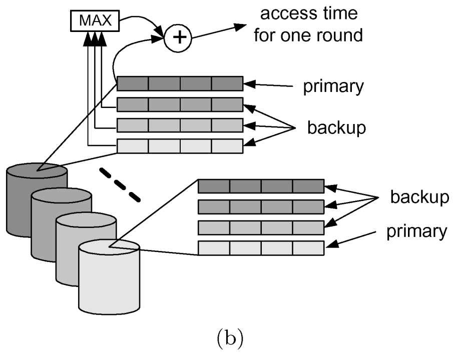


Fig. 7. (a) *Mirroring Reservation.* For each disk, there is a separate vector indexed by round number that accumulates the total estimated access time for retrieving primary and backup data in each round. (b) *Minimum Reservation.* For each disk, we maintain D separate vectors indexed by round number. One of them accumulates access delays for retrieving primary data. The remaining $D - 1$ vectors accumulate access delays for retrieving backup replicas that correspond to primary data stored on each of the other $D - 1$ disks. In each round, the sum of the primary data access time and the maximum of the backup data access times is reserved on each disk.

2.6.3 *Disk Bandwidth Reservation.* Retrieving backup replicas of data stored on a failed disk requires extra bandwidth to be reserved in advance across the surviving disks. This implies that the system will normally have to operate below full capacity. In what we call *Mirroring Reservation*, disk bandwidth is reserved for both the primary and backup replicas involved during a round of a media file playback (Figure 7(a)). Even though this seems reasonable, mirroring reservation doubles the disk bandwidth requirements of each stream in comparison to the nonredundant case. Ideally, we would prefer that the load normally handled by a failed disk is equally divided among the $D - 1$ surviving disks. In other words, tolerating one disk failure requires $\frac{1}{D-1}$ of its bandwidth capacity to be reserved on each other disk.

Given that only one disk is likely to fail, it is wasteful to reserve bandwidth on a disk for accessing backup replicas of primary data from more than one other disk. Thus, for each disk, we maintain D vectors indexed by round number of system operation. One vector keeps track of the total access time required for retrieving primary data. The remaining $D - 1$ vectors keep track of access delays due to backup data corresponding to primary data of the remaining disks. For every disk, we reserve the sum of the primary data access time and the maximum of the backup data access times in each round. We reserve the maximum across every other disk, since we don't know in advance which other disk is going to fail. We refer to this more efficient scheme as *Minimum Reservation* (Figure 7(b)). These two disk bandwidth reservation schemes can be orthogonally combined with the two replica placement policies that we introduced previously.

2.7 Load Balancing Enhancements and Other Extensions

The load of a failed disk can possibly be shared more fairly among the surviving disks if each backup replica is declustered across multiple devices. In the load balancing technique that we call *Backup Replica Declustering*, each backup replica is broken into blocks of fixed size B_d, an integer multiple of the logical block size B_l. The backup replica blocks corresponding to the primary data of each disk are distributed either round-robin or pseudorandomly across the rest of the disks, depending on whether deterministic or random replica placement is used. Alternatively, we can apply the *Dynamic Balancing* technique during normal operation. It takes advantage of multiple available data replicas by dynamically deciding to retrieve the replica stored on the disk expected to be the least loaded. The disk choice is based on access time estimations available through resource reservations that are made during admission control. It can be fully applied when all the disks are functional and is expected to reduce the load of the most heavily utilized disks in each round.

Handling multiple disk failures requires storing multiple backup replicas and making bandwidth reservations for more than one failed disk. In servers consisting of multiple nodes, failure of an entire node can also be handled gracefully, by keeping each disk of a node in a separate disk group and limiting the replication within each group. When a node fails, inaccessible data for each of its disks can be retrieved using replicas available on other disks of the corresponding group [Bolosky et al. 1996; Gafsi and Biersack 2000]. Provisioning for VCR functionality would require deallocation of previously reserved resources, when a stream playback is suspended or stopped earlier than its normal termination. This can be done in a straightforward way, when accumulating disk access delays separately for primary and backup data replicas, as was already described above.

3. PROTOTYPE IMPLEMENTATION

We have designed and built a media server prototype using C++ on AIX 4.2. The media server is responsible for stream file storage, resource reservation, admission control, buffer management, and data transfer scheduling (Figure 8). With appropriate configuration parameters, the system supports different levels of operation abstraction. We make full data transfers through raw-interface access to hardware disks in *Full Operation* mode. Alternatively, we gather

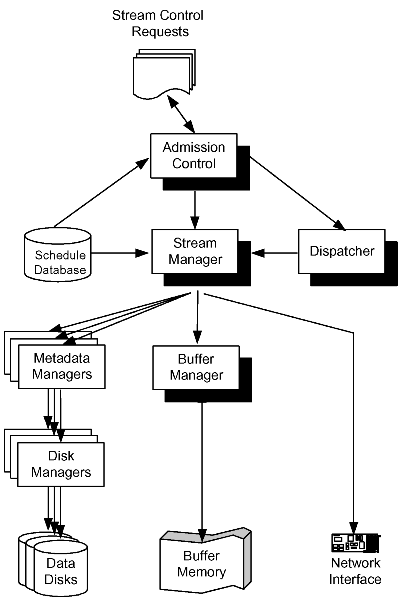


Fig. 8. System modules in the *Exedra* prototype implementation.

detailed disk access measurements with the DiskSim disk simulation package [Ganger et al. 1999] in *Simulated Disk* mode. Finally, when the system receives playback requests in *Admission Control* mode, we only apply admission control and resource reservation without transferring actual stream data.

3.1 Admission Control and Dispatching

The admission control module uses separate vectors to represent the allocated disk time, network time, and buffer space. On system startup, the disk time vectors are initialized to $2 \cdot T_{fullSeek}$, while the network time and buffer space are set to zero. When a new stream request arrives, the admission control ensures that the total service time of each disk in any round and the total network service time on each network interface may not exceed the round duration, while the total occupied buffer space on each node may be no larger than the corresponding server buffer capacity. If the admission control test is passed, then the resource sequences of the stream are added to the corresponding system vectors managed by the module, and the stream is scheduled for playback. At each upcoming round, the scheduled streams are activated and the corresponding data transfers are started.

3.2 Stream Scheduling

The stream management module initiates all the buffer reservation, disk transfer and network transfer requests for every active stream. The schedule descriptors provide the necessary information about the amount of data and the particular disks that should be accessed during each round. After allocating the required amount of buffer space, we prepare the disk transfer request that

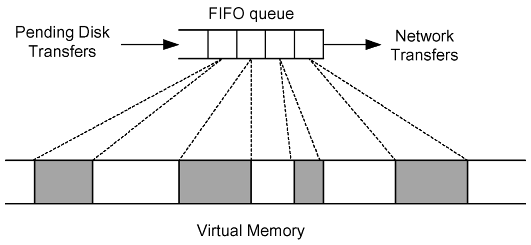


Fig. 9. Pending disk transfers are gathered in a fifo queue before they complete and allow the corresponding network transfers to begin.

specifies the buffer location, the stream file offset and the length of the transfer (Figure 9). We pass each request to the lower layers for the actual transfer to occur, and also insert a copy of the request descriptor into a fifo queue to keep track of the pending network transfers. We receive completion notification of the initiated data transfers through appropriate control information attached to each individual data block.

3.3 Metadata Management

Stream metadata management is organized in a layer above disk scheduling (Figure 8). It is responsible for disk space allocation during stream recording, and for translating stream file offsets to physical block locations during playback. The stream metadata are maintained as regular files in the host file system of each transfer node, while the stream data are stored separately on dedicated disks. The storage space of the data disks is organized in strides, with a bitmap that has a separate bit for each stride. A single-level directory maps the identifier of each recorded stream into a direct index of the corresponding allocated strides. A separate such directory exists for each different disk.

When a stream is striped across multiple disks, a stream file is created on each data disk. Each transfer request received by the metadata manager specifies the starting offset in the corresponding stream file and the number of logical blocks to be accessed. The stream index translates each request to a sequence of contiguous disk transfers. In order to keep the system performance predictable and unbiased by particular disk geometry features, when we allocate strides for a stream within each disk, we distribute them across all the zones of the disk. We create a separate metadata manager for each disk in order to be able to fully implement disk arrays consisting of heterogeneous disks [Anastasiadis et al. 2005]. This feature might prove crucial for the incremental growth and economic survival of large scalable media storage installations.

