Maximizing Throughput in Replicated Disk Striping of Variable Bit-Rate Streams

Stergios V. Anastasiadis*
Department of Computer Science
Duke University
stergios@cs.duke.edu

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

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

Abstract

In a system offering on-demand real-time streaming of media files, data striping across an array of disks can improve load balancing, allowing higher disk utilization and increased system throughput. However, it can also cause complete service disruption in the case of a disk failure. Reliability can be improved by adding data redundancy and reserving extra disk bandwidth during normal operation. In this paper, we are interested in providing fault-tolerance for media servers that support variable bit-rate encoding formats. Higher compression efficiency with respect to constant bit-rate encoding can significantly reduce per-user resource requirements, at the cost of increased resource management complexity. For the first time, the interaction between storage system fault-tolerance and variable bit-rate streaming with deterministic QoS guarantees is investigated. We implement into a prototype server and experimentally evaluate, using detailed simulated disk models, alternative data replication techniques and disk bandwidth reservation schemes. We show that with the minimum reservation scheme introduced here, single disk failures can be tolerated at a cost of less than 20% reduced throughput during normal operation, even for a disk array of moderate size. We also examine the benefit from load balancing techniques proposed for traditional storage systems and find only limited improvement in the measured throughput.

1 Introduction

Striping media files across multiple disks has the advantage of keeping the disks implicitly load-balanced. With several concurrent sessions of playback asynchronously started, different parts of each file are accessed from different disks. As a result it is no longer necessary to replicate media files according to their popularity, which leads to lower resource requirements and system administration cost. The disadvantage of disk striping is decreased system reliability because media files are left partially inaccessible when one or more disks fail. This causes service disruption to all the users served by the disk array at the moment of the failure. In contrast, when an entire file is stored on a single disk, only those users accessing files on the failed disk are negatively affected.

Previous work has addressed the general problem of disk array reliability by using data redundancy techniques to allow recovery of inaccessible data [12]. Some of these techniques have been successfully extended for handling the case of striped media files as well [6, 25]. However, all the known analytical and experimental work on this subject is either limited to streams of constant bit rates (CBR), or assumes stochastic admission control [27, 29].

Variable bit-rate (VBR) encoding of video can considerably reduce the size of the generated media files when compared to constant bit-rate encoding of equivalent perceptual quality [18, 22]. In addition, knowledge about the resource requirements of stored streams during transmission can be leveraged for better predicting access delays, and offering deterministic QoS guarantees [11]. Although striping of VBR streams has been previously studied, it remains unclear whether increased reliability can be provided with deterministic QoS guarantees in cost-effective ways.

Variability in the resource requirements over time makes efficient disk space allocation combined with access delay predictability a challenging task. Data striping across multiple disks with sufficient redun-

*Part of this research has been done, while the co-author was PhD student at the University of Toronto. Financial support from the Natural Sciences and Engineering Research Council of Canada is gratefully acknowledged.

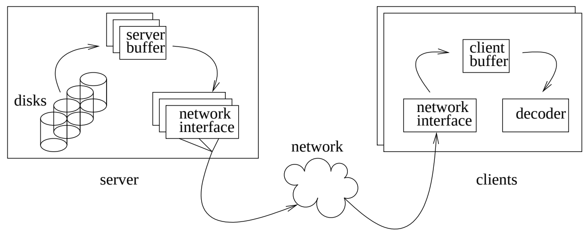

Figure 1: Compressed video streams are stored across multiple disks of the media server. Multiple clients (or proxies) can connect and start playback sessions via separate network links.

dancy to tolerate failures further aggravates the problem due to the need for keeping balanced the storage space and bandwidth requirements across the disk array, under both normal-operation and failed-disk conditions. In the present paper, we describe a number of data redundancy and bandwidth reservation schemes that can tolerate single disk failures without service interruption. We experimentally evaluate the cost of increased reliability in terms of reduced throughput during the normal operation of the system using a video server prototype implementation and MPEG-2 streams. We also investigate the achieved disk bandwidth utilization during normal and failed-disk operation. Additionally, we examine the extra benefit from retrieving data replicas stored on the least loaded disks, and from fragmenting data replicas across multiple disks.

The rest of this paper is organized as follows. In Section 2, we describe basic assumptions and architectural decisions of our system. In Sections 3, 4 and 5, we introduce alternative policies for replicating stream data and reserving disk bandwidth for improved reliability. In Section 6, we briefly present our prototype implementation and the experimentation environment that we use. In Section 7, we compare the performance of different replication techniques under alternative bandwidth reservation schemes and load balancing enhancements. In Section 8, we discuss possible improvements and extensions, and in Section 9 we summarize our conclusions.

2 System Architecture

In the present section, we describe the system architecture along with important resource management and reservation techniques that we use [2].

2.1 Overview

The operation of our media server is typical in current system designs. Client devices submit playback requests concurrently to the server. The system is assumed to operate according to the server-push model. When a playback session starts, the server periodically sends data to the client until either the end of the stream is reached, or the client explicitly requests suspension of the playback. Data transfers occur in rounds of fixed duration T_{round}. In each round, an appropriate amount of data is retrieved from the disks into a set of server buffers reserved for each active client. Concurrently, data are sent from the server buffers to the client through the network interfaces (Figure 1). The amount of stream data periodically sent to the client is determined by the decoding frame rate of the stream, the buffering constraints of the receiver, and the resource management policy of the network. As a minimum requirement, the client should receive in each round the amount of data that will be needed by the decoder during the next round.

The streams are compressed according to any encoding scheme that supports constant quality quantization parameters and variable bit rates. Playback requests arriving from the clients are initially directed to an admission control module, where it is determined whether enough resources exist to activate the requested playback session either immediately or within a limited number of rounds. A schedule database maintains for each stream information on how much data needs to be accessed from each disk in any given round, the amount of server buffer space required, and how much data needs to be transferred to the client. This scheduling information is generated when the media stream is first stored, and is used for both admission control and transfer of data during playback.

2.2 Stride-Based Disk Space Allocation

In our experiments, we use a method called *stride-based allocation* for allocating disk space [3]. In stride-based allocation, disk space is allocated in large, fixed-sized chunks called *strides*. The strides are chosen larger than the maximum stream request size per disk during a round. This size is known a priori, since stored streams are accessed sequentially according to a predefined (generally variable) rate. A stride may contain data of more than one round. When a stream is retrieved, only the requested amount of data is fetched to memory during a round, and not the entire stride.