3.4 Disk Scheduling

The disk management layer is responsible for passing data transfer requests to the disks. The *dual-queue C-SCAN* disk scheduling manages the operation of each disk with a separate pair of priority queues, called *Request Queue* and *Service Queue*. At the beginning of each round, data transfer requests for the current round are added asynchronously into the request queue of each disk,

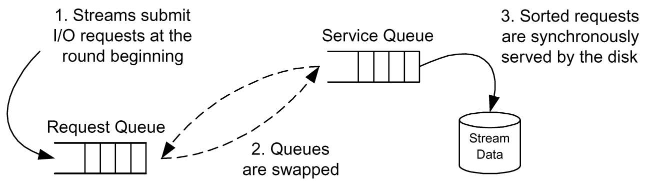

1. Streams submit I/O requests at the round beginning
Service Queue
3. Sorted requests are synchronously served by the disk
Request Queue
2. Queues are swapped
Stream Data

Fig. 10. Sequence of steps during the disk service of I/O requests using dual-queue C-SCAN.

where they are kept sorted in increasing order of their starting sector location. When all the requests have been gathered and the corresponding disk transfers of the previous round completed, the request queue of each disk is swapped with the corresponding service queue (Figure 10). Subsequently, requests from the service queue are synchronously submitted to the raw disk interface for the corresponding data transfers to occur. The dual-queue scheme prevents new requests from getting service before those of the previous round complete, thus keeping more stable the system operation.

3.5 Buffer Management

The buffer management module keeps the server memory organized in fixed size blocks of B_l bytes each, where B_l is the logical block size introduced in Section 2.4. The server memory is allocated in groups of blocks contiguous in virtual memory. We demonstrate the benefit of that in Section 5.2. For the allocation of buffer blocks we use a bitmap structure with an interface that can support block group requests. We support deallocations in granularity of individual blocks, as opposed to entire block groups obtained during allocation. This feature increases independence between disk accesses and network transfers. Paging of buffer space is prevented by locking the corresponding pages in main memory. In our design, we do not cache previously accessed data, unlike traditional file and database systems. We found that similar support for variable bit-rate streams would introduce several complications, especially in the admission control process. Instead, we assume that data transfers are done independently for each different playback [Bolosky et al. 1996].

3.6 Replication

The minimum reservation scheme requires maintaining number of vectors equal to the square of the number of disks. Each vector is accessed in a circular fashion and has minimum length equal to that of the longest stream expressed in numbers of rounds. When using large disk arrays, this might raise concerns regarding the computational and memory requirements involved. In practice, the reduction in unused bandwidth is diminishing as the number of disks increases beyond sixteen. Therefore, it makes sense to apply the data replication within disk groups of limited size, when the total disk array size becomes larger. This keeps limited the bookkeeping overhead and preserves the scalability of the system.

Table I. Features of the SCSI Disks Used in our Experiments

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

4. EXPERIMENTATION ENVIRONMENT

In the present section, we describe the hardware configuration that we used in our performance measurements, and the set of streams that composed our benchmarks. We also define a novel method for media server performance evaluation that is applicable to different system scales and stream characteristics.

4.1 Experimentation Setup

Even though we did most of our experiments in Admission Control mode, we used the Full Operation mode for low-level performance measurements, and the Simulated Disk mode for result validation of replicated disk arrays. We run the Full Operation mode on an IBM RS/6000 two-way SMP workstation with 233-MHz PowerPC processors running AIX4.2. We configure our system with 256-MB physical memory, and created the system and paging partitions on a 2-GB disk over a fast wide SCSI controller. We store the stream data on two 4.5-GB Seagate Cheetah ST-34501W disks (Table I) attached to a separate ultra-wide SCSI controller.[3] Although storage capacity is much larger in the latest disk drive models, the remaining performance features of the above two disks are typical of today's high-end drives. For the Admission Control and Simulated Disk modes, we assume Cheetah ST-34501 W disks.

We set the logical block size $B_l = 16$ KB, the physical sector size $B_p = 512$ B, the stride size $B_s = 2$ MB, and the round length $T_{round} = 1s$. For buffering, we use memory space of 32 MB per disk, organized in fixed size blocks of 16 KB. In our experiments, we drop the retrieved data to the $/dev/null$ device, thus leaving protocol processing and contention for the network outside the scope of the present study. However, from experiments that we did, we do not expect the network overhead to affect in any fundamental way the results presented here, unless it becomes the bottleneck resource in the system [Nagle et al. 2004].

We used six different variable bit-rate MPEG-2 streams of 30 minutes duration each. Each stream has 54,000 frames with a resolution of 720×480 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 1 Mbit/s and 9.6 Mbit/s.[4] We

[3]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 [IBM 1994].

[4]The digital versatile disk (DVD) specification sets a maximum allowed MPEG-2-bit rate of 9.8 Mbit/sec.

Table II. We Used Six MPEG-2 Video Streams of 30 Minutes Duration Each. Both the Autocorrelation and the Coefficient of Variation Shown in the Last Two Columns Change According to the Content Type

Content Type	Avg Bytes per rnd	Max Bytes per rnd	$\rho(1)$ per rnd	CoV per rnd
Science Fiction	624,935	1,201,221	0.885	0.383
Music Clip	624,728	1,201,221	0.782	0.366
Action	624,194	1,201,221	0.734	0.245
Talk Show	624,729	1,201,221	0.705	0.234
Adventure	624,658	1,201,221	0.739	0.201
Documentary	625,062	625,786	0.060	0.028

summarize the statistical characteristics of our clips in Table II. In our *mixed* basic benchmark, the six different streams are submitted round-robin. Where necessary, we also present experimental results from individual stream types.

4.2 Performance Evaluation Method

Although media server architectures have been investigated for more than a decade, performance parameters are usually evaluated in ad-hoc ways. Important decisions about the interarrival process, the stream scheduling, and practical system operation constraints vary inconsistently across different related studies. In general, we expect that a fair performance evaluation method: (i) demonstrates the system capacity, (ii) is applicable to a range of hardware configurations, (iii) is not biased against particular policies, and (iv) evaluates the practical operation of the system.

We assume the playback initiation requests arrive independently of one another in a Poisson process. Some workload characterization studies provide evidence that such an assumption holds in some cases [Almeida et al. 2001]. We control the system load through the arrival rate λ. If disk bandwidth is the bottleneck resource, we consider the ideal case of a system that incurs no disk overhead when accessing disk data. The streams have average data size S_{tot} bytes and the system consists of D disks with minimum transfer rate R^k_{disk} on disk k. Then, the completion rate μ, expressed in streams per round becomes

$$\mu = \frac{\sum_{k=0}^{D-1} R^k_{disk} \cdot T_{round}}{S_{tot}}. \tag{13}$$

The maximum arrival rate handled by the system is $\lambda = \lambda_{max} \leq \mu$, and creates enough load to demonstrate the performance benefit of arbitrarily efficient data striping policies. The system load is $\rho = \frac{\lambda}{\mu} \leq 1$.

When a playback request arrives, we check whether adequate resources are available for every round during playback. If the request cannot be initiated in the next round, the test is repeated for successive rounds until the first future round is found, where the requested playback can be started with guaranteed sufficiency of resources. We define as *lookahead distance* H_l the number of future rounds that are considered as candidate rounds for initiating a stream playback. Playback requests not accepted are discarded rather than being kept in a queue. Practically, a large lookahead distance leads to the possibility of a long potential waiting time for the initiation of the playback, while setting the lookahead distance too small can prevent the system from attaining full capacity.

We set the *basic lookahead distance* $H_l^{basic} = \lceil \frac{1}{\lambda} \rceil$. Intuitively, setting $H_l = H_l^{basic}$ allows the system to consider for admission control the number of upcoming rounds (on average) that will pass until another request arrives. More generally, we define as *lookahead factor* F_l the fraction $F_l = \frac{H_l}{H_l^{basic}}$, and set $F_l = 1$. In most practical situations, the rejection ratio should also be kept low, for example close to 1%. This can typically be achieved by operating the system at loads lower than the maximum 100%. All the experiments presented in this study are repeated until the half-length of the 95% confidence interval on the performance measure of interest lies within 5% of the estimated mean value. Our basic performance objective is to maximize the average number of active playback sessions that can be supported by the server.

4.3 Summary of Differences from Previous Studies

We summarize important differences in our assumptions from those of previous related studies.

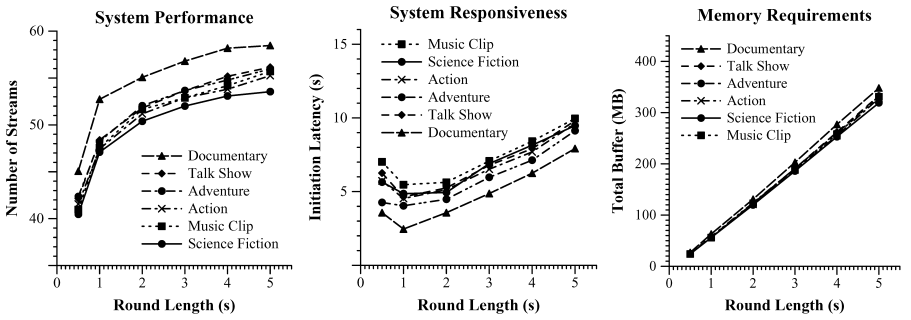
(1) We assume that playback requests arrive according to a Poisson process, which closely resembles system operation in practical situations.
(2) We adjust the system load according to the available resources in the system, which makes fair the comparison of different system configurations.
(3) We introduce a load-adjusted upper bound in the number of future rounds considered for initiating a playback requests. This is a reasonable compromise in the playback initiation latency, while allowing most of the system capacity to be reached.
(4) For each arriving request, we make an exhaustive search within the specified window of rounds for a location where playback can be initiated.
(5) We keep the rejection ratio of playback requests acceptably low.

5. STUDY OF BASIC PARAMETERS

We begin our experiments by evaluating the effect of the round length on the system operation along with the dependence of the disk throughput on the buffer organization. We also investigate implications of the disk space allocation on the disk bandwidth utilization, and we compare resource reservation statistics to actual utilization measurements.

5.1 Choosing the Right Round Length

We use the Admission Control mode with a four-disk array at load $\rho = 80\%$, lookahead factor $F_l = 1$ and Variable-Grain Striping. Experiments with other parameter values lead to similar conclusions. The system performance depends on the ratio between useful transfer time and mechanical overhead in the disk accesses. Longer rounds increase the data size of the disk transfers and improve the disk operation efficiency with a diminishing returns effect (Figure 11(a)).



(a) Number of Streams (b) Latency (c) Total Buffer

Fig. 11. The round length has interesting effects on the system operation parameters. Although a longer round (a) improves, up to a point, the average number of active streams, (b) it can also extend the stream initiation latency, and (c) linearly increases the total buffer space required.

In Figure 11(b), we notice that the average initiation latency increases almost linearly with rounds longer than one second. The latency depends on both the number and the length of the rounds tried during stream admission control. From Figure 11(c), it follows that the total buffer space required increases linearly with longer rounds. This is a result of the proportional increase in the amount of data retrieved from the disks during each round. In the rest of our study we use round length of one second to achieve high throughput under reasonable initiation latency and buffer requirements. Frequent use of the same round length in other studies facilitates the comparison of our results with those of previous related research.

5.2 Contiguity of Buffer Allocation

We measure the disk throughput at different sizes of I/O requests and degrees of contiguity in the buffer space allocated for each request. We initiate disk requests of a specific size at different locations uniformly distributed across the disk space. We transfer the disk data to pages locked in main memory and organized in blocks of size B_l. In Figure 12(a), we depict the average throughput when we invoke a separate `read()` call for each buffer block. Increasing block sizes change the disk throughput by a factor of three across different request sizes, while increasing request sizes change the throughput by more than a factor of two for a particular block size.

In Figure 12(b), we use the `readv()` system call with parameters the pointer to an array of address-length buffer descriptors and the number of the descriptors. The array size is typically limited to a small number (e.g., $IOV\_MAX = 16$ in AIX 4.2). Although we expected improved performance due to the collective information accompanying each `readv()` call, the measured throughput was less than half of what we measured with `read()`. Proper explanation of this pathological behavior would require internal knowledge of the readv() implementation in AIX 4.2, which we don't have. Nevertheless, we describe below a user-level approach to overcome performance dependences on the particular implementation of readv().

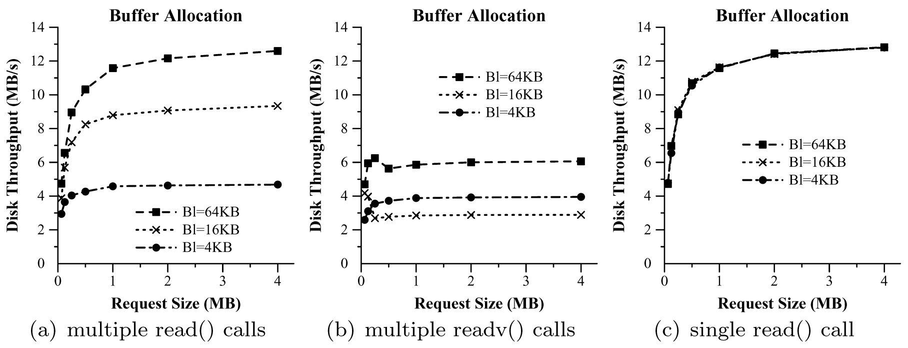


Fig. 12. (a). When we use a separate disk transfer for each buffer block, disk throughput depends critically on the block size. (b) Grouping multiple block transfers into a single call by using `readv()` decreases by more than 50% the achieved disk throughput. (c) Invoking a single `read()` for each request keeps disk throughput consistently high, independently of the buffer block size.

In Figure 12(c), we use only a single `read()` call for each request, in a way similar to our prototype implementation. It only requires contiguity at the virtual address space, without actual control on the physical addresses of the underlying pages. With average disk transfers of 625,000 bytes in our MPEG-2 clips, we achieve disk throughput higher than 11 MB/s, consistently with the minimum sustained rate 11.3 MB/s advertised in our disk data sheet. Although the achieved performance is similar to that of Figure 12(a) with large blocks, large block sizes reduce the benefit from multiplexing requests of different sizes, and shrink the number of accepted streams (see also Section 6).

In conclusion, contiguity in the buffer space allows a relatively small block size to guarantee both high number of streams and efficient disk transfers, thus simplifying the performance tuning of the system. The drawbacks are the software complexity of managing buffer ranges instead of fixed buffers, and the external fragmentation that requires a small percentage (10–15%) of buffers to remain unused.

5.3 Contiguity of Disk Space Allocation

If the disk space for each request was allocated contiguously, each disk access would require a single head movement and not a maximum of two incurred by stride-based allocation. In Figure 13, we measure the disk bandwidth utilization when retrieving streams allocated on a disk using different stride sizes. As we increase the stride size from 2 MB to 16 MB, the achieved stream throughput (not shown) remains about the same, while the disk bandwidth utilization drops by 2–3%. This percentage indicates that stream disk accesses are dominated by useful data transfers rather than mechanical overhead. As a result, the benefit from contiguous disk space allocation would be limited in an environment of multiple streams striped across several disks.

5.4 Resource Reservation Efficiency

In a system with 2 disks and 64-MB buffer memory, we compare the reserved and measured resource utilizations across different stream types at Full

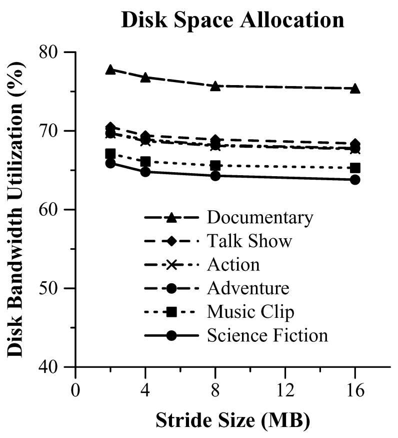


Fig. 13. Varying the stride size between 2 MB and 16 MB changes only marginally (2–3%) the disk bandwidth utilization across different stream types. This shows the limited benefit from larger strides, or contiguous disk space allocation. A single disk was used with load $\rho = 80\%$, lookahead factor $F_l = 1$ and Variable Grain Striping.