Stride-based allocation eliminates external fragmentation due to the fixed-size strides. Internal fragmentation remains negligible because of the large size of the streams relative to strides. Another advantage of stride-based allocation is that it sets an upper-bound on the estimated disk access overhead during retrieval. Since the size of a stream request never exceeds the stride size during a round, at most two partial stride accesses will be required to serve the request of a round on each disk.

2.3 Reservation of Server Resources

A mathematical abstraction of the resource requirements is necessary for scheduling streams. We consider a system with D functionally equivalent disks. Data of each stream are stored as sequences of strides on each disk. Each stride comprises an integer number of consecutive logical blocks with fixed size B_l. The logical block size is a multiple of the physical sector size B_p of the disk. Both the disk transfer requests and the memory buffer reservations are specified in multiples of the block size B_l. The *Disk Striping Sequence* $\mathbf{S_d}$ of length L_d determines the amount of data $S_d(i,k)$, $0 \leq i \leq L_d - 1$, that are retrieved from disk k, $0 \leq k \leq D-1$, in round i.

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

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

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

$$
\begin{aligned}
T_{disk}^j(i,k) = & 2 \cdot (T_{trackSeek} + T_{avgRot}) \\
& + S_d^j(i - l_j, k)/R_{disk}
\end{aligned}
$$

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

2.4 Variable-Grain Striping

For striping stream data across multiple disks, we use the *Variable-Grain Striping* policy. Data consumed by a client during a playback round is stored on (and accessed from) a single disk, while different disks are visited in round-robin fashion during successive rounds of a stream playback. When compared against alternative striping techniques, variable-grain striping demonstrates significant performance advantage due to i) reduced disk access overhead from accessing at most one disk per stream in a round, and ii) improved disk bandwidth utilization by statistically multiplexing I/O requests of different sizes from concurrently served streams [2].

3 Data Redundancy Policies

Due to the large number of components involved, it is necessary to assume device failures during the lifetime of a typical commercial server installation. With the estimated Mean Time To Failure of a modern disk at about $MTTF_{disk} = 1,200,000$ hours, combining $D = 1024$ disks results in $MTTF_{array} = \frac{MTTF_{disk}}{D} = 49$ days, assuming failure independence among different devices [26].[1] A typical system with 1024 drives could support about $51,000$ concurrent playbacks of 5 Mbit/s average bit rate each.[2] 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 [9, 19].

In order to provide higher system reliability, data redundancy techniques can be used. However, a practical solution should minimize the extra computa-

[1] The calculation assumes Seagate Cheetah 18GB Ultra160 SCSI disks with 31 MB/s formatted minimum internal transfer rate [28].

[2] These numbers are realistic in light of the popularity of similar services. For example, the number of cable television subscribers in 1998 was estimated to exceed 65 million in the US alone [13].

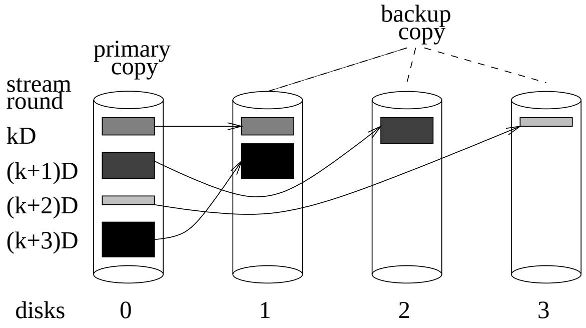

Figure 2: **Deterministic Replica Placement.** Data of a media file stored consecutively on disk 0 and retrieved during different playback rounds are replicated round-robin across the other disks. The primary data of the other disks are replicated in a similar way.

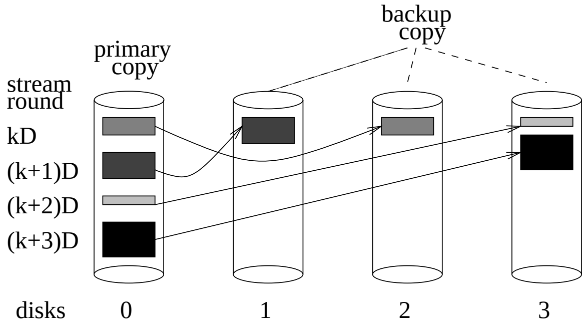

Figure 3: **Random Replica Placement.** Data of a media file stored consecutively on disk 0 and retrieved during different playback rounds are replicated on randomly chosen disks 1 to 3. The primary data copies of disks 1 to 3 are replicated in a similar way.

tion, storage and bandwidth requirements with respect to the non-redundant case. The present study focuses on single disk failures which are the most common. Multiple disk failures are less likely to occur simultaneously [12], and possible ways for handling them are described briefly later.

In the past, several parity-based techniques have been proposed that store error-correcting code for the data blocks of different disks [12]. When a disk fails, redundant information available on the surviving disks is used to recover the missing data blocks. Parity-based techniques trade extra disk bandwidth or memory buffer requirements for reduced storage space. Since disk storage space currently has the lowest cost of the three resources, it has been suggested that replication rather than parity is the preferred technique for tolerating disk failures [10, 17]. Furthermore, implementation of parity-based data recovery in a distributed architecture requires additional data traffic among different nodes [9]. This can introduce significant extra complexity and resource requirements in terms of network bandwidth and buffer space. For the above reasons, we do not consider parity-based techniques any further 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*. Although the two copies can be used symmetrically, distinct placement policies can be applied to each of them as we describe shortly. When one disk fails, its data remain available by retrieving their backup replicas from the rest of the disks. The required storage space is roughly doubled and the needed bandwidth from each disk is at most twice that of the non-redundant case. The backup replica of each data block can be stored in its entirety on a different disk, which requires only one access in the case of failure and minimizes the access overhead. The alternative of declustering a backup replica across multiple disks can potentially better balance the extra access load, but incurs the additional overhead of multiple accesses in the case of a disk failure.