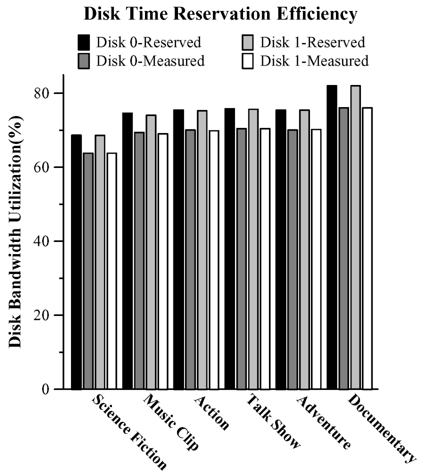


Fig. 14. In a two-disk configuration, the measured disk utilization is balanced between the two disks. On each disk, the difference between the reserved and measured disk utilization remains within 5%.

Operation mode. We set the buffer block size to $B_l = 16$ KB, the stride size to $B_s = 2$ MB, the system load $\rho = 80\%$, and use Variable Grain Striping. The average number of active streams is roughly between 20 and 25 depending on the stream type.

The measured busy time in most rounds is less than the total disk time reserved. In only less than 1% of the rounds, the measured busy time exceeds the reserved due to unexpected types of disk overhead. However, all the discrepancies could be masked from the client with an extra round of added playback initiation latency. Other than that, we got stable prolonged system operation at high loads without any observed problems. In Figure 14, we illustrate the fraction of

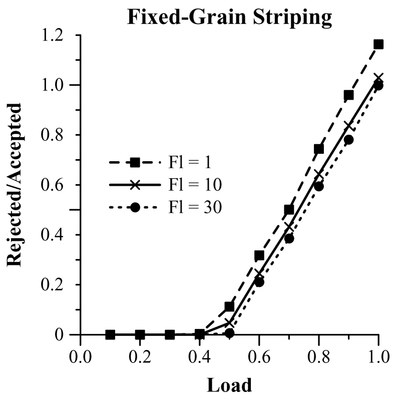


Fig. 15. For $B_f = 327,680$ bytes, 16 disks and mixed workload, we depict the total rejected over accepted streams during the measuring period. The ratio increases linearly as the load exceeds 50%, but changes by less than 20% as F_l varies between 1 and 30.

time each of the two disks is found (measured) or expected (reserved) to be busy. We notice that the load is equally balanced across the two disks. In addition, the busy fraction that we reserve does not exceed the corresponding measured by more than 5%. This allows the admission control procedure to successfully offer quality of service guarantees, without disk bandwidth underutilization.

6. COMPARATIVE STUDY OF DISK STRIPING

In the present section, we examine the effects of the system load and the lookahead distance parameters on the system performance. Subsequently, we compare the throughput and scalability of alternative disk striping policies. We demonstrate that the number of streams supported by the system scales linearly with the number of disks, while the striping policy can affect significantly the system throughput. With reasonable technology projections, our conclusions remain valid in the foreseeable future.

6.1 Study of Fixed-Grain Striping

We begin with a study of Fixed-Grain Striping. In Figure 15, we observe the ratio of rejected over accepted streams to be zero at load below 50%, and increase linearly beyond that threshold. Since changing the lookahead factor F_l from 1 to 30 affects only marginally the number of accepted streams, we set $F_l = 1$. In Figure 16, we measure the number of active streams as the block size increases from $B_f = 32$ KB to $B_f = 1$ MB in steps of 32 KB. We notice that the number of active streams increases when B_f approximates 320 KB, and drops at larger or smaller block sizes. This observation affects the choice of the block size that maximizes the number of streams, and confirms results from previous studies [Shenoy and Vin 1999].

The disk busy time normalized by the round time corresponds to the expected disk utilization. Part of the normalized access time is actuator overhead and decreases as the block size becomes larger. In Figure 17, we observe the maximum difference in the reserved busy times among different disks in a round

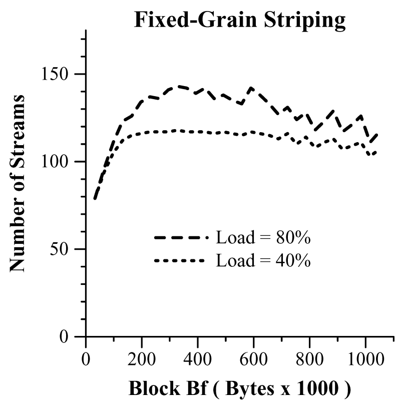


Fig. 16. At load values 40% and 80%, a maximum number of streams is achieved at $B_f = 327,680$. We experimented with 16 disks and mixed workload over a range of B_f between $32,768$ and $1,048,576$ at steps of $32,768$ bytes.

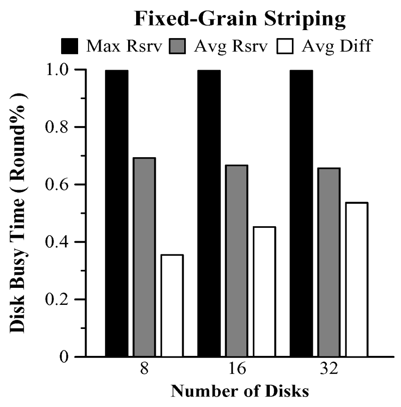


Fig. 17. The maximum and average reserved disk access time remain almost the same as we change the number of disks, while there is increase in the maximum difference (AvgDiff) between the access times reserved across the disks. We use mixed workload at load 80%.

(Avg Diff) to increase from almost 35% to above 50%. However, the average reserved time remains almost the same (within 2%) across the disks of an array. Also, the average disk bandwidth utilization drops only slightly from 69% with 8 disks to 66% with 32 disks. This implies that the useful transfer capacity of the system increases almost linearly as more disks are added. As we vary the disk array size from 4 to 64 in Figure 18, the block size $B_f = 320$ KB constantly maximizes the number of streams. At 80% load, the number of supported streams increases from 39 with 4 disks to 144 with 16 disks and 550 with 64 disks. This is within 9–14% of what perfectly linear scalability would achieve. Our observations at load 40% are similar.

6.2 Comparison with Variable-Grain Striping

In Figure 18, we examine the scalability of Variable-Grain Striping. At load $\rho = 80\%$, the number of streams increases from 48 with 4 disks to 786 with

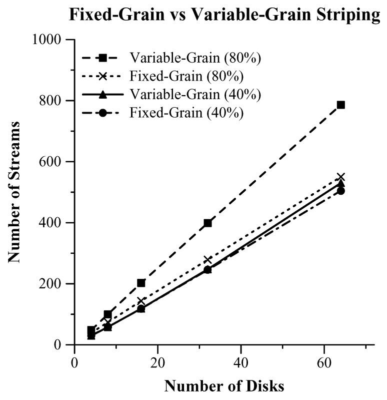


Fig. 18. At load $\rho = 80\%$, the streams increase almost linearly from 39 to 550 with Fixed-Grain Striping in comparison to increase from 48 to 786 with Variable-Grain Striping. At load $\rho = 40\%$, the two striping policies achieve about the same number of active streams.

64 disks. Thus, the number of streams remains within 3% of what perfectly linear scalability would achieve as a function of the number of disks. The advantage of Variable-Grain Striping over Fixed-Grain Striping increases from 23% with 4 disks to 43% with 64 disks.

A straightforward way to calculate the stream capacity of the system is to divide the total disk bandwidth of the server by the average transfer bandwidth required by a stream. This is not a tight upper bound since it ignores both the disk access overhead and the transfer size variability of variable bit-rate streams. From Table I, the transfer capacity of a disk is equal to 11.3 MB/s, while from Table II the average transfer bandwidth of a stream is equal to 0.624 MB/s. Therefore, the maximum capacity of the server cannot exceed 18 concurrent streams with one disk, 72 with 4 disks and 1158 with 64 disks. By dividing the measured throughout of the system with the estimated maximum capacity, we find that Fixed-Grain Striping achieves 44% of the maximum capacity with 64 disks, while Variable-Grain Striping gets close to 68%.

In Figure 19, we need a larger block size to maximize the performance of Fixed-Grain Striping, as the content type changes from Science Fiction to Documentary. Overall, Variable-Grain Striping maintains an advantage over Fixed-Grain Striping between 11% and 50%. The explanation is twofold. First, Variable-Grain Striping achieves disk access efficiency by accessing only one disk for a stream during each round. Second, by allowing variability in the data request sizes of different streams, there is a multiplexing effect that can hide disk access delay peaks from individual streams in Variable-Grain Striping.