3.1 Deterministic Replica Placement

In previous work, we have demonstrated that variable-grain striping of media files leads to equally utilized disks under sequential playback workloads [2]. 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, each disk is accessed every D rounds, where D is the total number of disks. In order to preserve the load-balancing property when a disk fails, data of a media file stored consecutively on each disk 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 this mirroring approach *Deterministic Replica Placement*. In Figure 2, for example, disk 0 is shown to store stream data requested during rounds $k\cdot D$, $(k+1)\cdot D$, $(k+2)\cdot D$ and $(k+3)\cdot D$. The respective replicas are distributed round-robin among disks 1, 2 and 3.

3.2 Random Replica Placement

Intuitively, having replicas of one disk's primary data distributed round-robin across the rest of the disks can keep the surviving disks equally utilized when one disk fails. An alternative replication approach

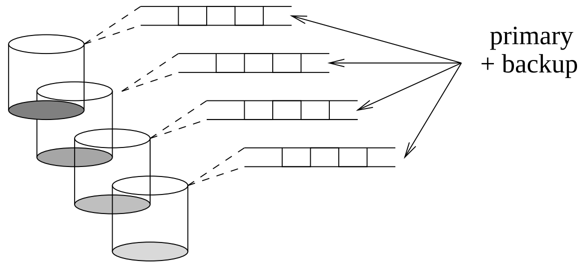

Figure 4: **Mirroring Reservation.** For each disk, there is a separate vector indexed by round number that accumulates the total estimated access time for retrieving primary and backup data in each round.

would use some pseudo-random sequence for specifying the disks that store the backup copies of one disk's primary data. An obvious constraint is that primary and backup copies are stored on different devices. The unit of replication corresponds to the data of a media file requested by a client in one round of playback. We call this mirroring technique *Random Replica Placement*.

An example is shown in Figure 3, where backup copies of data requested in rounds $k \cdot D, \ldots, (k+3) \cdot D$ are randomly placed on disks 1 to 3. It has been previously suggested that random placement of primary and backup replicas across different disks is applicable to a wider range of workload types and can outperform striping policies with round-robin placement [27]. In a later section, we examine this argument in the particular case of variable bit-rate streams by comparing it against deterministic replica placement.

4 Disk Bandwidth Reservation

Our goal in this section is to allocate resources in such a manner that service to accepted requests will not be interrupted during (single) disk failures. 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. Alternatively, when a disk fails and no extra bandwidth has been reserved, service will become unavailable for a number of active users with aggregate bandwidth requirements no less than the transfer capacity of one disk, assuming that data have been replicated as described previously.

The net benefit from uninterrupted service during disk failures is equal to the difference between two measures. One is the cost of having users frustrated

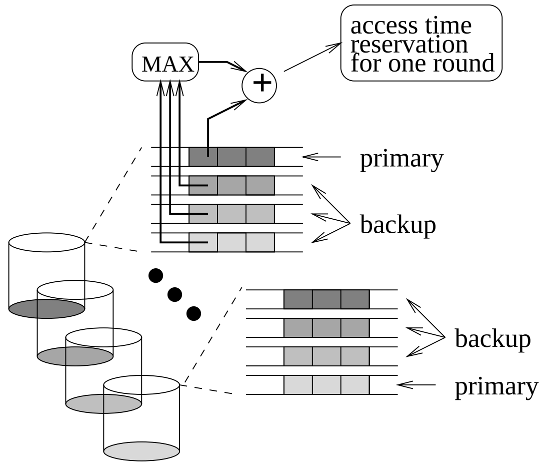

Figure 5: **Minimum Reservation.** For each disk, we maintain D separate vectors indexed by round number. One of them accumulates access delays for retrieving primary data. The remaining $D-1$ vectors accumulate access delays for retrieving backup replicas that correspond to primary data stored on each 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.

due to interrupted service from a failed disk. Its quantification would require determining the minimum number of users negatively affected when a disk fails. Detailed study of this issue is left for future work. The other measure is the cost of rejecting user requests due to additionally reserved disk bandwidth during normal operation. In the rest of this paper, we describe alternative approaches for reserving disk bandwidth (or equivalently access time) and improving reliability in media servers that support variable bit-rate streams. Subsequently, we experimentally evaluate the actual cost of these approaches in terms of reduced system throughput during normal system operation.

In what we call *Mirroring Reservation*, disk bandwidth is reserved for both the primary and backup replicas of a media file during its playback (Figure 4). At first glance, this seems to be a reasonable approach for guaranteeing timely access to backup replicas during a single disk failure. However, when compared to the non-redundant case, it doubles the bandwidth requirements of each stream and halves the maximum system throughput, assuming disk bandwidth is the bottleneck resource in the system. Indeed, we would prefer that the load normally handled by a failed disk is equally divided among the $D-1$ surviving disks. Thus, tolerating one disk failure should require that the extra

Replica Placement	Disk Bandwidth Reservation	
	Mirroring	Minimum
Deterministic	$\surd$	$\surd$
Random	$\surd$	$\surd$

Table 1: The replica placement policies can be orthogonally combined with the disk bandwidth reservation schemes.

bandwidth reserved on each disk be equal to $\frac{1}{D-1}$ its bandwidth capacity. Instead, mirroring reservation reserves extra bandwidth on each disk equal to half its bandwidth capacity.

Essentially, it is wasteful to reserve on a disk extra bandwidth for accessing backup replicas of primary data stored on more than one other disk. When a disk fails, we only need an estimate of the additional access load incurred on every surviving disk. In order to know that, the access time of the backup replicas stored on one disk can be accumulated separately for every disk that stores the corresponding primary data. Then, the additional access time that has to be reserved on a disk in each round is equal to the maximum time required for retrieving backup replicas for another disk that has failed. The maximum across every other disk is reserved, since we don't know in advance which other disk is going to fail.

For each disk, our implementation maintains D vectors, indexed by round number of system operation. One of the vectors 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 required in each round. We refer to this more efficient scheme as *Minimum Reservation*. An example with four disks is illustrated in Figure 5. In a later section, we discuss ways for limiting the additional computational and memory requirements of this approach.

The two disk bandwidth reservation schemes that we just described can be orthogonally combined with the two replica placement policies that we introduced previously, as it is shown in Table 1.

5 Load Balancing Enhancements

The load of a failed disk could possibly be shared more fairly among the surviving disks if each backup replica was declustered across multiple devices. Therefore, we break each backup replica into blocks of fixed size B_d, and we call this load balancing technique *Backup Replica Declustering*. We choose B_d to be an integer multiple of the logical block size B_l, introduced previously. We allow the last fragment of the replica to have a size that is smaller than B_d but integer multiple of B_l.

The backup replica blocks corresponding to the primary data of each disk are distributed either round-robin or pseudo-randomly across the rest of the disks, depending on whether deterministic or random replica placement is used. In the case of random replica placement, we improve block distribution by avoiding reusing the same disk for storing multiple replica blocks of the same file in one round unless we are running out of disks. When multiple blocks of size B_d are retrieved from a disk during one round of a file playback, the minimum required number of read requests is submitted to the disk, instead of one per block.

Alternatively, during normal operation we could take 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 could be based on access time estimations available through resource reservations that are made during admission control. We use the term *Dynamic Balancing* for this technique. 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.

Both these two techniques have previously been found to improve performance when applied to traditional transaction processing workloads [23]. Replica declustering has also been tried with constant bit-rate stream playback [9, 15]. Due to the potential for load imbalance and reduced device utilization introduced by variable bit-rate streams, we investigate the benefit from load-balancing techniques in that context.

6 Experimentation Environment

In order to keep our presentation complete, we briefly describe here important aspects of our prototype implementation, the characteristics of our benchmarks, and the performance evaluation method that we use for our experiments [2].

6.1 Prototype Overview

We have designed and built a media server experimentation platform, in order to evaluate the resource requirements of alternative disk replication policies [2]. The different modules are implemented in about 17,000 lines of C++/ Pthreads code on AIX4.1. The

Seagate Cheetah ST-34501N	
Data Bytes per Drive	4.55 GB
Average Sectors per Track	170
Data Cylinders	6,526
Data Surfaces	8
Zones	7
Buffer Size	512KB
Track to Track Seek(read/write)	0.98/1.24 msec
Maximum Seek(read/write)	18.2/19.2 msec
Average Rotational Latency	2.99 msec
Internal Transfer Rate	
Inner Zone to Outer Zone Burst	122 to 177 Mbit/s
Inner Zone to Outer Zone Sustained	11.3 to 16.8 MB/s

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

code is linked either to the University of Michigan DiskSim disk simulation package [16], which incorporates advanced features of modern disks such as on-disk cache and zones for simulated disk access time measurements, or to hardware disks through their raw device interfaces. The indexing metadata are stored as regular Unix files, and during operation are kept in main memory.

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

With appropriate configuration parameters, the system can operate at different levels of detail. In *Admission Control* mode, the system receives playback requests, does admission control and resource reservation, but no actual data transfers take place. In *Simulated Disk* mode, most modules become functional and disk request processing is simulated using the specified DiskSim disk array. Techniques for file system simulation similar to those previously proposed are used for integrating the simulated disks with our media server prototype [31]. There is also the *Full Operation* mode, where the system accesses hardware disks and transfers data to fixed client network addresses. For the experiments in the current study, we used both the Admission Control and the Simulated Disk Mode.

6.2 Performance Evaluation Method

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

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

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

For each playback request that arrives, the admission control module checks whether available resources exist for every round during playback. The test considers the data transfer requirements of the requested playback for every round and also the corresponding available disk transfer time, network transfer time and buffer space in the system. If the request cannot be initiated in the next round, the test is repeated for each round up to $\lceil \frac{1}{\lambda} \rceil$ rounds into the future, until the first round is found where the requested playback can be started with guaranteed sufficiency of resources. Checking $\lceil \frac{1}{\lambda} \rceil$ rounds into the future achieves most of the potential system capacity as was shown previously [2]. If not accepted, the request is rejected rather than being kept in a queue.

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

Table 3: We used six MPEG-2 video streams of 30 minutes duration each. The coefficient of variation shown in the last column changes according to the content type.

6.3 Experimentation Setup

We used six different VBR MPEG-2 streams of 30 minutes duration each. Every stream has 54,000 frames with a resolution of 720x480 and 24 bit color depth, 30 frames per second frequency, and a $IB^2PB^2 PB^2PB^2PB^2$ 15 frame Group of Pictures structure. The encoding hardware that we use generates bit rates between 1 Mbit/s and 9.6 Mbit/s. The statistical characteristics of the clips are given in Table 3. The coefficients of variation of bytes per round lie between 0.028 and 0.383, depending on the content type. In the *mixed* benchmark, the six different

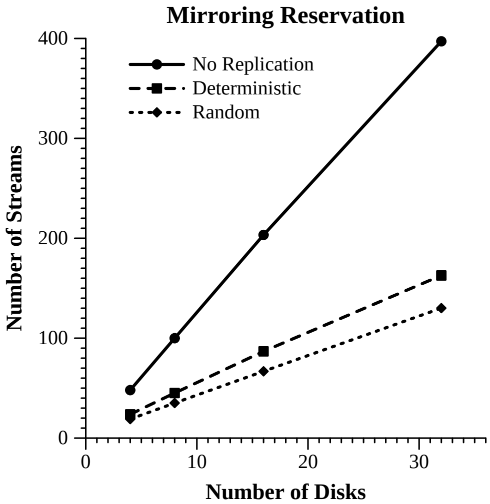

Figure 6: The mirroring reservation scheme reduces the number of streams supported by a factor of two compared to the no-replication case. Deterministic replica placement sustains an advantage of 25% or more relative to random replica placement under the mixed stream workload.

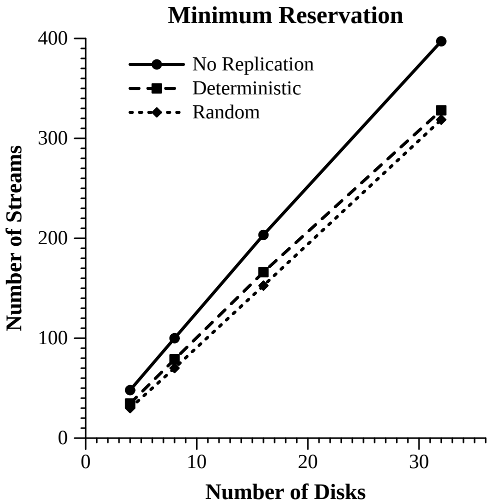

Figure 7: With minimum reservation and number of disks increasing from 4 to 32, the throughput advantage of deterministic over random replica placement drops from 15% to 3%. The corresponding throughput disadvantage of deterministic placement with respect to no replication drops from 28% to 17%.

streams are submitted round-robin. Where appropriate, experimental results from individual stream types are also shown.