6.3 Effect of Technology Trends

To project disk technology improvements, we extend past compound growth rates linearly into the future. Thus, we assume 30% increase in internal disk transfer rate per year, and 23% decrease in seek distance [Ng 1998]. For the track seek time, which is dependent on the square root of the seek distance, we assume decrease of 12% per year. Finally, we assume a rotation speed increase

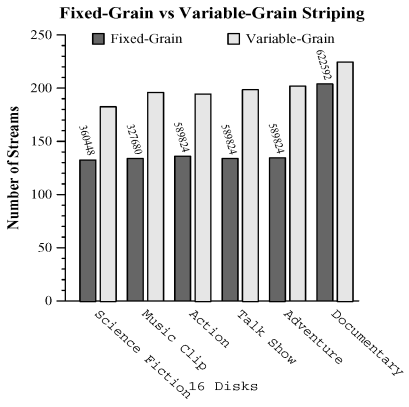


Fig. 19. The advantage of Variable-Grain Striping over Fixed-Grain Striping varies between 11% in the Documentary and 50% in the Adventure. The block size shown for Fixed-Grain Striping maximizes the number of streams in a range of block sizes between 32 KB and 1,024 KB. We set the load equal to 80%.

of 12% per year [Ruemmler and Wilkes 1994]. Although disk capacity and performance are expected to continue improving, the rate of advancement most likely will slow down [Grochowski and Halem 2003]. Conservatively, we presume that stream types and sizes will remain the same, which ignores potential demand for higher resolution and more content-rich streams.

Until now, we only considered Group-Grain Striping with $G = 1$ (Variable-Grain Striping), which maximizes the number of streams with the assumed disk technology. But as the disk access time drops, we found it beneficial to increase G, so that G rounds worth of stream data are transferred in a single round. This essentially reduces the amount of time spent on disk overhead during each round, without negatively affecting the initiation latency or the buffer requirements of the streams. Specifically, when using the mixed workload, we found that two years into the future, the number of streams supported with Group-Grain policy at $G = 2$ increases by 35% when compared to Fixed-Grain Striping. Five years into the future, the corresponding benefit of Group Grain Striping at $G = 3$ remains 29% (see Figure 20). Thus, under reasonable assumptions about technological improvements, there are significant performance improvements when using Group-Grain Striping instead of Fixed-Grain Striping.

7. FAULT-TOLERANCE COSTS

In the present section, we experimentally compare our data replication and bandwidth reservation techniques with respect to the average number of active playback sessions that can be supported by the server. We provide supplementary performance intuition with statistics on reserved and utilized disk access time across different stream types and numbers of disks.

7.1 Replica Placement Comparison

In Figure 21, we observe that data replication with mirroring reservation cuts almost into half the throughput achieved with no replication. Additionally,

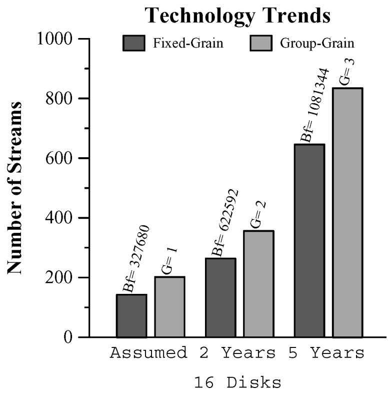


Fig. 20. With reasonable technology projections, after two years of progress, Group-Grain Striping maintains an advantage of 35% over Fixed-Grain Striping. The corresponding benefit in five years remains 29%. The shown values of B_f and G were found to maximize the throughput of the two policies respectively.

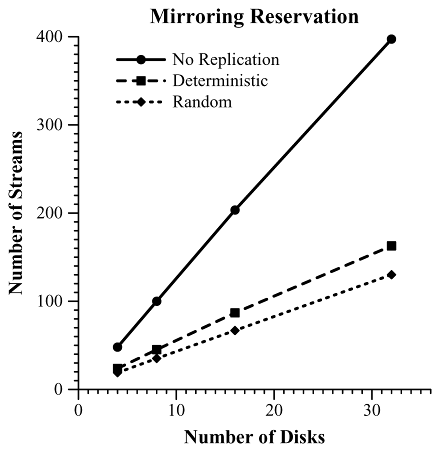


Fig. 21. The mirroring reservation scheme cuts into half the number of streams supported by the no-replication case. Deterministic replica placement sustains an advantage of 25% or more relative to random replica placement under the mixed stream workload.

deterministic replica placement achieves a throughput advantage of 25% or more relative to random replica placement, because the former is more consistent in fairly distributing the access load across the disks, especially in small disk arrays. When a disk fails, about 25–30% of the reserved disk bandwidth remains unused under both placement policies. This is not surprising, since the mirroring reservation scheme allocates disk bandwidth for both the primary and backup replicas of each accepted stream. We alleviate this inefficiency by using the minimum reservation scheme in the subsection that follows.

7.2 Minimizing Reserved Bandwidth

The minimum reservation scheme improves disk utilization by allocating on each disk the extra time required for accessing backup replicas of only one other

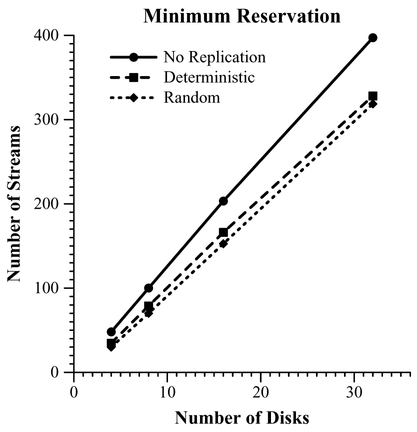


Fig. 22. With minimum reservation, the throughput advantage of deterministic over random replica placement drops from 15% to 3%. The corresponding throughput disadvantage of deterministic placement with respect to no replication drops from 28% to 17%.

disk. In order to ensure that any single disk failure can be handled properly, each disk keeps track and reserves the maximum additional time required for handling potential failure of any other disk. We calculate this maximum requirement separately for each disk in each round.

In Figure 22, we notice that the number of supported streams with deterministic placement is 21% lower than without replication on eight disks, and drops to 18% and 17% on sixteen and thirty two disks, respectively. From the way that the minimum reservation scheme allocates disk bandwidth, we would expect the total unutilized bandwidth during normal operation to be that of one disk. Correspondingly, the percentage of unused bandwidth should be inversely proportional to the number of disks in the system. For example, with 16 disks, only $\frac{1}{16} = 6.25\%$ of the total disk bandwidth should remain unused during normal operation. In practice (Figure 23), this does not hold because of the $MAX()$ operator applied for reserving access time of the backup replicas, in combination with the rounding error from the relatively large size of the data retrieved for a stream in each round.

In Figure 24, we measure the disk busy time using the Simulated Disk mode. Under normal operation, the deterministic placement keeps the disks busy for time 6% lower than the reserved for primary data access. When a disk fails, the remaining disks are busy for time 14% less than the total reserved. This is a significant improvement in comparison to the 25–30% difference between reserved and measured time that we reported for mirroring reservation. We should keep in mind that, with four disks, one third of the bandwidth of each disk has to be reserved for the case that one disk fails. This fraction drops as the disk array size increases (Figure 23).

Interestingly, when a disk fails, the difference between reserved and utilized access time increases from 6–8% to 13–14%. At first glance this discrepancy appears as reduced accuracy in access time estimation. In fact, the MAX() operator reserves enough access time to ensure uninterrupted system operation for

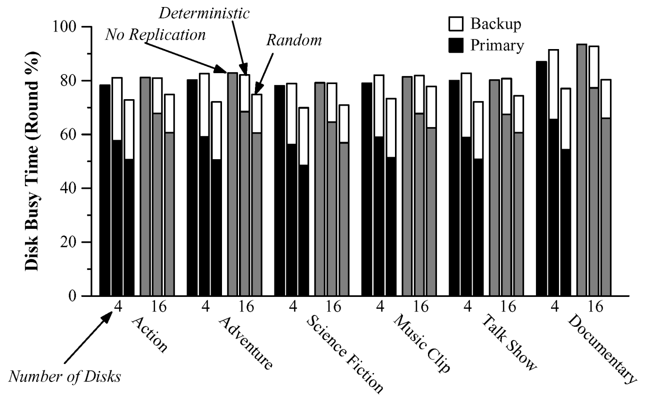


Fig. 23. With minimum reservation and deterministic replica placement, the disk time reserved for backup accesses drops from about 24% to 14% of the round length as the number of disks increases from 4 to 16. With random replica placement, the respective percentage drops from about 22% to 14%.