The disks assumed in our experiments are Seagate Cheetah with ultra-wide SCSI interface and the features shown in Table 2. Such disks were state of the art about three years ago, and have all the basic architectural characteristics of today's high-end drives. The logical block size B_l was set to $16KB$ bytes, while the physical sector size B_p was equal to 512 bytes. The stride size B_s in the disk space allocation was set to 2 MB. The server memory is organized in buffers of fixed size $B_l = 16KB$ bytes each, with space of 64 MB for every extra disk. The available network bandwidth was assumed to be infinite, leaving contention for the network outside the scope of the current work.

In our experiments, the round time was set equal to one second. We found this round length to achieve most of the system capacity with tolerable initiation latency. This choice also facilitates comparison with previous work in which one second rounds were used. We used a warmup period of 3,000 rounds and calculated the average number of active streams from round 3,000 to round 9,000. The measurements were repeated until the half-length of the 95% confidence interval was within 5% of the estimated mean value of the number of active streams. The system load was fixed at $\rho = 80\%$, which allows the system to reach its capacity while keeping the playback startup latency limited [2].

7 Experimental Evaluation

We compare the data replication and bandwidth reservation techniques that we introduced with respect to the average number of active playback sessions that can be supported by the server. The objective is to make this number as high as possible. We provide supplementary performance intuition with statistics on reserved and utilized disk access time across different stream types and numbers of disks.

We start with a performance comparison between the deterministic and random replica placement policies under the mirroring reservation scheme. Subsequently, we examine the improvement to the two placement policies when minimum reservation is applied. We also investigate the benefit of dynamic balancing assuming that disk bandwidth in each round is reserved for only one data replica out of the two available. Finally, we consider declustering the backup replica of each stream across multiple disks and allocating bandwidth according to the minimum reservation scheme.

7.1 Replica Placement Comparison

We use the mixed stream workload to compare the performance of alternative replica placement policies under the mirroring reservation scheme (Figure 6). With the number of disks varying between 4 and 32, the measured throughput of replicated disk striping

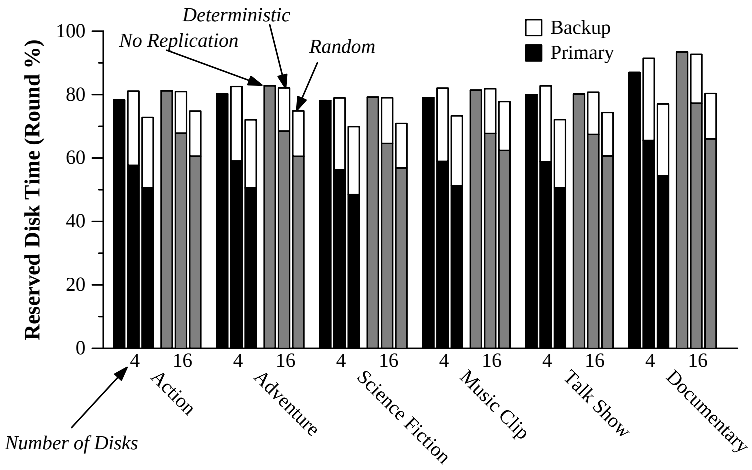

Figure 8: The disk time reserved in each round when individual stream types are used with minimum reservation. With deterministic replica placement, the disk time reserved for backup accesses drops from about 24% to 14% of the round length as the number of disks increases from 4 to 16. With random replica placement, the respective percentage drops from about 22% to 14%.

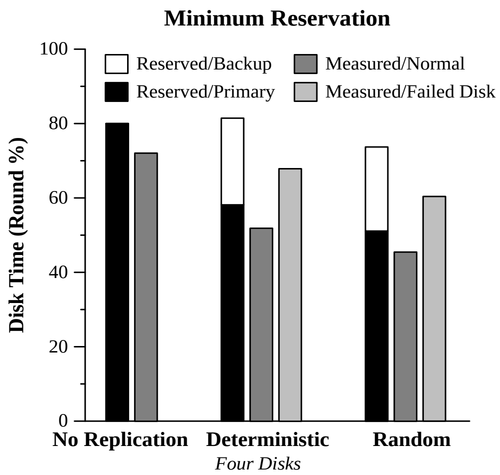

Figure 9: During normal operation, the average disk access time that is measured in each round remains within 6-8% below the time reserved for primary data accesses. When a disk fails, the measured access time is 13-14% lower than the total reserved.

is less than half of what is achieved with no replication. In addition, deterministic replica placement achieves a throughput advantage of 25% or more relative to random replica placement.

From measurements that we did (not shown here), we found that about half of the average disk time reserved in the replicated case is wasted for the possibility that the backup data will be retrieved. Furthermore, the access time reserved on each disk by random replica placement is about 15%-25% less than that of deterministic placement. Pseudo random choice of the disk that stores a backup replica does not completely eliminate the possibility of one disk storing more replicas than another, especially with small disk arrays. The probability of that occurring drops as the size of the disk array increases, though. However, deterministic placement is more consistent in fairly distributing the access load across the disk array devices.

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 see how this inefficiency is alleviated by the minimum reservation scheme in the following subsection.

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 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. This maximum requirement is calculated separately and is generally expected to be different for each disk in each round.

Figure 7 compares the throughput of the different replica placement policies under minimum reservation. At eight disks, the number of streams supported by deterministic placement is only 21% lower than that with no replication. This difference becomes 18% and 17%, respectively, with sixteen and thirty two disks. From the way that the disk bandwidth is allocated in the minimum reservation scheme, we would expect the total bandwidth that remains unutilized during normal operation to be equal to the bandwidth capacity of one disk. Therefore, the percentage of unused throughput with respect to the non-replicated case should be decreasing proportionally with the number of disks in the system. For example, ideally with 16 disks only the $\frac{1}{16} = 6.25\%$ of the total disk bandwidth should remain unused during normal operation.