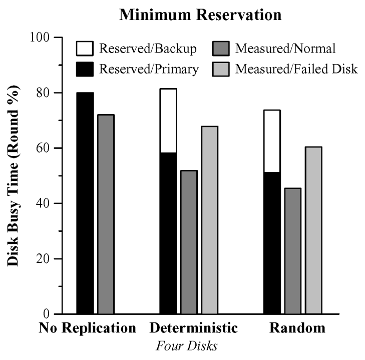


Fig. 24. During normal operation, the average disk access time measured in each round remains within 6–8% below the time reserved for primary data accesses. When a disk fails, the measured access time is 13–14% below the total reserved.

any failed disk, while the time that we report refers to a particular inaccessible disk. Overall, we believe that some limited discrepancy between predicted and measured access time leaves a reasonable cushion space for stable operation. This makes the system more robust, and guards it against nondeterministic factors hard to predict (e.g., system bus contention).

7.3 Improving Load Balancing

In Figure 25, we consider the case of declustering backup replicas across multiple disks using a fixed block size B_d in order to let the failed-disk load be more fairly shared among the surviving disks. With small block sizes, better load balancing leads to some limited throughput improvement of less than 3% with eight disks. With larger block sizes, load balancing gets successively worse and throughput decreases, while at $B_d = 1.2E6$ bytes we have a threshold beyond

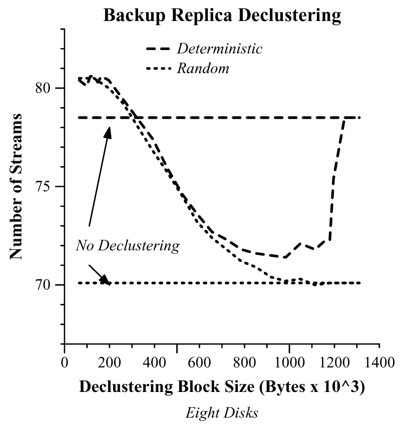


Fig. 25. There is a minor gain from applying backup replica declustering to deterministic placement, while the best throughput achieved with random placement approaches that of deterministic.

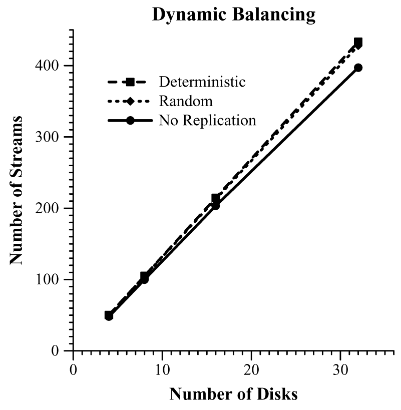


Fig. 26. During normal operation, accessing the replica of the least loaded disk improves the throughput by about 5–10% with respect to the nonreplicated case. The gain tends to increase as the disk array size increases.

which there is no declustering.[5] From additional measurements that we made, we found that even the least-loaded disk is expected to remain more than 80% utilized under deterministic replica placement with no declustering. We conclude that declustering is only worthwhile with small declustering block sizes, and its overall benefit is limited in media streaming.

With multiple data replicas available, better load balancing can be achieved by choosing the replica stored on the least-loaded disk during admission control. Thus, we leverage data replication to improve the system throughput, rather than tolerating disk failures. In Figure 26, we observe that the replica placement policies support 5–10% more streams than the nonreplicated case. The relative difference between the two placement policies is insignificant because the gain from the dynamic replica access exceeds the load balancing improvement

[5] $Bd = 1.2E6$ is the maximum amount of data retrieved for a stream in one round and originates from the encoding parameters.

from deterministic placement. Although choosing the least-loaded disk for each disk access can remove hot spots from the disk array, it turns out that sequential workloads utilize the disks equally, leaving a marginal additional performance benefit to be achieved with the above policy.

8. RELATED WORK

Existing media server designs either only support constant bit-rate streams [Tobagi et al. 1993; Berson et al. 1995; Buddhikot and Parulkar 1995; Ozden et al. 1996; Bolosky et al. 1996], or make resource reservations assuming a fixed bit rate for each stream [Martin et al. 1996; Shenoy et al. 1998], or have only been demonstrated to work with constant bit-rate streams [Santos et al. 2000]. Nevertheless, detailed resource reservation for variable bit-rate streams in a media server can increase throughput by more than a factor of two when compared to peak rate resource reservation [Makaroff et al. 1997]. The present article brings together and builds upon results that we presented in several previous research papers [Anastasiadis et al. 2001a, 2001b, 2002]. For the first time, low-level resource management decisions and utilization measurements are combined with high-level policies and experiments about admission control, disk striping and data replication towards proposing a complete scalable fault-tolerant continuous media server architecture.

8.1 Design and Implementation of Video Server Systems

One of the better known media servers is the Tiger fault-tolerant video file-server by Bolosky et al. [1997] that only supports distributed storage of streams with constant bit rates. The Fellini storage system by Martin et al. [1996] uses a client-pull model for accessing constant or variable bit-rate stream data and does resource reservation based on the worst case requirements of each stream. In the continuous media file server proposed by Neufeld et al. [1996], detailed resource reservation is done for each round, but the study focuses on storing data of an entire stream on a single disk. The Symphony multimedia file system by Shenoy et al. [1998] integrates data of different types on the same platform, with admission control based on peak rate assumptions. Instead, we customize our design to the storage of variable bit-rate stream data to maximize efficiency.

The RIO storage system by Muntz et al. [1998] is designed to handle several different data types, including video streams. The authors base the admission control on statistical predictions, and randomly distribute the stream blocks across different disks for load balancing. An early design of distributed data striping is described by Cabrera and Long [1991]. Their data striping and resource reservation policies do not take into account special requirements of variable bit-rate streams, while striped data pass through an intermediate node before being sent to the clients.

Keeping separate the metadata management of each disk relates in several ways to the design of the backing store server for traditional data by Birrel and Needham [1980]. The idea of grouping together buffer blocks can be traced back to the FFS design [McKusick et al. 1984]. Although stride-based allocation seems similar to extent-based [McVoy and Kleiman 1991] and other allocation methods [Chang and Zakhor 1996; Shenoy et al. 1998], one basic difference is that strides have fixed size. More importantly, when a stream is retrieved, only the requested amount of data is fetched to memory and not the entire stride.

Stream multicasting techniques reduce the required disk and network bandwidth by having the transmitted data shared across multiple clients [Eager et al. 2001]. In that case, the streaming server broadcasts different stream segments according to a specific schedule and does not transmit an entire stream separately for each playback request. Several testbed implementations demonstrate the feasibility of multicast streaming, depending on the availability of networking equipment with multicast support [Bradshaw et al. 2003]. Thus, the Tabbycat server prototype proactively broadcasts popular video streams according to a fixed schedule, instead of reactively responding to individual client requests [Thirumalai et al. 2003]. The server architecture that we propose provides real-time guarantees in the retrieval of stored variable bit-rate data and could be used for proactive multicasting after appropriate adjustment in the transfer schedule.

8.2 Disk Striping Policies

Chang and Zakhor [1994], in their study for multilayer encoded streams, describe *nonperiodic interleaving*, where an entire stream is split into parts equal to the number of disks and each part is stored on a separate disk. They also describe *periodic interleaving*, where stream data for a round are stored on a single disk that changes round-robin every round [Chang and Zakhor 1997]. In a different study for variable bit-rate streams, Chang and Zakhor [1996] suggest for future theoretical and experimental work the comparison of periodic interleaving to what they call hybrid data placement, where fixed-size blocks of stream data are placed round-robin across the disks.

Shenoy and Vin [1999] the fixed-size block striping of stream data, with both analytical and simulation methods. They basically investigate a tradeoff between disk access overhead and load imbalance between disks. They find that as the number of disks increases, the load imbalance across the disks becomes higher. They conclude that a smaller block size should be chosen in order to compensate for the imbalance, but that leads to higher disk actuator overhead. They claim that the number of supported streams increases only sublinearly with the number of disks, and conclude that only disk arrays of limited size can operate efficiently. Their paper leaves unclear how the load of the system is determined, and what process is assumed for the arrival of the client requests. Also, each individual block access is assumed to incur an extra disk arm movement, which can lead to an overestimatation of the disk overhead.