However, in practice, the percentage of the total unused bandwidth of each disk does not change proportionally with the number of disks (Figures 8). This effect can be explained by the $MAX()$ operator that is applied over the estimated time for accessing the backup replicas of different disks, in combination with the relatively large size (more than half megabyte on average) of the data retrieved for a stream in each round. We explore later the potential improvement from declustering backup replicas across multiple disks.

Additionally, the difference between deterministic and random replica placement becomes less significant than the statistical uncertainty at thirty two disks. Not surprisingly, deterministic placement maintains a clear advantage (of about 15%) for smaller disk array sizes due to the more regular way of distributing the backup replica access load across the different devices. These observations are consistent with the average access time reserved on each disk across different stream types and disk array sizes shown in Figure 8.

In Figure 9, we show the measured disk busy time. Under normal disk operation, we observe that deterministic placement keeps the disks busy an amount of time that is 6% lower than what is reserved for primary data.[3] When one disk fails, the remaining disks are busy 14% time less than the total reserved. With random replica placement, the corresponding difference becomes 13% of the round length. 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 disk array size equal to four, about one third of each disk's bandwidth has to be reserved for the case that one disk fails, and this fraction drops as the disk array size increases to sixteen (Figure 8).

It is interesting that, when the reserved backup access time is put into use due to a disk failure, 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 it is due to the MAX() operator that we apply to the backup access times corresponding to different disks in each round. This reserves enough access time to ensure uninterrupted system operation for any particular failed disk. However, the reported measured time is taken when the disk 0 is assumed inaccessible (Figure 2). 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 operation more robust, and guards it against nondeterministic factors, such as the system bus contention due to network transfers, not included in the previous measurements.

[3] In previous work [3], we reported similar differences between the average reserved time and the access time measured when using the hardware disks of Table 2, instead of their simulated models.

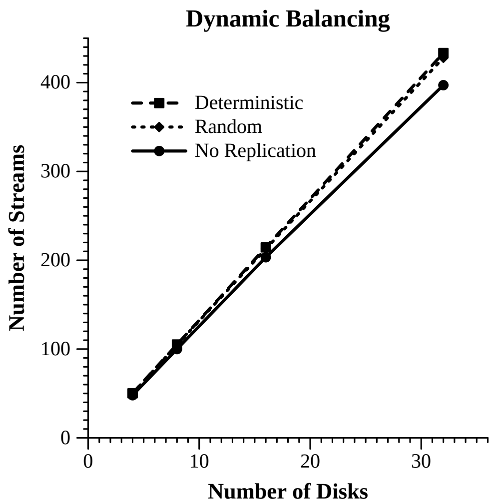

Figure 10: During normal operation, accessing the replica of the least loaded disk improves the throughput by about 5-10% with respect to the non-replicated case. The gain tends to increase as the disk array size increases.

7.3 Improving Load Balancing

With multiple data replicas available, better load balancing can be achieved by choosing the replica stored on the least loaded disk during admission control. In this case, we leverage data replication for improving the system throughput, rather than tolerating disk failures. We use the accumulated disk access time estimations in order to choose the least loaded disk. Making this choice based on actual measurement of the disk access load is not a feasible alternative, due to the round-based operation that prevents access load propagation from one round to the next.

From Figure 10, we see that, when this load balancing scheme is used, both replica placement policies can support 5-10% more streams than the non-replicated case. The difference between the two placement policies is statistically insignificant, however, since the gain from the dynamic replica access exceeds the improved load balancing of deterministic placement. Determining during admission control which disk will be used for each data access is a reasonable policy for removing hot spots in the disk array. However, under sequential workloads the different disks are equally utilized already, and only a limited additional performance benefit can be accrued with the above policy.

In Figure 11, we consider the case of declustering backup replicas across multiple disks using a fixed

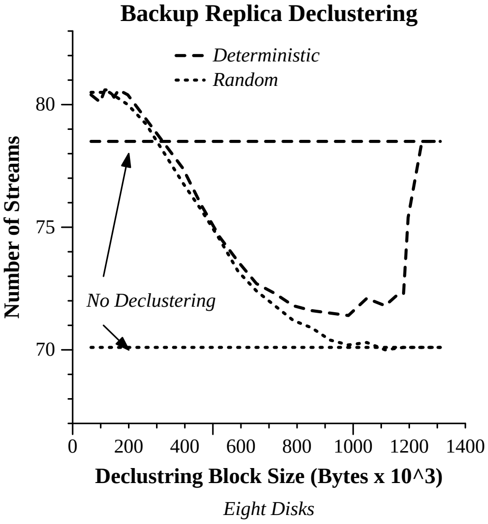

Figure 11: Each backup replica is divided into blocks of the specified size and distributed across multiple disks. There is a minor gain from applying backup replica declustering to deterministic placement, while the best throughput achieved with random placement approaches that of deterministic. The two horizontal lines correspond to the throughput achieved when no declustering is applied to the two replica placement policies, respectively.

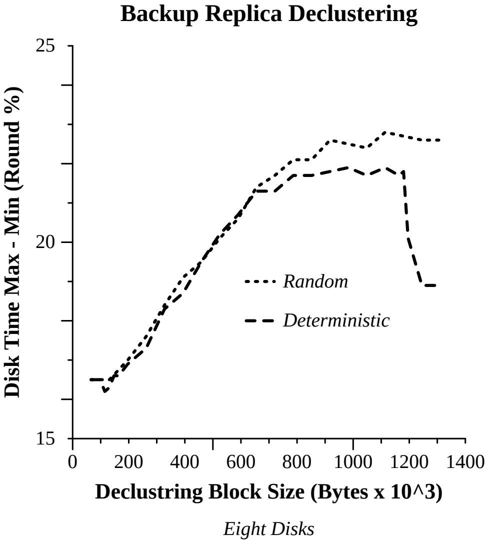

Figure 12: The average difference (expressed as round length percentage) in the reserved access times between the most and least loaded disk in each round varies according to the declustering block size. This difference can be interpreted as one measure of load imbalance within each round. From the shape, we see that it has a significant effect to the achieved throughput shown in Figure 11.

block size B_d. This approach is expected to let the failed-disk load be more fairly shared among the surviving disks. With small block sizes, better load balancing leads into some limited throughput improvement. As the block size becomes larger, load balancing gets successively worse and throughput decreases, because declustering creates fragments with sizes increasingly different.

We also anticipate that, with larger block sizes, the number of disks accessed for each stream drops and the total head movement overhead becomes lower. On the other hand, our stride-based allocation scheme ensures that at most two head movements are required per stream on each disk regardless of how small the block size is. This keeps limited the negative effect of access overhead to throughput. Finally, we observe a threshold behavior around $B_d = 1.2 \cdot 10^6$ bytes. This is the maximum amount of data retrieved in one round for each stream and originates from the bit-rate parameters used during encoding (Table 3). Effectively, beyond this point there is no declustering.

These observations are also verified by Figure 12, that shows the average difference between the access times of the most and least loaded disk in each round. Since the reserved access time of the most heavily loaded disk is typically 99% in each round, the plots in Figure 12 essentially indicate the access time requirements of the least loaded one. It is remarkable how the shape of the plots is reflected to those tracking the system throughput in Figure 11. We conclude that even the least loaded disk is expected to remain more than 80% utilized under deterministic replica placement with no declustering (equivalently, with declustering block size larger than $1.2 \cdot 10^6$).

In summary, declustering is only worthwhile with small declustering block sizes, and its overall benefit is found to be limited in the media streaming case (less than 3% with eight disks). Moreover, the throughput of random replica placement never exceeds that of deterministic.

8 Discussion

We considered data replication and bandwidth allocation schemes that allow tolerating single disk failures in disk arrays storing variable bit-rate streams. When using simple schemes for reserving disk bandwidth, more than half of the maximum achievable throughput is wasted during normal (i.e. no failure) operation. Instead, using the minimum reservation scheme for accommodating a single disk failure results only in throughput reduction of less than 20% at disk array sizes sixteen or larger.

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 disk array size becomes larger. This keeps the bookkeeping overhead limited and preserves the scalability of our method when stream data are striped across large disk arrays.

In previous work, we found that striping data using fixed-size blocks achieves lower throughput than when using variable-grain striping [2]. The backup replica declustering should not be confused with fixed-size block striping, since the primary data still use variable-grain striping. This maintains some benefit from multiplexing requests of different transfer sizes in each round, and absorbs correlations that otherwise would create maximum requirements much higher than the average.

Provisioning for VCR functionality is an important issue that we don't consider extensively in the present paper. In general, such flexibility would require deallocation of previously reserved resources, when a stream playback is suspended or stopped earlier than its normal termination. This can done in a straightforward way, when accumulating disk access delays separately for primary and backup data replicas, as was already described above.

The techniques we presented here could be extended in straightforward ways for handling multiple disk failures. That would require 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 groups [9, 15].

9 Related Work

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 show that mirrored disks can improve I/O performance in addition to providing enhanced reliability [8]. Hsiao and DeWitt describe chained declustering that replicates each database relation on two consecutive disks, while the workload is balanced across the system using a static load balancing algorithm [21]. Merchant and Yu propose using different stripe sizes for different data replicas [23]. Thus, system operation can be efficient with both small transaction requests and ad hoc queries on large parts of a relation.

In our previous work, we found the throughput measured with disk striping of variable bit-rate streams to increase linearly as a function of the number of disks [1, 2]. We also described several design decisions of our server prototype implementation [3]. The system throughput is further improved when the disk bandwidth requirements of individual streams are smoothed across different playback rounds [4], and high disk bandwidth utilization is achieved across both homogeneous and heterogeneous disks. System reliability is a crucial issue when building infrastructure for commercial services. Addressing this issue creates a strong case for storage of variable bit-rate streams, and makes the results of the present paper indispensable part of our previous published work.

The related work from media server research is mostly focused on fault-tolerance techniques when striping constant bit-rate streams [5, 6, 32]. Disks are grouped into clusters, and data blocks from separate disks in each cluster are combined with a parity block to form parity groups. The blocks of a parity group are considered to be retrieved and transmitted in one or multiple rounds, and the parity blocks are stored on data disks or dedicated parity disks. For improving overall efficiency, certain data blocks are not transmitted in a transition period following a disk failure.

Ozden et al. propose reading ahead the data blocks of an entire parity group prior to their transmission to the client [25]. 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 [20, 25]. The dynamic reservation scheme that they introduce minimizes the extra bandwidth that has to be reserved on a disk for reconstructing failed-disk data blocks.

Gafsi and Biersack compare several performance measures of alternative data-mirroring and parity-based techniques for tolerating disk and node failures in distributed video servers [15]. 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. The wasted throughput is critically reduced with the minimum reservation scheme that we propose here.

Tewari et al. study parity-based redundancy techniques for tolerating disk and node failures in clustered servers [30]. 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 replicate data blocks across non-intersecting permutations of disk groups [14]. Multiple available data blocks can be used for dynamic balancing of disk bandwidth utilization across different devices. Instead, Birk examines selectively accessing parity blocks of video streams for better balancing the system load across multiple disks [7].

For failures in video servers supporting variable bit-rate streams, Shenoy and Vin apply lossy data recovery techniques that rely on the inherent redundancy in video streams rather than error-correcting codes. Alternatively, they propose taking advantage of the sequential block accesses during playback and reconstructing missing data from surrounding available blocks, at the cost of an initial playback latency, or temporary disruption when a failure occurs [29].

Bolosky et al. decluster the block replicas of one disk across d other disks. In case of disk failure, the extra bandwidth required for retrieving the data of the failed disk is shared among the d other disks [9]. In later work, they also consider providing fault-tolerant support for multiple streams with different bit rates [10]. In our experience, declustering does not add significant improvement with respect to the case of replicating the data blocks of one disk in their entirety on different disks.

Mourad describes the doubly-striped disk mirroring technique that distributes replica blocks of one disk round-robin across the rest of the disks [24]. The system load is equally distributed across the surviving disks in case of a disk failure. The deterministic replica placement that we describe extends doubly-striped mirroring for handling variable bit-rate streams and the reduced device utilization that they potentially introduce.