Reddy and Wijayaratne [1999] studied the fixed-block striping technique called *Constant Data Length* (CDL), and the *Block-constrained Constant Time Length* (BCTL) technique, where each round of stream data is stored on a single disk in multiples of a fixed block size. They concluded that the throughput improvement of BCTL (at large block sizes) is insignificant, which we believe is due to the particular block constraint they assumed. In the present article, we show that the best throughput of fixed-block striping technique can be improved by up to 50% using alternative striping policies.

In a study of single disk systems and constant bit-rate streams, *Group Periodic Multi-Round Prefetching* is described that retrieves multiple rounds worth of data in a single round [Triantafillou and Harizopoulos 1999]. In this article, we introduce *Group-Grain Striping* that is general enough to be applicable to disk arrays and variable bit-rate streams. It differs from the *Generalized Constant Data Length* described by Biersack et al. [1996] for single disk systems, where the retrieval of a fixed amount of data for a variable rate stream is distributed across a fixed number of rounds.

8.3 Disk Array Reliability

Most of the previous work on disk array fault-tolerance has been done in the context of traditional file server and transaction processing workloads [Bitton and Gray 1988; Hsiao and DeWitt 1990; Merchant and Yu 1995]. The related work from media server research is mostly focused on fault-tolerance techniques when striping constant bit-rate streams [Tobagi et al. 1993; Berson et al. 1995]. Disks are grouped into clusters, and data blocks from separate disks in each cluster are combined with a parity block to form parity groups. Ozden et al. [1996] propose reading ahead the data blocks of an entire parity group prior to their transmission to the client. When a data block cannot be accessed, it can be reconstructed using a parity block that is read instead. Alternatively, an entire parity group is retrieved each time a block cannot be accessed. Balanced incomplete block designs are used for constructing parity groups that keep the load of the disk array balanced [Ozden et al. 1996].

Gafsi and Biersack [2000] compare several performance measures of alternative data-mirroring and parity-based techniques for tolerating disk and node failures in distributed video servers. When entire data blocks of one disk are replicated on different disks, half of the total bandwidth of each disk is reserved for handling the disk failure case. Tewari et al. [1996] study parity-based redundancy techniques for tolerating disk and node failures in clustered servers. By distributing the parity blocks of an object on a random permutation of certain disks they can keep balanced the system load when a disk fails. Alternatively, Flynn and Tetzlaff [1996] replicate data blocks across nonintersecting permutations of disk groups. Instead, Birk [1997] examines selectively accessing parity blocks of video streams for better balancing the system load across multiple disks.

For failure recovery of variable bit-rate streams, Shenoy and Vin [2000] on the inherent redundancy in video streams rather than error-correcting codes. Alternatively, they reconstruct missing data from surrounding available blocks, at the cost of initial playback latency, or temporary disruption. Bolosky et al. [1996] decluster the block replicas of one disk across d other disks, which we didn't find significantly advantageous. The two load-balancing techniques that we examine have previously been found to improve performance when applied to traditional transaction processing workloads [Merchant and Yu 1995]. Mourad [1996] describes the doubly-striped disk mirroring technique that distributes replica blocks of one disk round-robin across the rest of the disks. The deterministic replica placement that we describe extends doubly-striped mirroring to variable bit-rate streams. Santos et al. [2000] use constant bit-rate streams to conclude that random replication can outperform disk striping with no replication. Instead, with variable bit-rate streams we found an advantage of deterministic replication over random replication that diminishes as the number of disks increases.

9. CONCLUSIONS

It is our thesis that building scalable and fault-tolerant media servers is feasible. We focus on the support of variable bit-rate video streams, because they reduce resource requirements when compared to constant bit-rate streams of equivalent quality. In the *Exedra* media server architecture that we propose, we minimize the probability of overloads and data losses within the server by keeping detailed account for the buffer space and the disk or network transfer delays corresponding to each accepted stream over time.

We examine and experiment with storage management issues using a prototype system that we built. Using the stride-based disk space allocation scheme, we keep bounded the estimated head movement overhead, while avoiding fragmentation inefficiencies. We handle separately the metadata management of each disk, which significantly simplifies the system structure, and enables support of heterogeneous disks. We allocate the memory buffers for each request contiguously in virtual memory to achieve high disk transfer bandwidth, and simplify system performance tuning. Our performance evaluation method is applicable to media servers with different transfer capacities and streams with various characteristics. In our experiments, we make sure that most of the system capacity is reached, while keeping the playback initiation time and the request rejection ratio acceptably low.

We formally specify the Fixed-Grain and Variable-Grain Striping policies, and generalize the latter to Group-Grain Striping. In our experiments, we demonstrate an almost linear increase in the supported number of concurrent users, as the number of disks increases. We show Group-Grain Striping to considerably outperform Fixed-Grain Striping when using streams of moderate or high variability. Finally, we introduce the Minimum Reservation Scheme to minimize the wasted throughput required for keeping accepted playbacks uninterrupted during a disk failure. At moderate disk array sizes, the throughput is less than 20% lower than what is achieved with no replication.

ACKNOWLEDGMENTS

We are thankful to the anonymous reviewers for the constructive comments that helped us improve the clarity and presentation of the article.

REFERENCES

ALMEIDA, J. M., KRUEGER, J., EAGER, D. L., AND VERNON, M. K. 2001. Analysis of educational media server workloads. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video* (Port Jefferson, NY). 21–30.

Alvarez, G. A., Borowsky, E., Go, S., Romer, T. H., Becker-Szendy, R., Golding, R., Merchant, A., Spasojevic, M., Veitch, A., and Wilkes, J. 2001. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Trans. Comput. Syst. 19*, 4, 483–518.

Anastasiadis, S. V. 2001. Scalable support for variable bit-rate streams in a continuous media server. Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, ON, Canada.

Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. 2001a. Disk striping scalability in the exedra media server. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference* (San Jose, CA). ACM, New York, 175–189.

Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. 2001b. Modular and efficient resource management in the exedra media server. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems* (San Francisco, CA). 25–36.

Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. 2002. Maximizing throughput in replicated disk striping of variable bit-rate streams. In *Proceedings of the USENIX Annual Technical Conference* (Monterey, CA). 191–204.

Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. 2005. Shared-buffer smoothing of variable bit-rate streams. *Perf. Eval. 59*, 1 (Jan.), 47–72.

Berson, S., Golubchik, L., and Muntz, R. R. 1995. Fault tolerant design of multimedia servers. In *Proceedings of the ACM SIGMOD Conference* (San Jose, CA), ACM, New York, 364–375.

Biersack, E., Thiesse, F., and Bernhardt, C. 1996. Constant data length retrieval for video servers with variable bit rate streams. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems* (Hiroshima, Japan), IEEE Computer Society Press, Los Alamitos, CA, 151–155.

Birk, Y. 1997. Random raids with selective exploitation of redundancy for high performance video servers. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video* (Zushi, Japan). 13–23.

Birrel, A. D. and Needham, R. M. 1980. A universal file server. *IEEE Trans. Softw. Eng. 6*, 5 (Sept.), 450–453.

Bitton, D. and Gray, J. 1988. Disk shadowing. In *Proceedings of the Very Large Data Base Conference* (Los Angeles, CA). 331–338.

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. 1996. The tiger video fileserver. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video* (Zushi, Japan). 97–104.

Bolosky, W. J., Fitzgerald, R. P., and Douceur, J. R. 1997. Distributed schedule management in the tiger video fileserver. In *Proceedings of the ACM Symposium on Operating Systems Principles* (Saint-Malo, France). 212–223.

Bradshaw, M. K., Wang, B., Sen, S., Gao, L., Kurose, J., Shenoy, P., and Towsley, D. 2003. Periodic broadcast and patching services—implementation, measurement, and analysis in an internet streaming video testbed. *Multimed. Syst. 9*, 1 (July), 78–93.

Buddhikot, M. M. and Parulkar, G. M. 1995. Efficient data layout, scheduling and playout control in mars. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video* (Durham, NH). 318–329.

Cabrera, L.-F. and Long, D. D. E. 1991. Swift: Using distributed disk striping to provide high I/O data rates. *Comput. Syst. 4*, 4, 405–436.

Chang, E. and Zakhor, A. 1994. Scalable video data placement on parallel disk arrays. In *Proceedings of the IS&T/SPIE International Symposium on Electronic Imaging: Image and Video Databases* (San Jose, CA). 208–221.

Chang, E. and Zakhor, A. 1996. Cost analyses for VBR video servers. *IEEE Multimed.* 56–71.

Chang, E. and Zakhor, A. 1997. Disk-based storage for scalable video. *IEEE Trans. Circ. Syst. Video Tech. 5* (Oct.), 758–770.

Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. 1994. Raid: High-performance, reliable secondary storage. *ACM Comput. Surv. 26*, 2 (June), 145–185.

Clark, T. 1999. *Designing Storage Area Networks*. Addison-Wesley, Reading, MA.

Eager, D. L., Vernon, M. K., and Zahorjan, J. 2001. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Trans. Knowl. Data Eng. 13*, 5 (Sept-Oct), 742–757.

Flynn, R. and Tetzlaff, W. 1996. Disk striping and block replication algorithms for video file servers. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems* (Hiroshima, Japan), IEEE Computer Society Press, Los Alamitos, CA, 590–597.

Gafsi, J. and Biersack, E. W. 2000. Modeling and performance comparison of reliability strategies for distributed video servers. *IEEE Trans. Parall. Distrib. Syst. 11*, 4 (Apr.), 412–430.

Ganger, G. R., Worthington, B. L., and Patt, Y. N. 1999. 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, MI. Dec.

Garofalakis, M. N., Ioannidis, Y. E., and Ozden, B. 1998. Resource scheduling for composite multimedia objects. In *Proceedings of the Very Large Data Bases Conference* (New York, NY). 74–85.

Gray, J. and Shenoy, P. 2000. Rules of thumb in data engineering. In *Proceedings of the IEEE International Conference on Data Engineering* (San Diego, CA). IEEE Computer Society Press, Los Alamitos, CA, 3–10.

Gringeri, S., Shuaib, K., Egorov, R., Lewis, A., Khasnabish, B., and Basch, B. 1998. Traffic shaping, bandwidth allocation, and quality assessment for MPEG video distribution over broadband networks. *IEEE Netw.* 6 (Nov./Dec.), 94–107.

Grochowski, E. and Halem, R. D. 2003. Technological impact of magnetic hard disk drives on storage systems. *IBM Syst. J. 42*, 2, 338–346.

Haskin, R. L. and Schmuck, F. B. 1996. The tiger shark file system. In *Proceedings of the IEEE COMPCON* (Santa Clara, CA). IEEE Computer Society Press, Los Alamitos, CA, 226–231.

Hsiao, H.-I. and DeWitt, D. J. 1990. Chained declustering: A new availability strategy for multiprocessor database machines. In *Proceedings of the IEEE International Conference on Data Engineering* (Los Angeles, CA). IEEE Computer Society Press, Los Alamitos, CA, 456–465.

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

Lakshman, T. V., Ortega, A., and Reibman, A. R. 1998. VBR video: Tradeoffs and potentials. *Proce. IEEE 86*, 5 (May), 952–973.

Makaroff, D., Hutchinson, N., and Neufeld, G. 1997. An evaluation of VBR disk admission algorithms for continuous media file servers. In *Proceedings of the ACM International Conference on Multimedia* (Seattle, WA). ACM, New York, 143–154.

Martin, C., Narayanan, P. S., Ozden, B., Rastogi, R., and Silberschatz, A. 1996. The Fellini multimedia storage system. In *Multimedia Information Storage and Management*, S.M.Chung, Ed. Kluwer Academic Publishers, Boston, MA.

McKusick, M. K., Joy, W. N., Leffler, S., and Fabry, R. S. 1984. A fast file system for UNIX. *ACM Trans. Comput. Syst. 2*, 3 (Aug.), 181–197.

McVoy, L. and Kleiman, S. R. 1991. Extent-like performance from a UNIX file system. In *Proceedings of the USENIX Winter Technical Conference* (Dallas, TX). 33–43.

Merchant, A. and Yu, P. S. 1995. Analytic modeling and comparisons of striping strategies for replicated disk arrays. *IEEE Trans. Comput. 44*, 3 (Mar.).

Microsoft. 2003. Choosing an encoding method, Windows Media 9 series, development network library. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmform/htm/choosinganencodingmethod.asp.

Mourad, A. 1996. Doubly-striped disk mirroring: Reliable storage for video servers. *Multimed. Tools Applic. 2*, 273–297.

Muntz, R., Santos, J. R., and Berson, S. 1998. A parallel disk storage system for real-time multimedia applications. *Int. J. Intelligent Systems 13*, 12 (Dec.), 1137–1174.

Nagle, D., Serenyi, D., and Matthews, A. 2004. The panasas activescale storage cluster—Delivering scalable high bandwidth storage. In *Proceedings of the ACM/IEEE Conference on Supercomputing* (Pittsburgh, PA). ACM, New York, 53.

Neufeld, G., Makaroff, D., and Hutchinson, N. 1996. Design of a variable bit rate continuous media file server for an atm network. In *Proceedings of the IS&T/SPIE Multimedia Computing and Networking Conference* (San Jose, CA), 370–380.

Ng, S. W. 1998. Advances in disk technology: Performance issues. *Computer 31*, 15 (May), 75–81.

Nieuwejaar, N., Kotz, D., Purakayastha, A., Ellis, C. S., and Best, M. L. 1996. File-access characteristics of parallel scientific workloads. *IEEE Trans. Paral. Distrib. Syst. 7*, 10 (Oct.), 1075–1089.

Ozden, B., Rastogi, R., and Silberschatz, A. 1996. Disk striping in video server environments. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems* (Hiroshima, Japan). IEEE Computer Society Press, Los Alamitos, CA, 580–589.

Reddy, A. L. N. and Wijayaratne, R. 1999. Techniques for improving the throughput of vbr streams. In *Proceedings of the IEEE ACM/SPIE Multimedia Computing and Networking Conference* (San Jose, CA). 216–227.

Ruemmler, C. and Wilkes, J. 1994. An introduction to disk drive modeling. *Computer 27*, 3 (Mar.), 17–28.

Santos, J. R., Muntz, R. R., and Ribeiro-Neto, B. 2000. Comparing random data allocation and data striping in multimedia servers. In *Proceedings of the ACM SIGMETRICS* (Santa Clara, CA). ACM, New York, 44–55.

Sen, P., Maglaris, B., Rikli, N., and Anastassiou, D. 1989. Models for packet switching of variable bit-rate video sources. *IEEE J. Selec. Areas Commun. 7*, 5 (June), 865–869.

Sen, S., Dey, J., Kurose, J., Stankovic, J., and Towsley, D. 1997. Streaming CBR transmission of VBR stored video. In *Proceedings of the SPIE Symposium on Voice, Video and Data Communications* (Dallas, TX). 26–36.

Shenoy, P. J., Goyal, P., Rao, S. S., and Vin, H. M. 1998. Symphony: An integrated multimedia file system. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking Conference* (San Jose, CA). ACM, New York, 124–138.

Shenoy, P. J. and Vin, H. M. 1999. Efficient striping techniques for multimedia file servers. *Perf. Eval. 38*, 3–4 (Oct.), 175–199.

Shenoy, P. J. and Vin, H. M. 2000. Failure recovery algorithms for multimedia servers. *Multimed. Syst. J. 8*, 1 (Jan.), 1–19.

Tewari, R., Dias, D. M., Mukherjee, R., and Vin, H. M. 1996. High availability in clustered multimedia servers. In *Proceedings of the IEEE International Conference on Data Engineering* (New Orleans, LA). IEEE Computer Society Press, Los Alamitos, CA, 336–342.

Thirumalai, K., Paris, J.-F., and Long, D. D. E. 2003. Tabbycat: an inexpensive scalable server for video-on-demand. In *Proceedings of the IEEE International Conference on Communications*. (Anchorage, AK). IEEE Computer Society Press, Los Alamitos, CA, 896–900.

Tobagi, F. A., Pang, J., Baird, R., and Gang, M. 1993. Streaming raid—A disk array management system for video files. In *ACM International Conference on Multimedia* (Anaheim, CA). ACM, New York, 393–400.

Triantafillou, P. and Harizopoulos, S. 1999. Prefetching into smart-disk caches for high performance media server. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems* (Florence, Italy). IEEE Computer Society Press, Los Alamitos, CA, 800–805.

Verbiest, W., Pinnoo, L., and Voeten, B. 1988. The impact of the ATM concept on video coding. *IEEE J. Sel. Areas Commun. 6*, 9 (Dec.), 1623–1632.

Received January 2005; revised June 2005; accepted July 2005