Santos et al. compare disk striping against data replication on randomly chosen disks [27]. Using constant bit-rate streams, they conclude that random replication can outperform disk striping with no replication. In our comparison using variable bit-rate streams instead, we found an advantage of deterministic replication over random replication that diminishes as the number of disks increases.

10 Conclusions

We studied issues related to data replication of variable bit-rate streams striped across multiple disks for improving system reliability and performance. We introduced the *minimum reservation scheme* that minimized 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. Deterministic placement of backup data is found to achieve better performance than random placement across the different disks, although the advantage becomes insignificant as the number of disks increases. Retrieving the data replica of each stream stored on the least loaded disk adds an improvement of no more than 10% with respect to the non-replicated case. Finally, declustering the backup replicas across multiple disks does not seem to considerably improve the performance achieved with deterministic replica placement.

References

[1] Anastasiadis, S. V. *Supporting Variable Bit-Rate Streams in a Scalable Continuous Media Server.* PhD thesis, Department of Computer Science, University of Toronto, June 2001.

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

[3] Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. Modular and Efficient Resource Management in the Exedra Media Server. In *USENIX Symposium on Internet Technologies and Systems* (San Francisco, CA, Mar. 2001), pp. 25–36.

[4] Anastasiadis, S. V., Sevcik, K. C., and Stumm, M. Server-Based Smoothing of Variable Bit-Rate Streams. In *ACM Multimedia Conference* (Ottawa, ON, Oct. 2001), pp. 147–158.

[5] Berson, S., Ghandeharizadeh, S., Muntz, R., and Ju, X. Staggered Striping in Multimedia Information Systems. In *ACM SIGMOD Conference* (Minneapolis, MN, May 1994), pp. 79–90.

[6] Berson, S., Golubchik, L., and Muntz, R. R. Fault Tolerant Design of Multimedia Servers. In *ACM SIGMOD Conference* (San Jose, CA, May 1995), pp. 364–375.

[7] Birk, Y. Random RAIDs with Selective Exploitation of Redundancy for High Performance Video Servers. In *International Workshop on Network*

and Operating System Support for Digital Audio and Video (Zushi, Japan, May 1997), pp. 13–23.

[8] Bitton, D., and Gray, J. Disk Shadowing. In *Very Large Data Base Conference* (Los Angeles, CA, Aug. 1988), pp. 331–338.

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

[10] Bolosky, W. J., Fitzgerald, R. P., and Douceur, J. R. Distributed Schedule Management in the Tiger Video Fileserver. In *ACM Symp. Operating Systems Principles* (Saint-Malo, France, Oct. 1997), pp. 212–223.

[11] Chang, E., and Zakhor, A. Cost Analyses for VBR Video Servers. *IEEE Multimedia* (Winter 1996), 56–71.

[12] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. RAID: High-Performance, Reliable Secondary Storage. *ACM Computing Surveys* **26**, 2 (June 1994), 145–185.

[13] General Information on Cable TV and its regulation. Fact Sheet, Federal Communications Commission, June 2000. http://www.fcc.gov/csb/facts/csgen.html.

[14] Flynn, R., and Tetzlaff, W. Disk Striping and Block Replication Algorithms for Video File Servers. In *IEEE International Conference on Multimedia Computing and Systems* (Hiroshima, Japan, June 1996), pp. 590–597.

[15] Gafsi, J., and Biersack, E. W. Modeling and Performance Comparison of Reliability Strategies for Distributed Video Servers. *IEEE Transactions on Parallel and Distributed Systems* **11**, 4 (Apr. 2000), 412–430.

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

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

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

[19] Haskin, R. L., and Schmuck, F. B. The Tiger Shark File System. In *IEEE COMPCON* (Feb. 1996), pp. 226–231.

[20] Holland, M., and Gibson, G. A. Parity Declustering for Continuous Operation in Redundant Disk Arrays. In *ACM Conference on Architectural Support for Programming Languages and Operating Systems* (Boston, MA, Oct. 1991), pp. 23–35.

[21] Hsiao, H.-I., and DeWitt, D. J. Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines. In *IEEE International Conference on Data Engineering* (Los Angeles, CA, Feb. 1990), pp. 456–465.

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

[23] Merchant, A., and Yu, P. S. Analytic Modeling and Comparisons of Striping Strategies for Replicated Disk Arrays. *IEEE Transactions on Computers* **44**, 3 (Mar. 1995).

[24] Mourad, A. Doubly-Striped Disk Mirroring: Reliable Storage for Video Servers. Multimedia Tools and Applications **2** (May 1996), 273–297.

[25] Ozden, B., Rastogi, R., Shenoy, P., and Silberschatz, A. Fault-tolerant Architectures for Continuous Media Servers. In *ACM SIGMOD Conference* (Montreal, Canada, June 1996), pp. 79–90.

[26] Patterson, D. A., Gibson, G., and Katz, R. H. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *ACM SIGMOD Conference* (Chicago, IL, June 1988), pp. 109–116.

[27] Santos, J. R., Muntz, R. R., and Ribeiro-Neto, B. Comparing Random Data Allocation and Data Striping in Multimedia Servers. In *ACM SIGMETRICS Conference* (Santa Clara, CA, June 2000).

[28] *Cheetah 36XL Product Manual.* Seagate Technology, Feb. 2001. http://www.seagate.com/cda/products /discsales/enterprise/family/0,1130,318,00.html.

[29] Shenoy, P. J., and Vin, H. M. Failure Recovery Algorithms for Multimedia Servers. *ACM Multimedia Systems Journal* **8**, 1 (Jan. 2000), 1–19.

[30] Tewari, R., Dias, D. M., Mukherjee, R., and Vin, H. M. High Availability in Clustered Multimedia Servers. In *IEEE International Conference on Data Engineering* (New Orleans, LA, Feb. 1996), pp. 336–342.

[31] Thekkath, C. A., Wilkes, J., and Lazowska, E. D. Techniques for File System Simulation. *Software-Practice and Experience* **24**, 11 (Nov. 1994), 981–999.

[32] Tobagi, F. A., Pang, J., Baird, R., and Gang, M. Streaming RAID - A Disk Array Management System for Video Files. In *ACM Multimedia Conference* (Anaheim, CA, Aug. 1993), pp. 393–400.